Analysis of Spatial Stream Networks for Salmonids

Phase 2: Fish Data Analysis Tool System Architecture Design and Recommendations

Report covers work performed under BPA contract #81134.

Report was completed under BPA contract #81134

Report covers work performed from: March 2019 – May 2020

Erin E. Peterson[1,2], Bryce Christensen[1], Alan Woodley[1], Samuel Smith[1] and Dan Teleki[3]

[1] Erin Peterson Consulting, Brisbane, QLD, Australia (erin@peterson-consulting.com)

[2] Queensland University of Technology, Brisbane, QLD, Australia

[3] DTGeoinformatics, Edgewater, British Columbia, Canada

Report Created: May 4, 2020

Executive Summary

Significant investments have been made by the Bonneville Power Administration (BPA) and numerous partner organizations to monitor anadromous fish populations and restore habitats within the Columbia River Basin in previous decades. As a result, juvenile and adult fish counts now exist at hundreds to thousands of sites throughout the region. These counts are valuable for answering status and trend questions that originally motivated collection of the datasets, but they can also be integrated within a statistical modeling framework, at a relatively low cost, to better understand habitat constraints and make predictions of fish abundance and distributions at regional scales.

The Fish Data Analysis Tool (FDAT) is designed to incentivize the development of consistent, centralized databases and automate the process of network-scale modeling of fish density and abundance datasets. FDAT includes several useful features, including the ability to:

1. Visualize existing fish density data collected by multiple organizations in space and time;
2. Import fish survey data uploaded by users, who interactively quality assurance quality control (QA/QC) the spatial location of survey sites;
3. Automatically undertake the extensive spatial-data processing steps needed to fit spatial statistical stream-network models;
4. Fit spatial statistical stream-network models and generate predictions at multiple management-relevant scales (e.g., site, population, and/or catchment), with associated estimates of uncertainty;
5. View prediction maps of fish densities and total abundances throughout full stream networks and subsets of the network;
6. View interpolated maps of the uncertainties in fish densities that could guide subsequent data collection efforts to locations that would most efficiently reduce uncertainty; and
7. Download the pre-processed spatial data used to fit the models, as well as the model predictions and a report documenting the methods used to generate them.

In addition, FDAT provides the functionality to monitor data upload activity by registered users, which the BPA could tie to contractual payment milestones.

The aim of this project was to outline the functional requirements and provide a detailed software architecture design for a web-based FDAT, used to apply spatial statistical stream-network models in an automated, computationally efficient, and consistent manner to anadromous fish datasets across the whole-of-the Columbia River Basin. While only one architecture design is provided in Section 5, we review a range of technical software including cloud-based hosting solutions, as well as open-source and proprietary software that could be

used for particular FDAT system components. We discuss the pros and cons of each software choice and make recommendations for future development and implementation.

Spatial Data Processing

Numerous spatial data-processing steps take place behind the scenes in FDAT to ensure the field data contributed by users is in a format suitable for spatial stream-network modeling. In the prototype FDAT developed in Phase 1 of the project (FY18), the spatial data processing, modeling and prediction were undertaken in two steps using both proprietary (ESRI ArcGIS and Microsoft Access database) and open-source software (R Statistical Software). We created the osSTARS package for R in this project, which replicates the core spatial-data processing steps needed to support the FDAT. This alleviates reliance on proprietary software subscriptions, and reduces future time and costs associated with system maintenance. As a result, all of the spatial data processing, statistical modeling and prediction takes place in R, creating a seamless data-analysis pipeline within one open-source software product.

Mapping Library and Provider

We considered five open-source (CesiumJS, Mapbox GL JS, Leaflet, and OpenLayers) and proprietary (ArcGIS API JavaScript) mapping libraries to support the web-based interactive mapping functionality of FDAT. After careful consideration, we recommend CesiumJS and/or MapBox GL JS mapping libraries because they are completely customizable and perform particularly well for visualizing, analyzing and sharing large spatial datasets. Both are freely available, open-source code libraries that have been in development for many years and have large, active online communities.

We also reviewed four popular mapping providers, three of which were proprietary (Cesium ion, ArcGIS, and Mapbox) and the other open-source (OpenStreetMap) software. We recommend using Mapbox, a proprietary map provider because of the large number of basemaps to choose from, as well as a reasonable, usage driven pricing scale, with a free quota. This means there will be no map provider costs incurred during the FDAT development, and potentially throughout the life of FDAT.

Database

We recommend the open-source database software PostGreSQL because of its ability to handle massive datasets, support for geospatial queries, its robust and rapid data-storage mechanism, and the ability to import and export common spatial data formats. We believe that this software is the best option and that there are no benefits to be gained from using a proprietary software product.

Hosting Environment: Cloud-base versus on-premises hosting

The most important decision concerning the FDAT architecture is whether to host it on the BPA network (i.e., on-premises hosting) or on a cloud service provider such as Amazon Web Services (AWS), Google Cloud, or Microsoft Azure. We recommend cloud-based hosting over on-premises hosting because:

- More cost-effective;
- Allows for rapid scalability and on-demand computing;
- No upfront hardware costs;
- Cloud vendor maintains hardware and software updates and patches;
- Automated backup and recovery; and
- Cloud vendor handles security.

Cloud-based hosting cost estimates on Amazon Web Services

The cost of cloud-based hosting is affected by the:

- Number of users accessing data (e.g., downloading, clicking on map features to generate data summaries, and uploading data);
- Frequency of spatial data processing and/or modeling (e.g., daily
- versus annually); and
- Changes in data storage needs, which could occur if there were large increases in observations or predictions, or a new variable was added (e.g., temperature).

Based on the assumptions outlined in Section 3, the cost estimates for cloud-based hosting are:

- Expected use: $4995 per year, or $14,985 over three years.
- Exceeds usage expectations or modeling occurs more than 5 times per year: $8095 per year or $24,285 over three years.
- Significantly exceeds usage expectations and modeling occurs on demand: $8100-$10,000 per year and $24,000-30,000 over three years.

These costs are provided only as a guide for discussion, rather than a quote, and are accurate as of April, 2020.

BPA on-premises hosting cost estimates

We have provided on-premises hosting costs, but note that they are difficult to estimate without detailed knowledge of hardware purchasing providers, staff costs and the existing network infrastructure. Assuming that hardware is replaced every three years and there are no hardware failures, the cost estimate for on-premises hosting is:

- Expected usage: $21,900 over three years, with $13,700 in the first year, and an additional $4,100 in the second and third years for system maintenance.

- Exceeds, or significantly exceeds usage expectations and modeling occurring on demand: $43,800 over three years, with $27,400 in the first year, and an additional $8,200 per year in the second and third years for system maintenance.

While the cost of implementing, hosting and maintaining FDAT is not insignificant, we believe that it provides long-term value for money. Enormous field sampling and habitat restoration efforts have occurred in the Columbia River Basin, but existing data are often housed within different organizations and/or stored in disparate formats, making data integration at regional scales challenging. Analyzing data from multiple sources can also be a technical barrier for users who would benefit from basin-scale assessments, given the advanced skills in aquatic biology and ecology, spatial science, and computational modeling and statistics needed to fit spatial statistical stream-network models, and other comparable models. FDAT addresses many of these needs.

- It is designed to incentivize the creation, standardization, use, and maintenance of a central fisheries database across multiple agencies, which is freely available to the public and continually updated each year.
- It automates the process of network-scale modeling of fish density and abundance datasets collected by multiple organizations.
- The predictive model outputs provide new information about current status and future trends mined from existing datasets, which are also freely available for download.
- Analysis methods are well documented and reproducible, to promote transparency and provide users with the confidence in the results.

While FDAT has been designed for fish density and abundance data, the software architecture was designed to be cost-effective, secure, scalable and robust as FDAT evolves. Thus, it can also be used to integrate and model other stream variables such as temperature, with relatively minor modifications.

Table of Contents

1 Introduction

Significant investments have been made by the Bonneville Power Administration (BPA) and numerous partner agencies to monitor anadromous fish populations and restore habitats within the Columbia River Basin in previous decades. Spatially indexed juvenile salmon and steelhead/rainbow trout counts now exist at hundreds to thousands of sites throughout the region. These counts are valuable for answering status and trend questions that originally motivated collection of the datasets, but they can also be repurposed at low cost for use with other spatial datasets related to stream environments (e.g., slope, elevation, land use) and spatial stream-network models (Ver Hoef et al. 2006; Ver Hoef and Peterson 2010) to understand habitat constraints and make predictions of fish abundance and distributions throughout full river networks (Isaak et al. 2017). Predicted abundances can be integrated over the full network or subdomains within the network to obtain population estimates at a variety of spatial and/or temporal scales that are most relevant to conservation and investment planning. Moreover, spatial statistical stream-network models differ from previous models in that they do not require spatially independent samples, but instead benefit from some degree of non-independence among samples in close proximity. This flexibility allows the models to make use of data from multiple sources and incentivizes the development of consistent, centralized databases because the models become increasingly accurate as the amount of data and site clustering in modeled datasets increase.

In Phase 1 of the Analysis of Spatial Stream Networks for Salmonids project (BPA contracts #77234, 77246, and 46273 REL 13), we developed an automated web-based prototype Fish Data Analysis Tool (FDAT) to demonstrate the process of network-scale modeling of fish density datasets (Peterson et al. 2018). The prototype FDAT was based on data from a small study area in northeastern Oregon and included several useful features:

1. Visualize existing fish density data collected by multiple organizations in space and time (Figure 1a);
2. Import fish survey data uploaded by users, who interactively QA/QC the spatial location of survey sites (Figure 1b);
3. Automatically undertake the spatial-data processing steps needed to fit spatial statistical stream-network models (Figure 1c);
4. Fit spatial statistical stream-network models and generate predictions at multiple management-relevant scales (e.g., site, population, and/or catchment), with associated estimates of uncertainty (Figure 1d);
5. View prediction maps of fish densities and total abundances throughout full stream networks and subsets of the network (Figure 1a);
6. View interpolated maps of the uncertainties in fish densities that could guide subsequent data collection efforts to locations that would most efficiently reduce overall uncertainty (Figure 1a); and

7. Download the pre-processed spatial data used to fit the models, as well as the model predictions and a report documenting the methods used to generate them (Figure 1a).

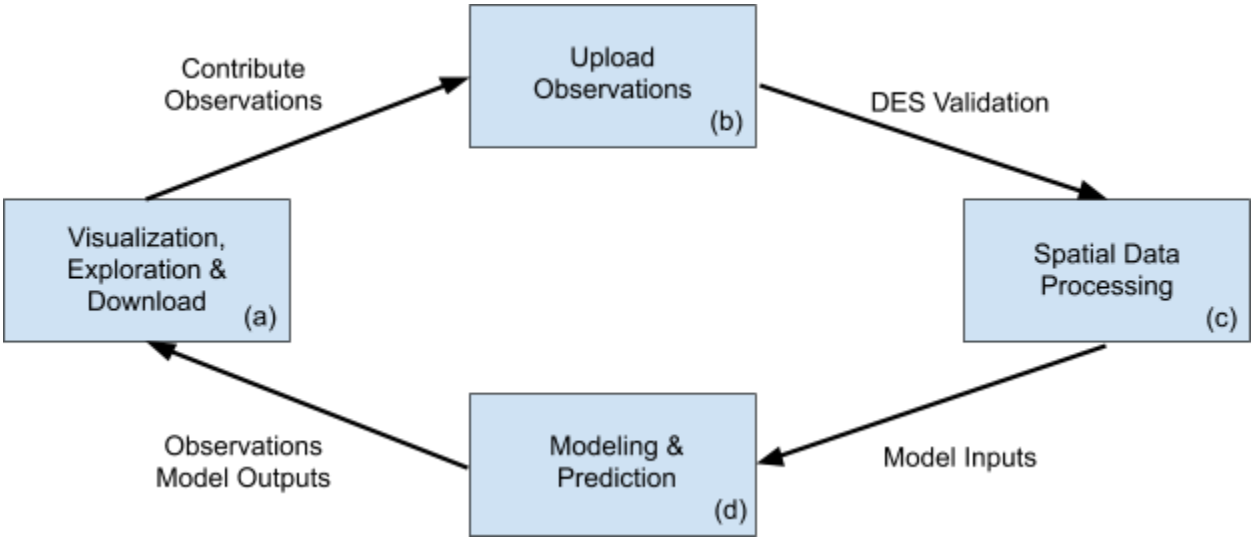

Figure 1. Functional overview of the four main components of FDAT, including (a) Visualization, Exploration, & Download, (b) Upload Observations, (c) Spatial Data Processing, and (d) Modeling and Prediction.

While the Phase 1 prototype enabled us to demonstrate the feasibility and advantages of an automated analysis approach, it was not optimally designed for analyzing and visualizing large sets of field observations and predictions at the Columbia River Basin scale. In addition, the prototype FDAT made use of proprietary software, which would increase the cost of maintaining FDAT in the future.

Here we provide a software architecture option for a web-based FDAT, used to apply spatial statistical stream-network models in an automated, computationally efficient, and consistent manner to anadromous fish datasets across the whole of the Columbia River Basin; with the ultimate goal of displaying quantitative field results and modeled abundance and distribution information to guide salmonid resource management and habitat restoration, as well as protection actions. We review a range of technical software including cloud-based solutions, as well as open-source and proprietary software. We discuss the pros and cons of each approach and make recommendations for future development and implementation.

2 Recommendations

The FDAT is an internet-accessible web application designed to ingest, process, and model field data (i.e., observations) collected in river systems (Figure 1). We provide a detailed description of the proposed FDAT system architecture in Section 5. All efforts have been made to consider the most cost-effective, secure, scalable and robust software architecture, while future-proofing the design so that it can be applied to other variables (e.g., temperature) with minor modifications. However, there are a number of decisions that must be made regarding the application hosting environment and the use of proprietary versus open-source software. We provide recommendations about those choices below, followed by cost estimates for implementation (Section 3).

2.1 Application Environment

The most important decision the BPA must make concerning the FDAT architecture is whether to host it on the BPA network (i.e., on-premises hosting) or on a cloud service provider such as Amazon Web Services (AWS), Google Cloud, or Microsoft Azure. There are pros and cons to both approaches (Figure 2; Leoni 2019), but we recommend cloud hosting over on-premises hosting for a number of reasons.

A cloud hosting approach provides a single point of security management, government approved security levels, and highly granular internal security roles and restrictions. More importantly, it provides on-demand processing and scalability, as the demand and scope of FDAT evolves (Leoni 2019). For example, the spatial data processing, modeling, and prediction functions in FDAT (Figure 1) are computationally intensive and require significant hardware resources (Section 3) to ensure processes finish in a reasonable amount of time. However, these processes only run intermittently (e.g., daily, monthly, seasonally, or annually) and so the server sits idle most of the time. In a cloud computing environment, there are no up front hardware costs (Figure 2). The user only pays for the computation time on the server and computational resources can be easily adjusted up or down depending on the demand and number of users. All servers, databases and underlying file resources are automatically backed up, and the entire system can be restored or redeployed from backup. In addition, hardware and software upgrades are handled by the cloud vendor. However, it is worth noting that cloud hosting costs can escalate if the application is not properly designed and managed (Leoni 2019).

An on-premises deployment is much less scalable and agile than cloud hosted applications, but the user has full control over hardware and security (Figure 2; Leoni 2019). For example, organizations pay the full purchase price for servers deployed on-premises up front, regardless of how often they are used. Servers must also be updated every 3 to 5 years, with additional hardware purchases required to scale up computation, and no option to downscale

computational resources. Another important cost to consider is BPA information technology (IT) staff time required for hardware purchases, software updates, ongoing operating, and maintenance. However, users may choose to host applications on-premises when data are particularly sensitive, or there are specific security and compliance requirements because they have full control over security and data access. We do not believe that this is the case for FDAT, which is designed to promote data sharing and transparency between multiple users and organizations. Even if FDAT data were deemed sensitive, there are cloud hosting options, such as AWS GovCloud, which are specifically designed to comply with the most stringent US government security and compliance requirements (Amazon Web Services 2020a).

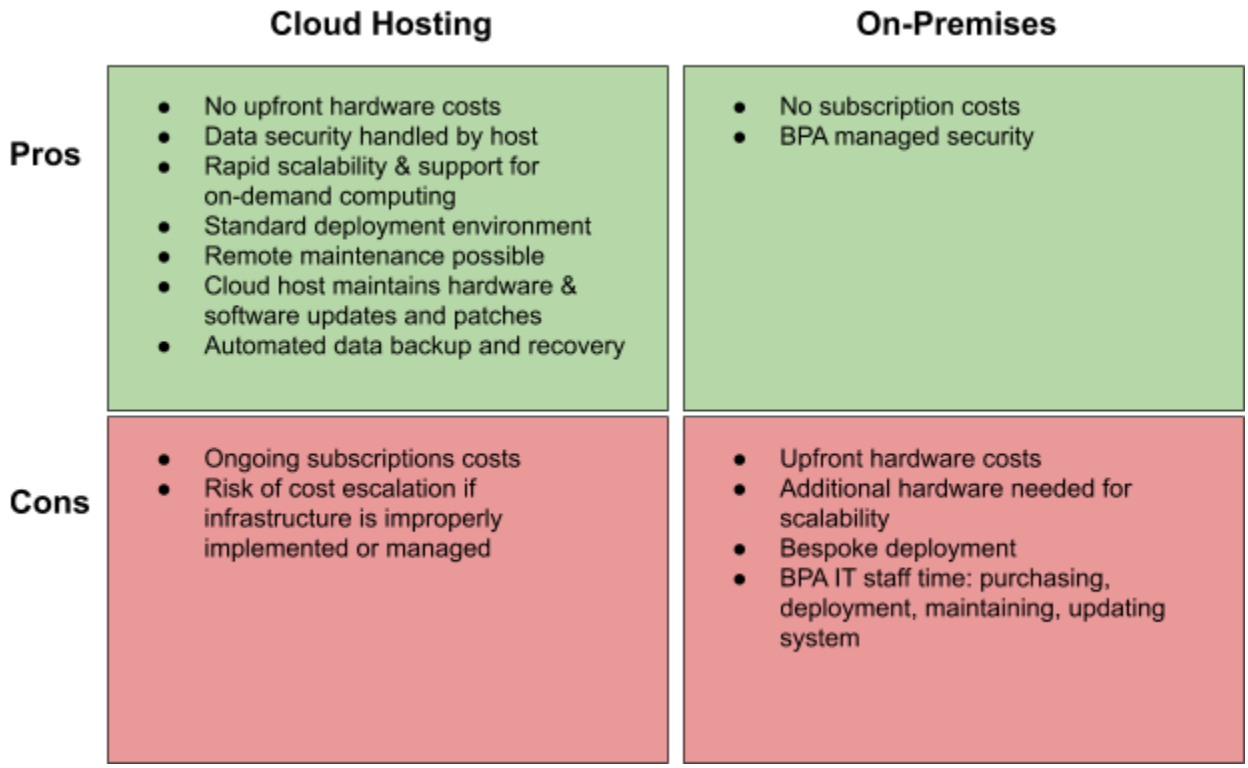

Figure 2. Pros and cons of FDAT cloud-based hosting versus hosting on the Bonneville Power Administration (BPA) network (On-Premises).

In recent years, many large web-based systems have been migrated to cloud-based services (Amazon Web Services 2020b); in part because a well-planned and managed cloud hosting solution provides a more cost effective and financially sustainable outcome than traditional on-premises hosting. While any cloud hosting provider could be used for FDAT, we recommend AWS because it is the most mature cloud vendor and holds the majority of the market share

globally. In addition, our team's personal experience building internet-accessible web applications on AWS, underpinned by large database instances and on-demand computing to support modeling and spatial visualizations (e.g., Virtual Reef Diver 2020; ILAQH 2020), confirms that this cloud vendor is a suitable host for FDAT.

2.2 Data Exchange Standard

The BPA and their partner organizations are currently in the process of developing a data exchange standard (DES) to make data sharing between organizations more efficient. A consistent data format is especially important in FDAT, where the aim is to automatically ingest, analyze, and visualize data contributed by multiple organizations throughout the Columbia River Basin. We received a copy of the draft DES in Excel spreadsheet format from the BPA (Appendix 1) and have used it to design the data ingestion process in FDAT. However, this portion of the design can easily be altered, as long as partners adhere to a common format.

Two clear advantages to using a DES in Excel spreadsheet format are that the software is accessible, and the format familiar to most users. However, there are numerous issues related to the use of spreadsheets. Users often introduce errors related to data integrity and consistency in spreadsheets, where data of different formats (e.g., character, date, or number) can be accidentally entered into a single column. When formulas embedded in spreadsheets have errors, even small errors can have significant financial consequences for organizations and policy implications (Soto 2019). Recent evidence also suggests that unanticipated automatic formatting in Excel (e.g., shortening column names or re-formatting data in cells) has introduced errors to approximately 20% of genetic studies published in top scientific journals (Ziemann et al. 2016). In addition, the use of spreadsheets may also inadvertently increase the time required for data entry. Although different partner organizations often collect similar information in field studies (e.g., species, age class, count, or area sampled), they tend to have an existing data entry format they prefer. As a result, some partners will continue to record data in their own preferred format before re-entering the data in the DES spreadsheet; thus, increasing the amount of time needed for data entry and increasing the probability of data entry errors.

There are alternative data formats that could be used to address these issues. For example, a web page could be created that provides the inputs present in the Excel spreadsheet, with drop down menus for those fields with a set number of options. Validation could be done on the fly, so that users can instantly see if a field has been filled incorrectly, and also when the user attempts to submit the web form. Submitted data can then be automatically stored in a database, which alleviates the need to re-enter data in multiple formats, assuming the data are valid. Such a solution could be embedded in an app that initially stores the data in a local database and then submits it to the BPA servers when there is an internet connection present.

A form embedded in a web page or phone app (or both) could also be designed to be user friendly for mobile users who lack a physical keyboard and mouse.

2.3 Database Instance

We reviewed a number of open source and proprietary database providers (Appendix 2), any of which could meet the needs of FDAT. However, we recommend the open source software PostGreSQL because of its ability to handle massive datasets (Mapbox 2015), support for geospatial queries (Planet 2012), its robust and rapid data-storage mechanism (Citus Data 2017), and the ability to import and export common spatial data formats such as shapefiles, GeoJSON and scalable vector graphics (SVG; Planet 2012). This choice is also supported by our own experience developing similar data-driven environmental sensing applications such as the KOALA Air Quality monitoring project (ILAQH 2020). PostGreSQL has efficiently handled more than 9 million rows of data to date and successfully provides geospatial query support through the PostGIS extension (PostGIS 2020). We believe that PostGreSQL is more suitable than the other open source and proprietary software options we considered and as such, we have not provided a recommendation for proprietary database software.

2.4 Mapping Library and Provider

There are two components to consider when implementing the FDAT visualization component (Figure 1d). One is the mapping library, which is the source code library used to render the mapping functionality in the web browser. The mapping library connects to an underlying mapping provider, which supplies basemaps such as satellite imagery, a simple street map, or a customized topographic map. We reviewed a variety of open source and proprietary map libraries and providers and compared their pricing, performance and overall suitability for use in FDAT (Appendix 3).

We considered five open-source (CesiumJS, Mapbox GL JS, Leaflet, and OpenLayers) and proprietary (ArcGIS API JavaScript) mapping libraries for use in FDAT (Appendix 3, Table A3.1). OpenLayers and Leaflet were deemed unsuitable due to performance issues with large datasets (Netek et al. 2019). In contrast, ArcGIS API for JavaScript (ESRI 2020a) is an advanced web mapping application fully capable of visualizing and analyzing massive datasets online (Ekenes 2020). It provides cartography tools and widgets to support custom mapping and GIS analyses (Powell et al. 2020), which makes it relatively straightforward for most users to build interactive maps. This is a substantially different approach than other mapping libraries such as CesiumJS (Cesium 2020a), which is designed for use by developers who have the technical skills needed to customize every aspect of the map (Stähli 2017; Stepnov 2019). This makes it difficult to switch from ArcGIS API for JavaScript to other mapping libraries. In addition, it is proprietary software and thus, adds to the maintenance costs (Table A3.1). After careful

consideration, we recommend CesiumJS (Cesium 2020a) and/or MapBox GL JS (Mapbox 2020a) mapping libraries because they are completely customizable and perform particularly well for visualizing, analyzing and sharing large spatial datasets in interactive web-based maps (e.g., Mapbox 2013; Seeger 2018). Both are freely available, open-source code libraries that have been in development for many years and have large, active online communities. While CesiumJS and MapBox GL JS are both widely used, Mapbox GL JS is by far the most popular library (Potter 2020). Note that, both CesiumJS and Mapbox GL JS were used in FDAT Phase 1 (Peterson et al. 2018), with CesiumJS used for the mapping interface and Mapbox GL JS for the site snapping interface and functionality (Section 4.3.2). In addition, we have used CesiumJS and Mapbox GL JS to develop other large-scale, web-based mapping applications such as Virtual Reef Diver (2020), the RAISE toolkit (QUT Design Lab 2020) and the Queensland Land Change tool (McLaughlin et al. 2019; Woodley et al. 2019) and in our experience they perform well.

We reviewed four popular mapping providers, three of which were proprietary (Cesium ion, ArcGIS, and Mapbox) and the other open-source (OpenStreetMap) (Appendix 3, Table A3.2). Although OpenStreetMap (OpenStreetMap Contributors 2020) is free to use, it does not provide satellite imagery, which is an appealing basemap option in rural and remote areas. Cesium ion (Cesium 2020b) could also be used, but it is primarily designed for 3D data streaming services rather than 2D mapping, and so much of the functionality would not apply to FDAT. A wide range of ESRI basemaps are provided with the ArcGIS API JavaScript Builders License (Table A3.2), but the subscription price is fixed at $125 per month (ESRI 2020b). Instead, we recommend the MapBox provider service. There are a large number of basemaps to choose from including satellite imagery, terrain maps and street maps (Mapbox 2020b) and it has a reasonable, usage driven pricing scale, with a free quota (Mapbox 2020c). This means there will be no provider costs during the FDAT development phase, and potentially throughout the life of FDAT (Appendix 3).

3 Cost Analysis

In this section we outline the minimal requirements and cost of FDAT, based on our understanding of the estimated storage, growth and use of the application. In the event the application is more widely used, or grows in scope to include areas or variables, the costs will increase accordingly. The costs are accurate as of April 2020, but also stress that they are provided only as a guide for discussion, rather than a quote.

3.1 Assumptions

Several assumptions about the frequency of use, scale of the underlying datasets, and the data access by public users underpin the FDAT cost analysis and are described in more detail below.

3.1.1 Data

The spatial extent of FDAT Phase 3 includes at a minimum, the entire Columbia River Basin. There are approximately 6500 fish observations available in FDAT Phase 2 and we expect there to be another 6000 existing observations in Phase 3, when the extent includes the Interior of the Columbia River Basin. Prediction sites represent discrete point locations on the stream where field sampling has not occurred and thus, predictions will be made. The total number of prediction sites is determined by the number of species, age classes, years, suitable habitat and climate scenarios, as well as the spacing between prediction sites on the stream. In FDAT Phase 2, predictions are set at 250 meter intervals and there two species (steelhead/rainbow trout and Chinook salmon). At present, there are approximately 77,000 steelhead/rainbow trout and 39,000 Chinook salmon prediction sites and we expect these numbers to double in Phase 3. Therefore, we based our resource requirements and the architectural requirements for the application design on the following estimates for spatial dataset file sizes and number of spatial features.

- Stream network file size: 78MB
- Number of prediction point locations: 113,000
- Prediction points file size: 156MB
- Number of Chinook salmon observation locations: 13-15,000
- Number of steelhead/rainbow trout observation locations: 14-17,000

We expect that an additional 500-700 observations will be uploaded to the FDAT system each year for both steelhead/rainbow trout and Chinook salmon. In addition, predictions will be made for each year at the point, population, and catchment scale, which are stored in a database instance and as .csv files for download by users.

3.1.2 Modeling Frequency

The FDAT is designed to automatically fit spatial statistical stream-network models to user uploaded data at set intervals determined by the BPA. While this could occur at any time (e.g., daily, monthly seasonally, or annually), we have assumed that it will happen once per year.

Thus, the cost estimates for the FDAT architecture assume a 5% total usage time for the server used for modeling and prediction (i.e., RScript Server; Section 5.1.2). This should provide ample

time for annual modeling, as well as occasional on-demand modeling desired by BPA partners. If there is a requirement for more frequent predictive modeling, the on-demand costs for the RScript Server would also increase accordingly.

3.1.3 FDAT Users

We have assumed that there will be less than 50 BPA administrators and registered users uploading data to FDAT in total, with fewer than 10 uploading data simultaneously at peak times. The other consideration is the public users, who we have less ability to predict. We assume that as awareness of FDAT grows, the number of users will increase over time. While we considered on-demand scalability, redundancy, and system performance in the design, we have assumed there will be no more than 1000 unique public users per month. This estimate is based on the NorWeST website[1], which has approximately 600 visitors per month (D. Isaak, personal communication).

3.1.4 Domain Name and SSL Certificate

We suggest that the BPA provide the domain name for FDAT and register it through a provider of their choice, to ensure that there are no future issues with ongoing registration costs or loss of domain registrar contact information. Domain name costs are typically in the order of $10 per year. However, the choice of hosting environment will affect this cost. For example, the SSL certificate could be provided for free as part of the AWS cloud computing solution, or supplied by the BPA through a SSL certificate provider at their own cost.

3.2 Cloud Computing

For the purposes of this proposed architecture, we have focused on the cloud computing provider AWS, but note that comparable services are offered through Google Cloud Compute and Microsoft Azure; although they use different naming conventions and there are implementation details specific to each provider. The following hardware specifications have been identified as suitable for the FDAT system as allowing reasonable operation under the estimated user load, while balancing lower costs with reasonable redundancy and failover systems.

[1] NorWeST website: https://www.fs.fed.us/rm/boise/AWAE/projects/NorWeST.html

3.2.1 Specifications

Web Server

AWS ‘t3.large’ Hardware Specifications

- 2 x vCPUs
- 8GB RAM
- 5GBps network burst speed
- Elastic load balancer up to 2780 MBps
- 2 x 30GB SSD storage

R Script Server

AWS ‘m5.2xlarge’ Hardware Specifications

- 5% utility / uptime (on demand)
- 8 x vCPUs
- 32GB RAM
- 10GBps network burst speed
- Elastic load balancer up to 4750 MBps
- 2 x 30GB SSD storage

File Storage

AWS utilizes a highly available and redundant storage framework called S3 storage. The nomenclature for a file folder is known as a ‘bucket’, which operates in a similar manner to a normal file folder. The S3 bucket can replicate files to multiple regional servers to improve delivery speed and reduce latency.

S3 File bucket allocation

- 10GB storage
- 100,000 PUT/COPY/POST/LIST Requests
- 100,000 GET/SELECT and Other Requests

Database Instances

We assume that a PostGreSQL relational database system (RDS) will be used. The AWS equivalent of a pre-configured RDS instance type is db.r5.large, which is an ‘R’ series fixed performance instance, providing consistent performance suitable to a web delivery architecture. The db.r5.large RDS instance provides the following hardware specifications:

- 20GB of storage;
- 1 million input/output requests per month;
- 1 CPU Core;

- 2 vCPUs;
- 16GB RAM;
- Up to 3500 MBps dedicated bandwidth;
- Up to 10GBps network performance;
- Up to 3.1 GHz Intel® Xeon® Platinum 8000 processor; and an
- Intel AVX, Intel AVX2, and Intel Turbo.

3.2.2 Cost Estimate

The ability to scale the hardware resources up or down to suit the application demand allows for real-time cost optimization, with consideration for the application performance. Factors which could increase costs, and potentially require additional hardware allocations include changes to the:

- Number of public users accessing data (e.g., downloading, clicking on map features to generate data summaries, and uploading data);
- Frequency of spatial data processing and/or modeling (e.g., daily versus annually);
- Data storage needs, which could occur if there were large increases in observations or predictions, or a new response variable was added (e.g., temperature); and
- Application performance after user testing.

If FDAT is used less than we anticipate or the requirements for model outputs are reduced, then the hardware resources can also be scaled down, decreasing costs.

We estimate the minimum annual cost to deploy FDAT on AWS to be $4995 ($14,985 over three years) based on the assumptions about system usage, system demand, and storage needs (Table 1). If FDAT usage exceeds expectations or modeling occurs more than five times per year, a replica RDS service may be needed to improve performance and responsiveness; meaning that FDAT would effectively be running two synchronized databases to service modeling and web-based user queries (Section 5.3.1). In this case, the replica database (Amazon RDS Service, Table 1) would double in price, and the annual cost would be $8095 ($24,285 over three years). In the event that FDAT usage far exceeds usage expectations, we estimate that the cost for the upper end in performance, scale and responsiveness would be between $8,100 and $10,000 per year ($24,300- $30,000 over three years).

Table 1. Minimum annual cost to deploy FDAT on Amazon Web Services (AWS), broken down by service type. A three-year cost estimate is also provided for comparison with on-premises hosting costs.

Service	Annual Cost (USD)	3-Year Cost (USD)
Amazon EC2 Service	1,405	4,205
Amazon S3 Service	10	30
Amazon RDS Service	3,100	9,300
Amazon Elastic Load Balancing	275	825
AWS Transfer In	0	0
AWS Transfer Out	205	615
AWS Support	0	0
Total	4,995	14,975

3.3 On-Premises Hosting

The cost estimates for on-premises hosting by BPA are based on the assumptions described in Section 3.1. However, the estimate provided in Table 2 should only be used as a rough guide to costs. To obtain a more detailed on-premises costing, the following considerations should be made by the BPA:

- Whether existing hardware is suitable for FDAT hosting;
- BPA Network and System Administrator staff costs;
- Ongoing patch, security and software maintenance for operating systems and database instances;
- Internal software licensing costs for operating systems;
- Cost to upgrade the hardware over the life of FDAT, while also allowing for replacement if hardware failure occurs;
- Server redundancy and potential to scale up additional servers and database instances as needed, depending on FDAT usage;
- Internal and external network traffic costs, load balancing, distributed denial of service (DDoS) protection and intrusion detection; and
- Domain name and SSL certificate registration and maintenance.

3.3.1 Specifications

The hardware specifications are equivalent to the AWS hardware specifications provided in Section 3.2.1.

Web Server - Linux (equivalent AWS ‘t3.large’)

- 100% utility / uptime
- 2 x vCPUs
- 8GB RAM
- 5GBps network burst speed
- Elastic load balancer up to 2780 MBps
- 2 x 30GB SSD storage

RScript Server - Linux (equivalent AWS ‘m5.2xlarge’)

- 5% utility/uptime (on demand)
- 8 x vCPUs
- 32GB RAM
- 10Gbps network burst speed
- Elastic load balancer up to 4750 MBps
- 2 x 30GB SSD storage

PostGreSQL Database Instance

- 20GB of storage
- 1 million input/output requests per month
- 1 CPU Core
- 2 vCPUs
- 16GB RAM
- Up to 3500 MBps Dedicated Bandwidth
- Up to 10GBps Network Performance
- Up to 3.1 GHz Intel® Xeon® Platinum 8000 Processor
- Intel AVX, Intel AVX2, Intel Turbo

3.3.2 Cost Estimate

It is difficult to estimate the on-premises hosting costs without knowledge of hardware purchasing providers, staff costs and the existing network infrastructure. Hardware costs have been estimated based on a Hewlett-Packard workstation provider (ZWorkstations 2020), although a mainframe would be utilized in practice. We also assume that hardware would be

replaced every three years and that there will be no hardware failures. Staff costs are based on the 75% salary percentile for a Network and Computer Systems Administrator (U.S. Department of Labor Statistics 2019).

Based on these assumptions, the minimum cost for on-premises hosting over three years is $21,900, with an upfront cost of $13,700 in the first year and an additional $4,100 per year for maintenance (Table 2). To up scale the system to obtain better performance, responsiveness and/or more storage, the total cost estimate is $43,800 (approximately $27,400 in the first year and an additional $8,200 in the second and third years). Under this scenario the hardware costs and staff time needed to setup the servers doubles ($19,200), and annual maintenance costs grow to $8,200 per year.

Table 2. Minimum cost estimate for on-premises FDAT hosting in years 1, 2, and 3, broken down into hardware and staff costs. The assumption is that hardware will be replaced every three years and all costs are in US dollars.

Expenditure Type	Description	Cost Year 1	Cost Years 2 & 3
Hardware Costs			
Web Server Hardware	HP Z800	850	0
RScript Server Hardware	HP Z820	1,750	0
Server for PostGreSQL database	HP Z800	850	0
Staff Costs			
Server Setup	15 Days for Network and Computer Systems Administrator	6,150	0
Server updates and maintenance	10 days per year for Network and Computer Systems Administrator salary	4,100	8,200
Total Costs		**13,700**	**8,200**

3.4 Summary

The cost of implementing, hosting and maintaining FDAT is not insignificant, but we believe that it provides long-term value for money. Enormous field sampling and habitat restoration efforts have occurred in the Columbia River Basin, but existing data are often housed within different organizations and/or stored in disparate formats, making data integration at regional scales challenging. Analyzing data from multiple sources can also be a technical barrier for users who would benefit from basin-scale assessments, given the advanced skills in aquatic biology and ecology, spatial science, and computational modeling and statistics needed to fit spatial statistical stream-network models, and other comparable models. The FDAT addresses many of these needs.

- It is designed to incentivize the development of consistent, centralized databases that are freely available to the public.
- It removes the technical barriers by automating the spatial data processing and network-scale modeling of fish density and abundance datasets collected by multiple organizations.
- The predictive model outputs, including uncertainty estimates, provide new information about current status and future trends mined from existing datasets, which are also freely available for download.
- Predicted abundances can also be integrated over the full network or subdomains within the network to obtain population estimates at a variety of spatial and/or temporal scales that are most relevant to conservation and investment planning.
- Analysis methods are well documented, easy to locate, and reproducible, to promote transparency and provide users with the confidence in the results.

While the primary focus of FDAT is on fish density and abundance data, the software architecture was designed to be scalable and robust as FDAT evolves. Thus, the same system architecture can be used to integrate data and model other stream variables such as temperature, with relatively minor modifications.

4 Functional Requirements

4.1 User Roles

Security and authentication form an important part of any publicly exposed web application. The web interface will provide the ability to register for a new user account, login, and reset passwords. Credentials and authentication data will be encrypted and stored in the application database, as per industry best practices.

To manage users and allow restricted access to different application functionality, there are several account roles required:

- **Administrators:** approve, create and remove users;
- **Public Users:** explore, visualize, and download input data, shapefiles, model outputs, and documentation describing spatial processing and modeling methods; and
- **Registered users:** have access to all of the functionality of public users, plus the ability to upload new observation datasets.

4.1.1 Administrative users

Administrator accounts are for use by BPA staff and will have permission to approve new account registrations, disable access for existing account registrations, view statistics for registered users, and remove erroneous data. In addition, all interactions within the FDAT application will be tracked in the system database. This will allow the Administrator to examine:

- Authentication attempts and IP addresses;
- Account registrations;
- Data upload activity, by user;
- Log files for modeling that contain error messages; and
- Data download statistics.

Note that the reporting tools and associated data will only be available to Administrative users.

4.1.2 Registered users

Registered users will be able to log into the system using a standard authentication form containing a username and self-managed password. They will then be able to upload a dataset in a standardized format, which is subsequently incorporated into the existing data used for predictive modeling. There are a combination of approaches that could be used to initiate new Registered user accounts. New users could request a Registered user account, which would require approval from the Administrator. The BPA could also generate and send Registered user credentials with temporary passwords to contractors, requiring a new password on first log in. This would allow the BPA to automatically generate user accounts associated with each project they fund so that field data may be uploaded and stored in a central repository.

4.1.3 Public users

Public users anyone interested in exploring data or FDAT model outputs who does not have permission to contribute data to the system. This might include university researchers, contractors or other scientists who are not currently working with the BPA, as well as the general public. Public users will not be required to register to access and download observation data, shapefiles, or model outputs. However, it would be possible to implement rate-limited application programming interface (API) services in the future if website usage scales beyond initial expectations. API rate limiting would be implemented by requiring public users to register before accessing data APIs. A unique API token would be used to digitally sign their data requests, which enables usage tracking and where necessary, allows limits to be placed on access for excessive users. This would in some ways hinder public data access and so it is not recommended at this stage.

4.2 Visualization, Exploration & Download

All FDAT users will interact through a web-based application, which provides access to a spatial data exploration interface, custom generated figures summarizing observations and model outputs, and the ability to download observations and model outputs (Figure 1a).

4.2.1 Mapping

The web-based mapping interface is the main tool for interactive data exploration, which allows users to visualize data from multiple organizations, as well as model outputs. Standard mapping mouse-based interactions, similar to those used in common interfaces such as Google Maps and Apple Maps, are used to navigate spatial layers within the spatial extent of the stream network. This standardized interface ensures that the vast majority of users will be able to intuitively use the system.

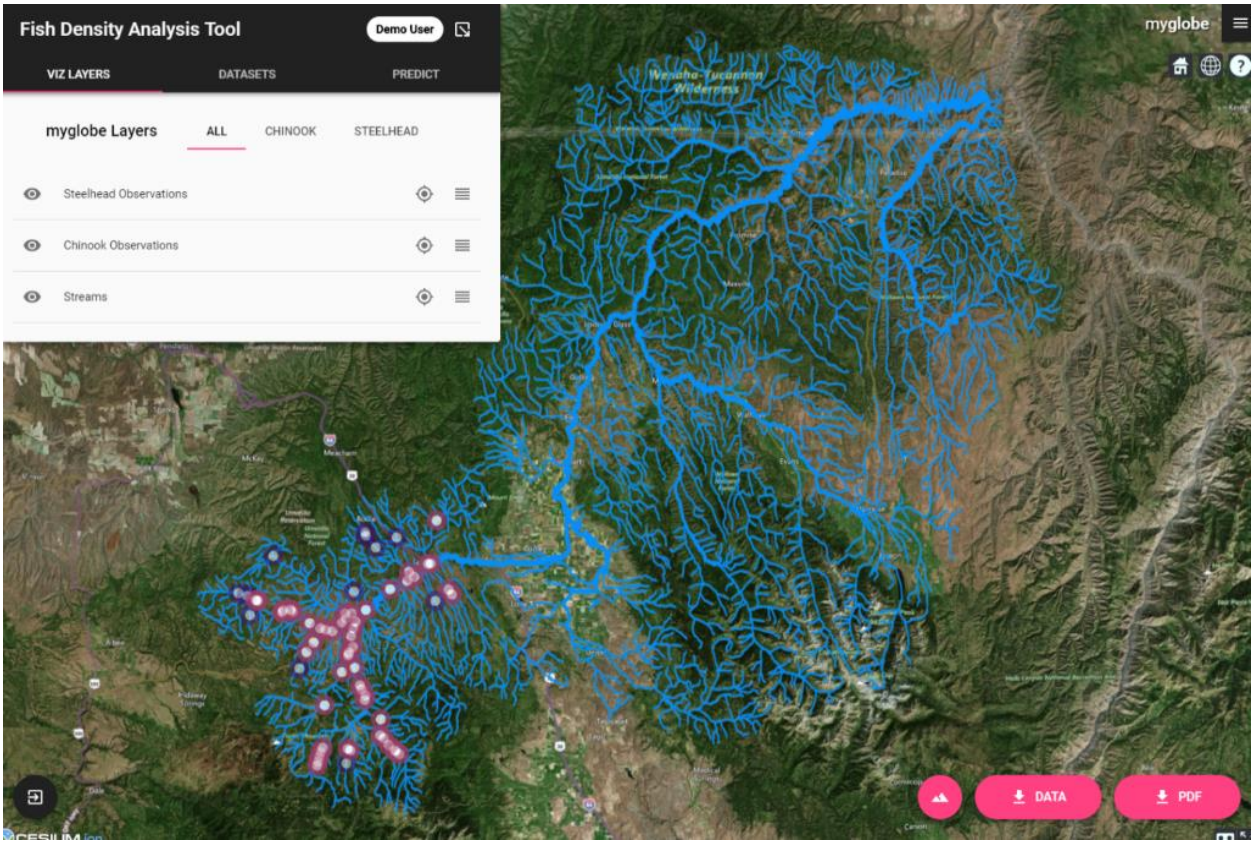

Figure 3. FDAT web-based mapping interface allows users to explore field observations from multiple organizations.

4.2.2 Figures/Graphs

FDAT users will also be able to interactively generate and explore a number of visual summaries through the mapping interface, including figures showing data collected at a site over time or summaries of predictions and associated estimates of uncertainty over predetermined spatial scales (i.e., population or catchment). The user can turn a variety of spatial base layers on and off using a menu/legend interface (Figure 4a); while also being able to further interrogate data associated with spatial features such as observation and prediction points, populations, and catchments by clicking on or selecting the interactive components of the map.

4.2.2.1 Explore Observations

A user can explore the existing observational data by interacting with the map interface and clicking on individual observation sites. Each unique observation location will be shown as a point feature on the map, overlaid on the stream network and underlying mapping layer (Figure 4a). When a site is selected, the user will be presented with a figure showing the observed species density per year, broken down into juvenile and adult age classes (Figures 3a and b). If more than one species is present at a site, then figures may be stacked or contain different tabs that allow the user to switch between species and explore data.

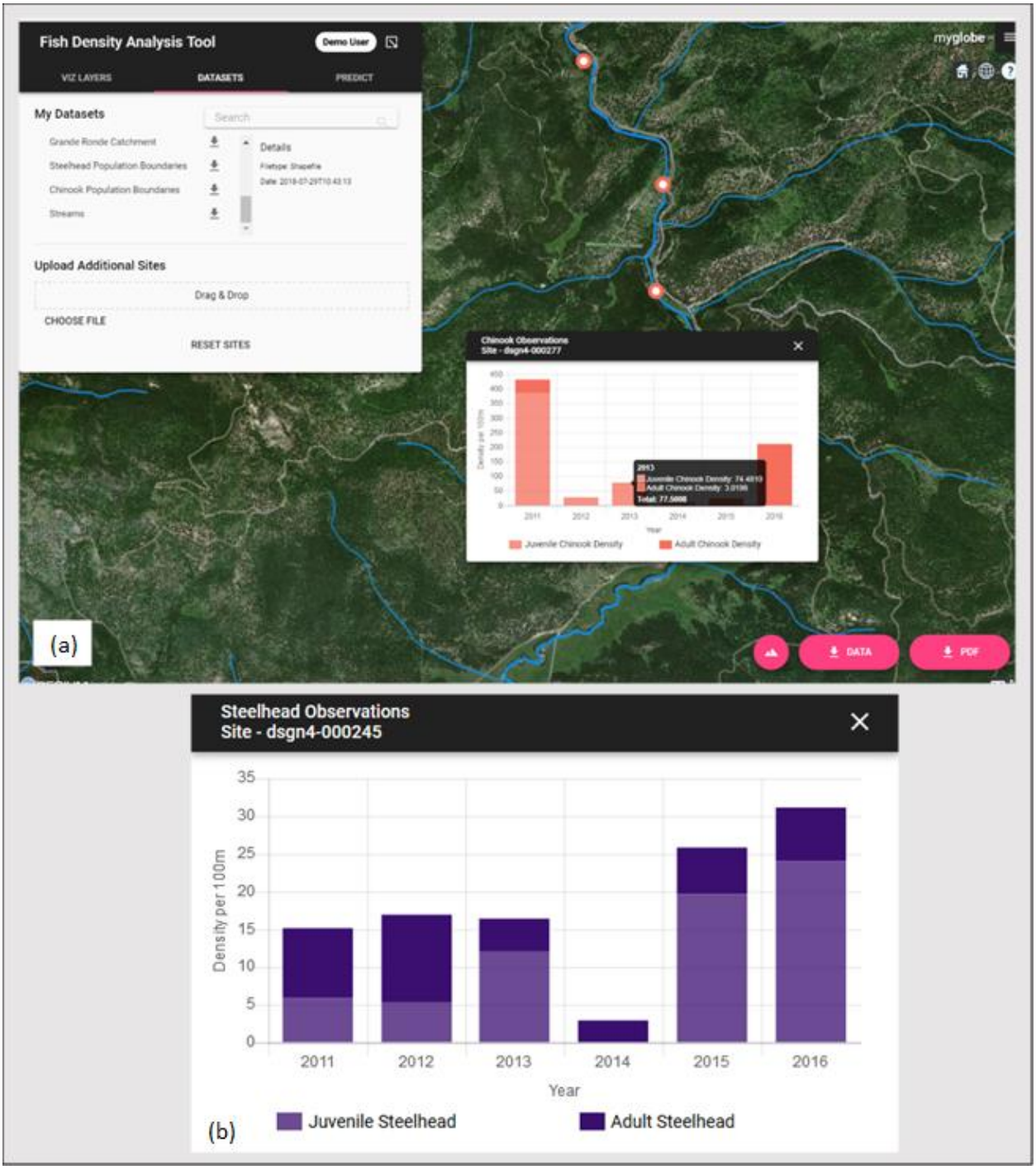

Figure 4. Visual summaries of observation data can be interactively launched through the FDAT mapping interface.

4.2.2.2 Explore Predictions

To explore fish density predictions for given a site, population, and/or catchment by species, the user selects a layer in the menu/legend interface and clicks on a spatial feature in the map to visually display information. Figures will appear summarizing predictions for a given species, per year, including the estimates of uncertainty for the predictions (Figure 5).

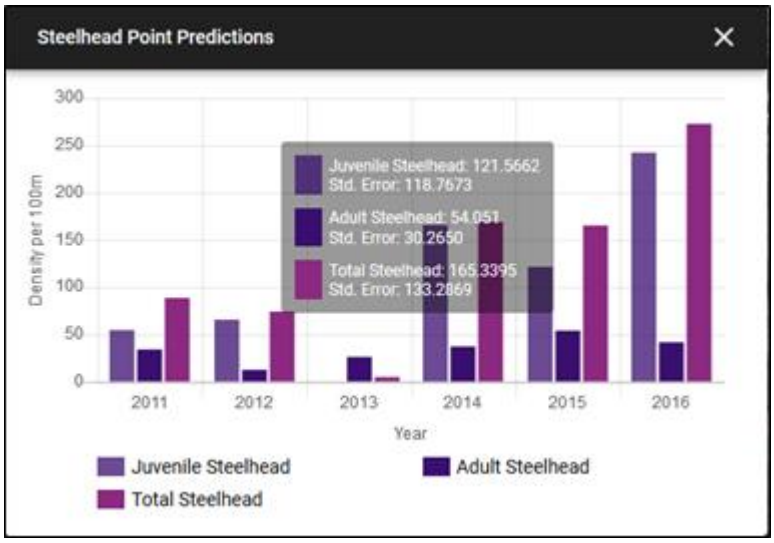

Figure 5. Visual summaries of point predictions show the predicted fish density, by species, over time as a total and broken down by age class. Associated estimates of uncertainty are also provided for each of the predictions.

4.2.2.3 Dynamic Summaries

An optional feature that could be included in FDAT is to allow users to interactively select point predictions through the mapping interface and then request mean predictions, with estimates of prediction variance over the user-defined area. These are referred to as ‘block kriging’ predictions and would be displayed in a similar style to populations and catchments. In addition, the predictions could be made on-the-fly and provided for download as a .csv file through the mapping interface.

4.2.3 Download

One advantage of the FDAT is that it provides a central repository for field observations collected by multiple organizations and model outputs. This ensures that data are freely accessible to the public, that analysis methods are well documented, and that the modeling outputs are reproducible. FDAT users will be able to download the: input data used to fit the predictive models; shapefiles representing streams, observation locations, prediction locations, and catchment and population boundaries; and .csv files containing predictions with estimates of uncertainty at point, population, and catchment scales. In addition, a pdf document describing the methods used to process the data, fit the statistical models, and generate predictions will be available for download through the website.

4.3 Data Upload

Registered users are granted permission to upload new observations to FDAT in the BPA data exchange standard (DES) format (Appendix 1) using a traditional file upload form. The data are validated (Figure 6) before additional processing, modeling and visualization takes place (Figure 1b).

4.3.1 Data Validation

The DES is formatted Excel file, which enables data from multiple organizations to be automatically combined within FDAT. Once the user has uploaded their data file, it is analyzed on the server to ensure the data entered is complete, conforms to the standard correctly, and is free of obvious errors (Figure 6). If the file fails the validation process, it will be rejected by the system. The user will then receive an error message explaining why the file was rejected and will be asked to make corrections and retry. If the file passes the validation process, the user is taken to the Snap Points user interface.

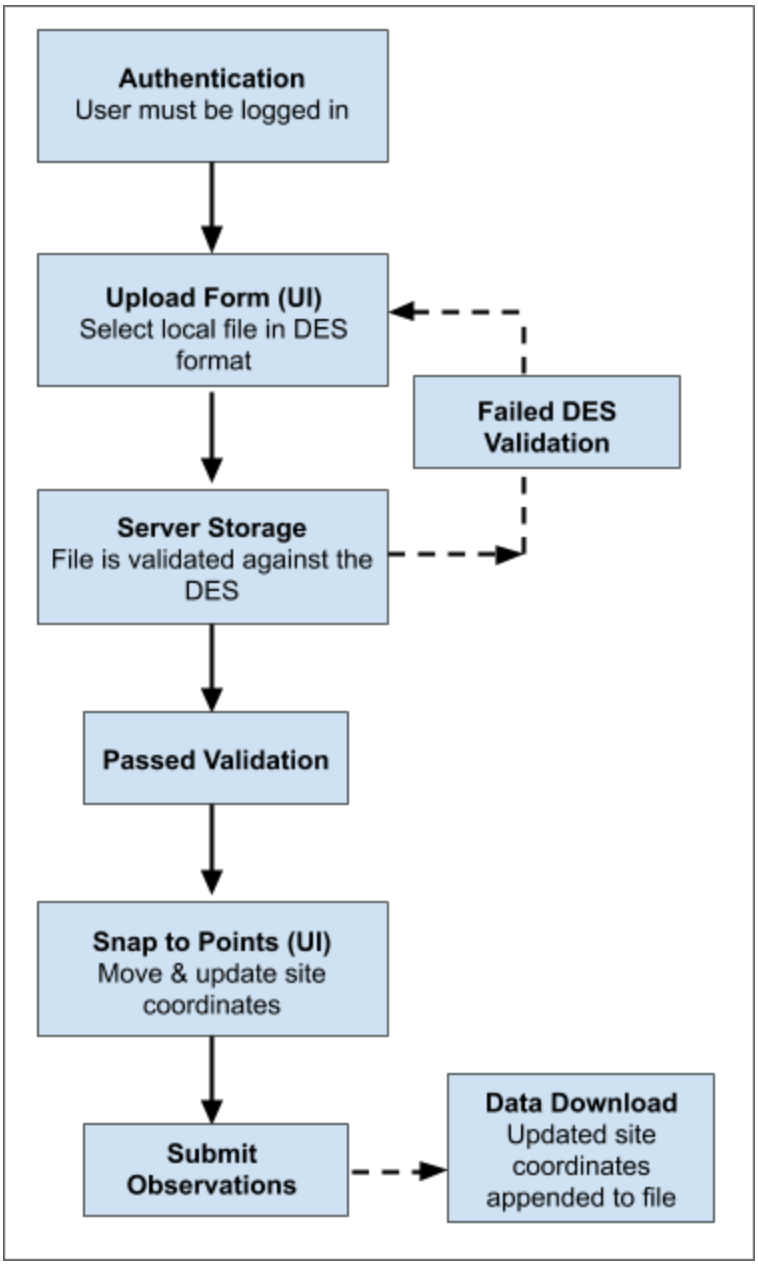

Figure 6. The Data Upload workflow within FDAT.

4.3.2 Snap Points (UI)

When a registered user uploads their observations in DES format, the locations of the observation points may not align exactly to a location *on* the stream network (Figure 7) due to errors in the GPS locations, the resolution of the digital streams dataset, or changes in the real-world location of a stream reach (i.e., meandering streams). Nevertheless, this is a requirement for the spatial stream-network models used within FDAT and so locations must be moved to the correct line segment. The Snap Points interface allows the user to interactively undertake quality assurance and quality control (QA/QC) of the data editing process to ensure data are represented correctly in FDAT.

The Snap Points interface allows Registered users to visually examine each site location and manually drag observation points nearer to the correct stream location if necessary (Figure 7). Once the site locations have been adjusted, the user clicks the ‘Snap Points’ button which initiates the process of moving observation points to the nearest location on a line segment. The new site locations are then displayed on the map so that the user can clearly see where the new sites reside. This iterative process continues until the user is satisfied the locations are correct and clicks the Save Changes button.

After the QA/QC process is complete and the changes have been saved, the user submits the data to the FDAT database. They will also be given the option to download a modified version of the data file they originally uploaded in the DES format, with two new columns appended representing the coordinates for the updated field site locations.

4.4 Spatial Data Processing

Numerous spatial data-processing steps take place behind the scenes in FDAT to ensure the field data contributed by users is in a suitable format for spatial stream-network modeling (Figure 1c). An overview of these steps is provided in the following sections.

4.4.1 Observation Queueing

When new observations are uploaded by users throughout a given day, the new records are queued in a separate part of the database before the spatial processing takes place. A scheduled task is triggered at set time points (e.g., daily, seasonally, annually, etc.), which starts the spatial data-processing process (Figure 1c). Note that this schedule can be adjusted accordingly to suit the BPA requirements once the system is in use. It also occurs independently of the predictive modeling tasks, which means that the spatial data processing and predictive modeling can be scheduled to run at different time intervals. New observations, potentially contributed by multiple users, are converted to shapefile format and imported into R Statistical Software (R Core Team 2020) using the osSTARS package developed in this project. As each

new set of observations is processed, the database is updated to remove them from the queue, with the success or failure of the process logged in the database for examination by the Administrator if required. If there are no records in the queue, the process is terminated before execution.

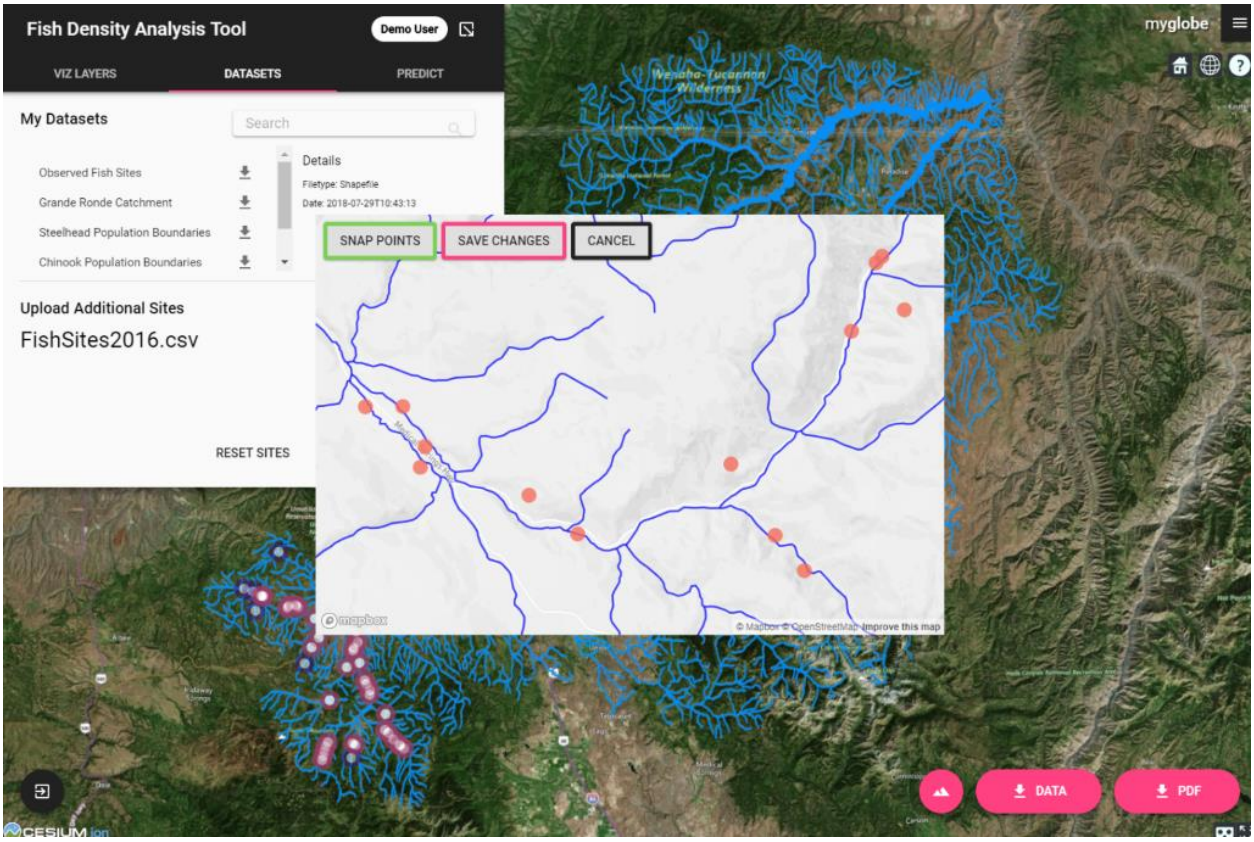

Figure 7. The Snap Points user interface, which allows users to view the current location of field observations in relation to the stream segment and manually adjust them prior to snapping.

4.4.2 osSTARS

The osSTARS package replicates the core functionality of the STARS custom toolkit (Peterson and Ver Hoef 2014) needed to support the FDAT. Five new functions were created that have a 1:1 relationship with those found in STARS (Table 3; Appendix 2). These functions convert the streams and observed sites to a Landscape Network (LSN) structure, which is the primary data format in STARS (Peterson and Ver Hoef 2014). There are also functions to calculate the upstream distance (upDist) from the stream network outlet to the upstream node of each line

segment and to each individual observed site location. Custom R scripts are used to automatically generate and record covariate values (i.e., predictor variables) in the new observation sites attribute table, based on attributes in the edges shapefile (e.g., stream slope), raster layer values (e.g., land use), or data from nearby weather stations (e.g., air temperature). Additive function values are also needed to fit spatial stream-network models using the tail-up covariance function (Ver Hoef and Peterson 2010) and these are generated using the additive.function function in the SSN package (Ver Hoef et al. 2014) for R Statistical Software (R Core Team 2020); thus, a corresponding function was not included in osSTARS. Once the osSTARS Pre-processing and Calculate functions (Table 3) have successfully completed, the new observation sites are merged with the existing set of observation sites using custom R code to create a single, updated shapefile of observation sites. At this point, all of the data are exported to a .ssn object, which is identical to .ssn objects created using the STARS tools. This .ssn object represents the input data for the Modeling and Predictions component of FDAT (Figure 1d). Please see Appendix 4 for a more detailed description of the individual osSTARS functions.

Table 3. The relationship between STARS tools and equivalent functions within the osSTARS package. osSTARS functions are listed in the order that they will be called within FDAT.

STARS Tool Name	**osSTARS Function Name**
Pre-processing Polyline to Landscape Network Snap Points to Landscape Network	 polyline_to_network snap_pts_network
Calculate Upstream Distance – Edges Upstream Distance – Sites	 getUpstreamDist_Edges getUpstreamDist_Sites
Export Create .ssn object	 CreateSSN_Obj

4.5 Modeling & Predictions

Significant investments in long-term field sampling throughout the Columbia River Basin create both an opportunity and a challenge for spatial stream-network modeling at regional scales. We estimate that the number of fish density observations currently available will be between 10,000 and 15,000, with up to 150,000 prediction locations, depending on species. Traditional spatial statistical models are not feasible for these data because solutions are not found in realistic

timeframes or the memory requirements needed to invert the autocorrelation matrices and/or store them cannot be met.

The SSNbd package (https://github.com/jayverhoef/SSNbd) for R statistical software was developed in this project to address the computational challenges of fitting spatial statistical stream-network models to large datasets and making predictions across the Columbia River Basin. First, new R functions were created to generate pairwise stream distances between observations, observations and predictions, and between predictions using the filematrix package (Shabalin 2018). These functions allow pairwise hydrologic distances to be calculated in computationally manageable steps and then stored as files rather than attempting to store the massive matrices in computer memory. Another advantage is that the matrices can be accessed using indexing, as is usual with R matrices stored in memory. The functions used to fit spatial statistical stream-network models and make block kriging predictions in the SSN package have also been modified to make use of data partitioning methods in the SSNbd package, which improves the computational efficiency of statistical estimation and prediction. This approach allows the dataset to be subsampled to produce smaller, spatially structured subsets of observations, which are used to quickly estimate parameters. These parameter estimates are then combined using a weighting scheme informed by cross-validation predictive ability to produce global parameter estimates used in the modeling and prediction (Barbian and Assuncão 2017). In addition to the methodological modifications, all of the functions have been written to take advantage of parallel processing, which further reduces processing time by enabling the tasks to run on multiple microprocessors simultaneously.

4.5.1 Methods

The spatial, topological, and attribute data contained in the .ssn object are re-imported into R, where updated distance matrices are calculated. The SSNbd package (Ver Hoef et al. In Prep) is then used to fit spatial statistical stream-network models and produce semi-continuous maps of fish density using the universal kriging equations (Cressie 1993). Methods for spatial stream-network modeling and block kriging were originally described in Ver Hoef et al. (2006) and Ver Hoef (2008), with recent modifications specific to estimating fish population sizes in stream networks described in Isaak et al. (2017). A detailed description of the statistical methods is also provided in Monitoring Method ID 6621.

FDAT produces abundance estimates, with estimates of uncertainty, by species and age class at multiple scales. First, predictions of average fish density per 100m are generated at a dense set of prediction locations spaced evenly across the stream network (e.g., 250m to 1km; Figures 7 and 8). Mean predicted abundance, along with associated uncertainty estimates, can then be generated over larger extents, such as population boundaries (Figure 10), by integrating values at prediction points found within the area (Ver Hoef 2008; Isaak et al. 2017). This process can

be repeated at other management-relevant scales (e.g., catchment or HUC) and used to generate abundance estimates for multiple species across the Columbia River Basin.

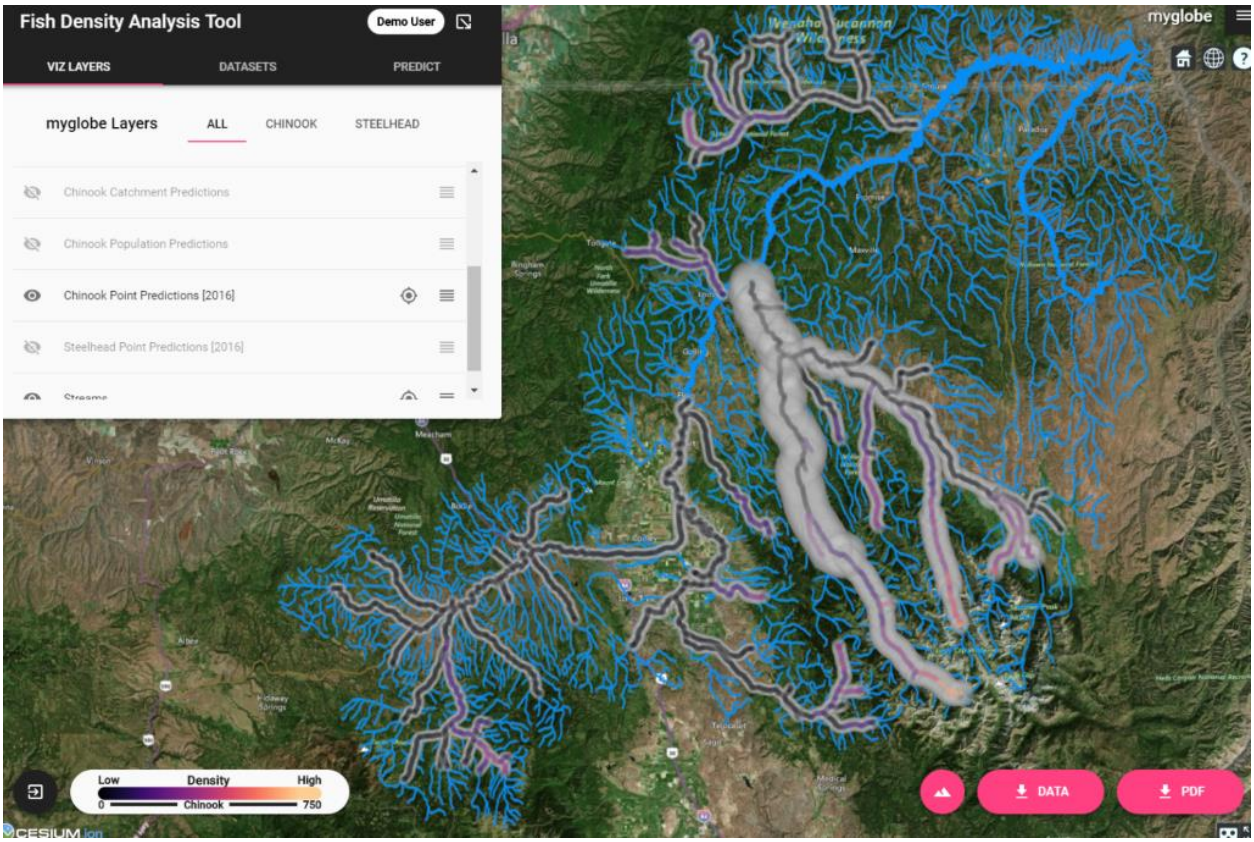

Figure 8. Point-scale fish density/100m with associated estimates of uncertainty can be visually explored through the FDAT web-based interface (modified from Peterson et al. 2018).

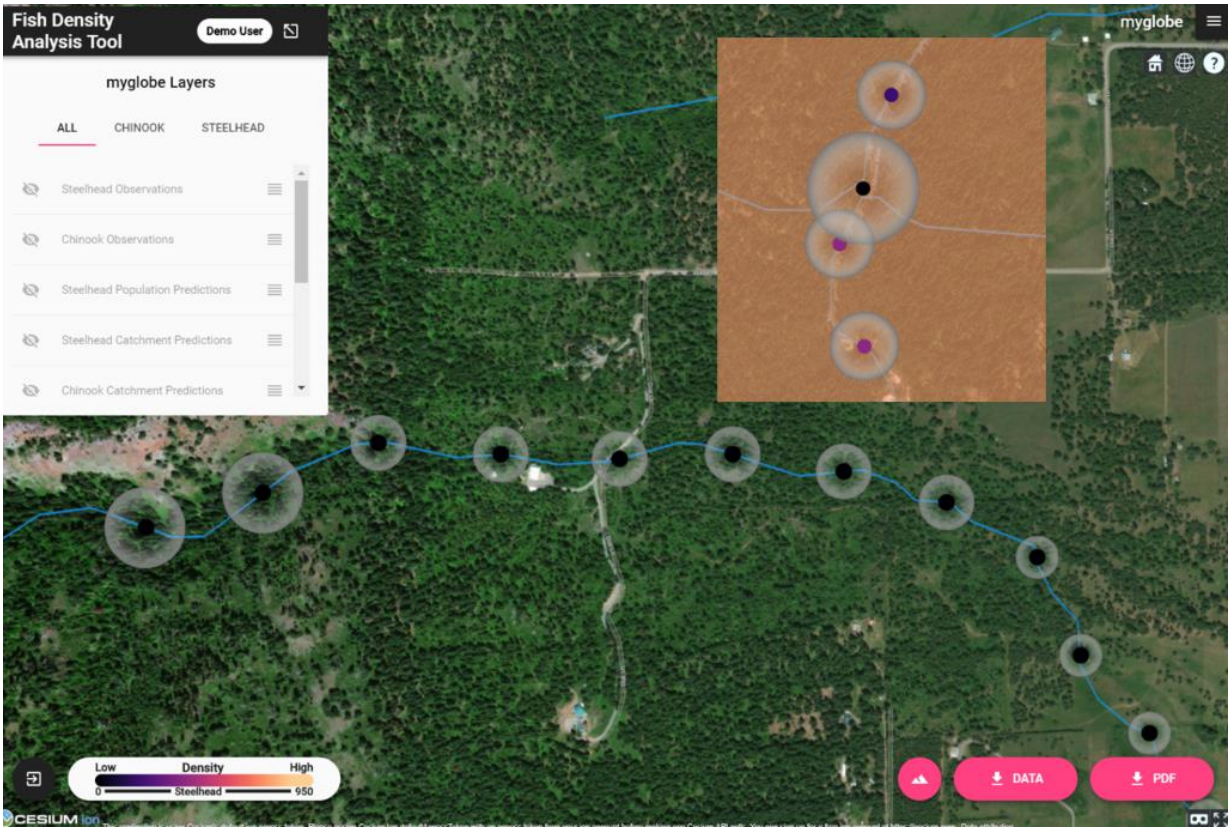

Figure 9. Prediction standard errors appear as translucent circles as the user zooms in. Circles are proportional to the relative magnitude of the standard error. The orange inset is simply an example showing differences in prediction standard errors from another part of the network.

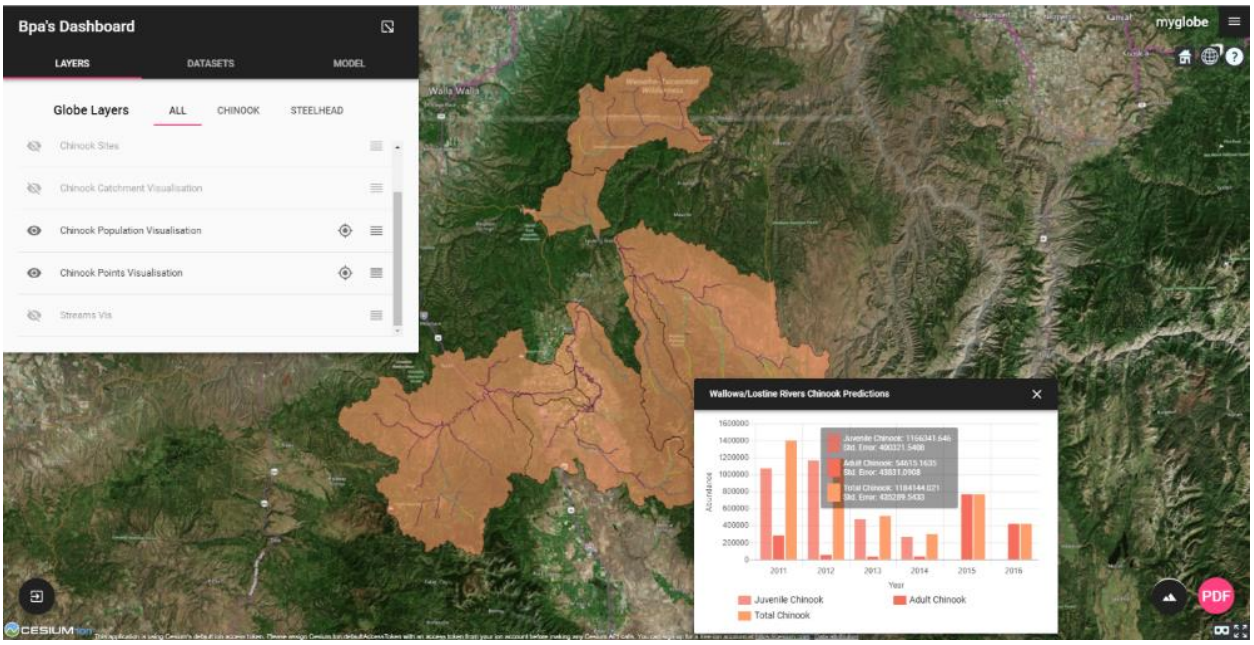

Figure 10. Predictions of Chinook salmon abundance for the Lostine Rivers population (modified from Peterson et al. 2018).

5 Software Architecture

The recommended FDAT system architecture is best described as a traditional web-server environment with supporting database, file store and a customized RScript Server (Figure 11). In the following sections we provide a detailed description of the architecture requirements, including the servers, file storage and web assets, database instances, application environments, and browser support. Only the necessary components of FDAT will be exposed to the public internet (Figure 11), which ensures that the security of the database and underlying RScript server are never compromised.

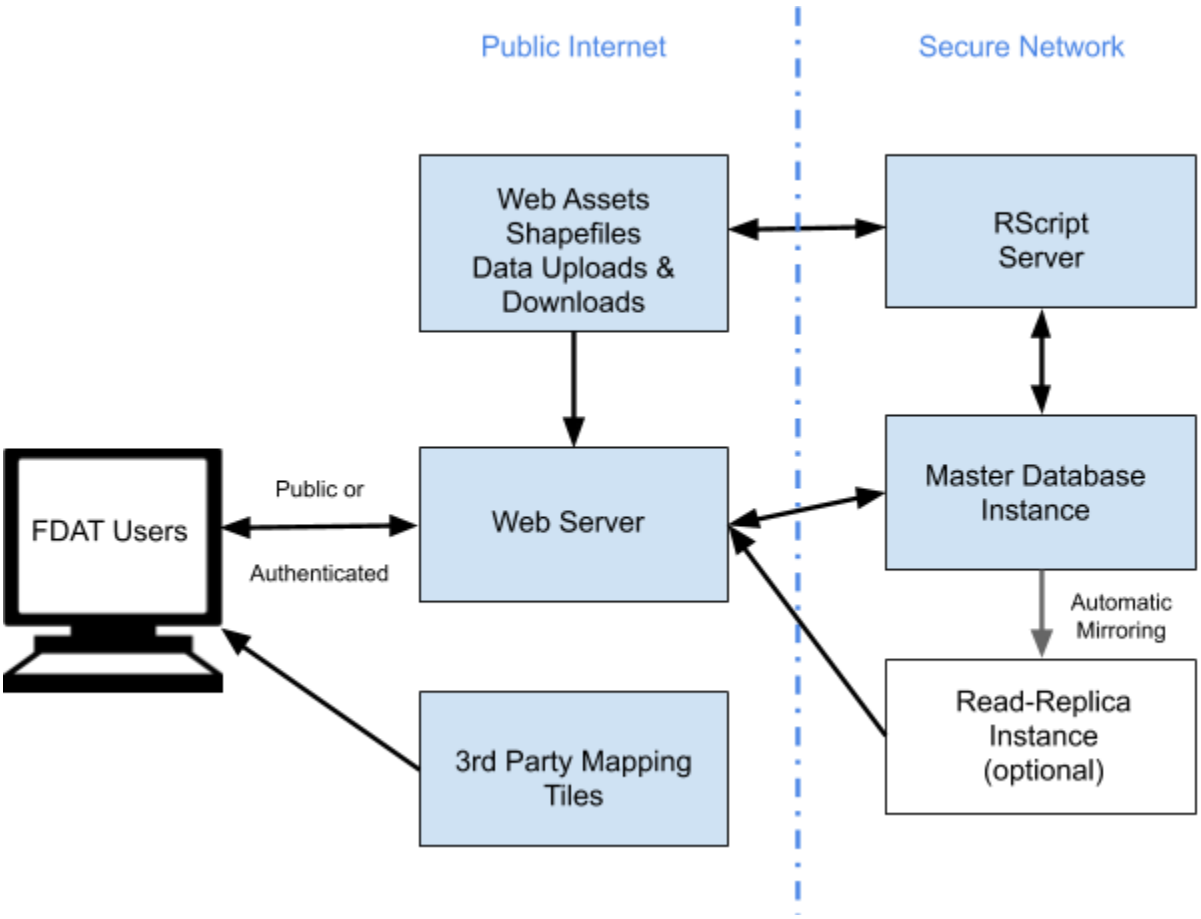

Figure 11. An overview of the system architecture with network topology.

We also considered how data will move through the system in the infrastructure design (Figure 12). It should be noted that the same Web server provides a source of entry for the observation data and the delivery mechanism for model outputs, while the RScript server undertakes spatial data processing and modeling.

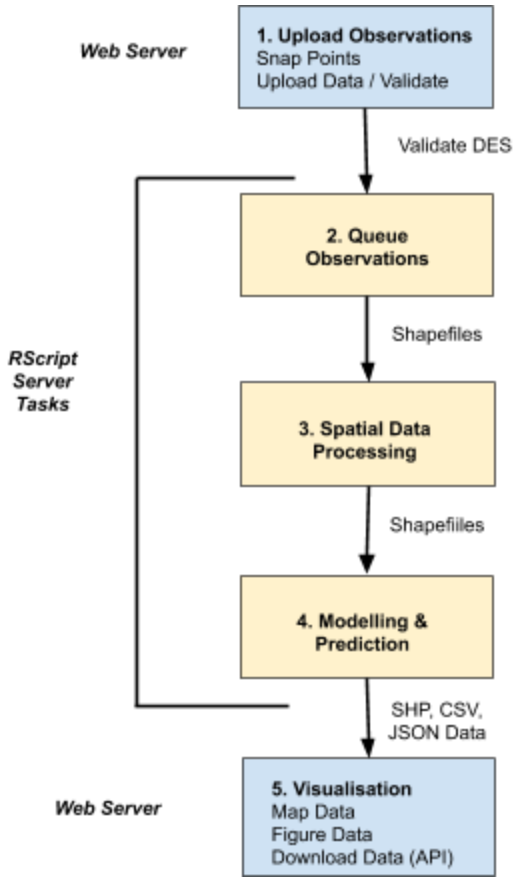

Figure 12. Data flow between the Web and RScript servers.

5.1 Servers

Separate servers and database instances will be used for FDAT to ensure that the computing resources are optimally allocated. This ensures hosting costs are utilized correctly to suit the specific functional requirements of the application, while compartmentalizing the components in a logical manner for ease of design build and maintenance. For example, we expect public users to explore and access data regularly throughout the year, which requires relatively little computing power. In contrast, the RScript server will use considerably more CPU processing resources and memory than the Web server instance when data are processed and modeled, but this will only be needed at set times. Thus, the ability to separate the Web and RScript servers enables scaling and resource allocation as required, thereby, improving processing speeds and ensuring optimal user experience.

5.1.1 Web Server

In general, the initial FDAT Web server set-up requires:

- Internet accessible server, with ports and firewalls implemented as needed;
- SSL Certificate with suitable domain name;
- Read access to the public file storage folders;
- Write access to the File Uploads folder; and
- Authenticated access to the database, only exposed through an API publicly.

We also recommend using NodeJS (OpenJS Foundation 2020) in FDAT because it is:

- A widely used and well-developed open-source web server environment;
- Custom designed on the Google V8 Engine;
- Designed for data intensive, response applications; and
- Can be scaled to multiple computing resources.

Additional benefits of NodeJS include:

- Access to Node Package Manager, which is a library of open source repositories;
- Operating system independent and can be deployed on Windows or Linux;
- File streaming module support for the delivery of large files using streaming;
- Supports multithreading, utilizing server resources;
- Shared language and syntax between server and client, allowing code reuse;
- Supports non-blocking event-driven operations and is well suited to web applications; and
- Designed for real-time data from server to client.

It should also be noted that in a cloud environment, the entire NodeJS web server can be redeployed by the developer with single line commands, without assistance. This removes the need for a system administrator to perform deployments and speeds up iterations of application testing and development.

5.1.2 RScript Server

The RScript server will be a standalone virtual-server instance, which will run an up-to-date instance of an operating system, regardless of whether it is a Windows or a Linux variant; noting that R statistical software (R Core Team 2020) can run in either environment. The server will not be directly accessible or available on the public internet (Figure 11). Instead it will be used to create file assets (e.g., updated observed sites shapefile or historical backups) and modeling outputs via database queries, file inputs and RScript tasks, which are delivered through to the FDAT Web server file store and API for internet accessibility (Figure 12).

The RScript server will have a series of scheduled tasks, which perform the data queuing, spatial data processing, and modeling as detailed in the following sections. Depending on the operating environment, these will either be Windows scheduled tasks or Linux cron job calls at set time intervals. These scripts will launch batch scripts and R scripts to perform the spatial data processing and modeling, file creation, file copying and database interactions. In terms of network topology (Figure 11), the RScript server will need to make authenticated database queries and have the ability to read and write from the various asset folders, for the generation of shapefiles and files for data downloads (see Section 5.2 for additional details).

The hardware allocation for the RScript server will be informed by performance testing during the development phase of FDAT. Both performance and cost should be considered to ensure that the system can perform the modeling in an acceptable time frame at reasonable cost, with future processing needs in mind. It is also important to note that, in an on-demand cloud computing environment, the RServer instance would only be operational and incurring an expense when it is in use. This removes the need for a server to sit idle when modeling is not performed, greatly reducing hardware allocation costs.

5.1.2.1 Software Requirements

R statistical software (R Core Team 2020) ≥ version 3.4.0 will be used to undertake the RScript server tasks and must be installed on the server. Several R packages are also needed to support the underlying RScript tasks:

- SSN
- osSTARS
- SSNbd
- RSQLite (≥ 1.1-2)
- sp
- igraph (≥ 1.0.0)
- maptools
- lattice
- methods
- Matrix
- rgdal (≥ 1.2-5)
- rgeos (≥ 0.3-22)
- filematrix
- foreach
- doParallel
- iterators
- itertools
- parallel

- nabor
- sf

The RScript server will require a system account with the ability to execute Windows scheduled tasks or Linux cron jobs, which will call R scripts stored on the RScript Server. This account should have appropriate read and write permissions to the file storage locations detailed in Section 5.2.

5.1.2.2 RScript Scheduled Tasks

Three tasks are executed on the RScript server to process new observation data, fit models, and generate model outputs. The three tasks include Queue Observations, Spatial Data Processing and Modeling & Predictions (Figure 13), which are executed at scheduled intervals (e.g., daily, weekly, monthly, seasonally, or annually), with a minimum offset between each task to allow the previous process to finish before the next starts. The separation of the tasks within the RScript server allows for independent scheduling and computing resources to be applied to each process, optimizing performance and resource allocation. Throughout the execution of each RScript task, both errors and success of operations are tracked by allowing the RScript server to log events to relevant database tables. This provides Administrators with easily accessible information about the cause of the errors in the event there are issues in the execution of tasks.

5.1.2.2.1 Queue Observations

New observations will be queued in separate database tables to indicate they are pending processing (Figure 14). A scripted query will be performed on the Observations Database tables, to check for records that have not been processed before beginning the process. The process requires read/write access to the Working Directory and Historical New Observations folders.

In the event that this dataset contains new observations, the following tasks are triggered (Figure 14).

1. A New_Obs shapefile is created, which contains all the new observations uploaded by users which have not yet been processed.
2. The resulting shapefile is placed into the Working Directory, which holds all the unchanged file inputs needed for the Spatial Data Processing task.
3. The previous version of the sites shapefile is copied into a backup folder and appropriately renamed with a date stamp to allow for potential rollback, or the examination of changes to the files over time if required.

4. For each completed observation set, a query will be called on the database, which takes as input the Row ID for a given observation, and updates the given row Boolean flag to be marked as Not Pending, meaning it has been processed. This completes the processing for a set of observations.

In the event that there are no pending records, the task exits without triggering these actions.

The Queue Observations process enables multiple users to upload data simultaneously and prevents excessive CPU usage incurred by running the Spatial Data Processing and Modeling and Predictions tasks after each new data upload. Testing during the development phase will determine the minimum offset between tasks, but we anticipate a minimum of one hour to process batches of new observations before modeling tasks commence.

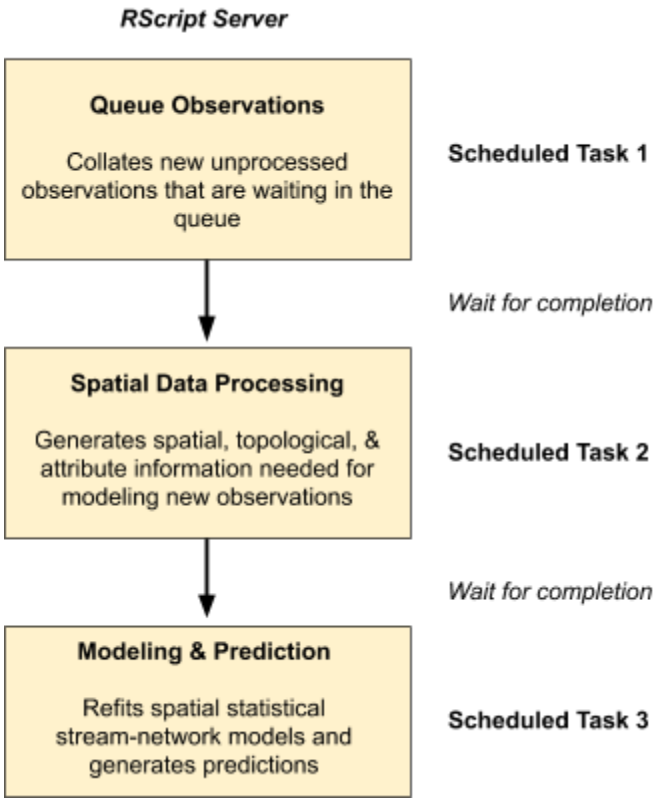

Figure 13. Workflow of RScript server scheduled tasks. Each task is dependent on the outputs of the previous task.

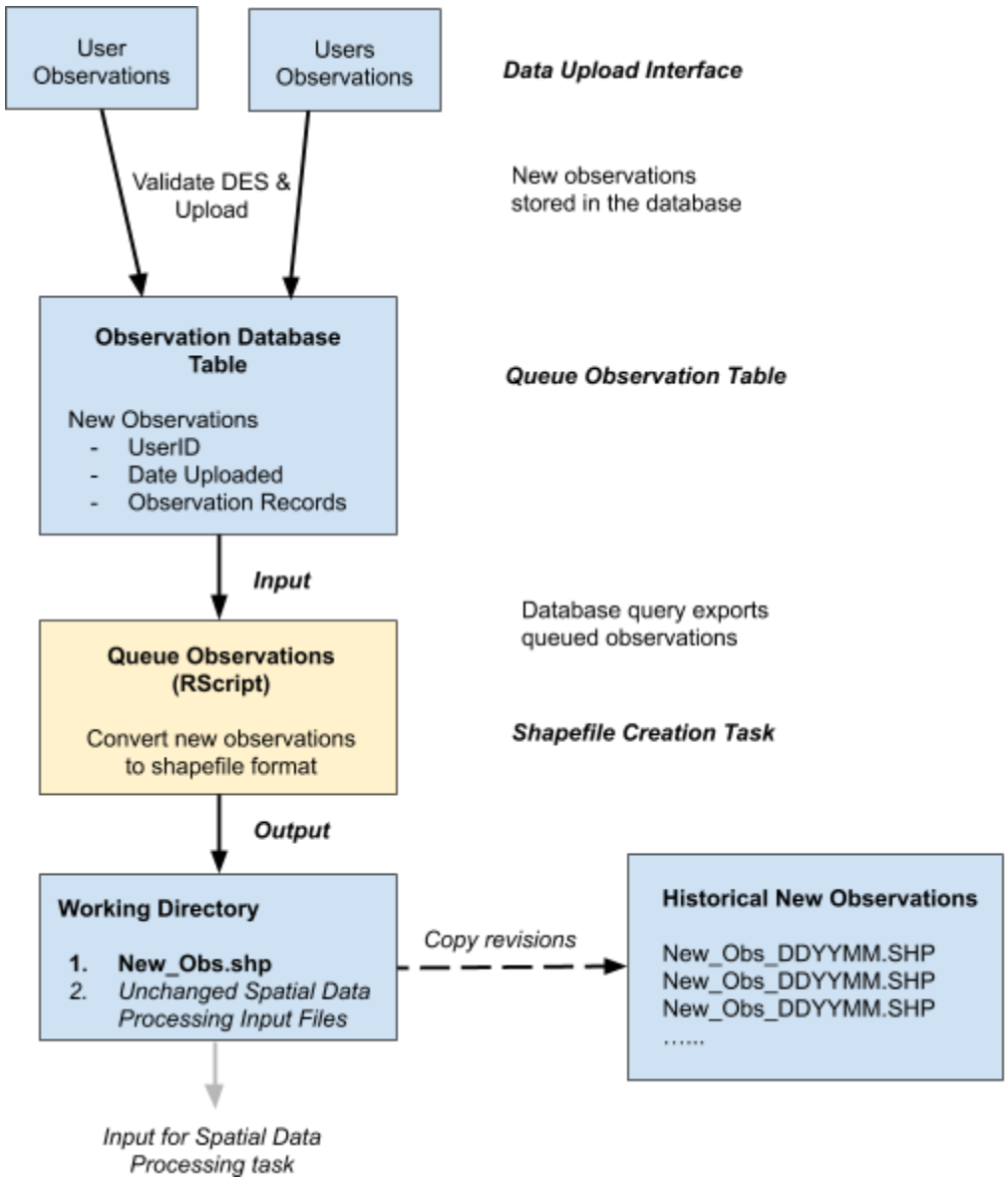

Figure 14. Queue Observations workflow.

5.1.2.2.2 Spatial Data Processing

The second scheduled RScript task runs a series of R scripts that generate the spatial, topological, and attribute information needed to fit spatial statistical stream-network models to the new observations in the Modeling & Predictions task (Figure 15; Section 4.5 and Appendix

4). The process requires read/write access to the Working Directory, which serves as both the input for this task, as well as the destination output folder.

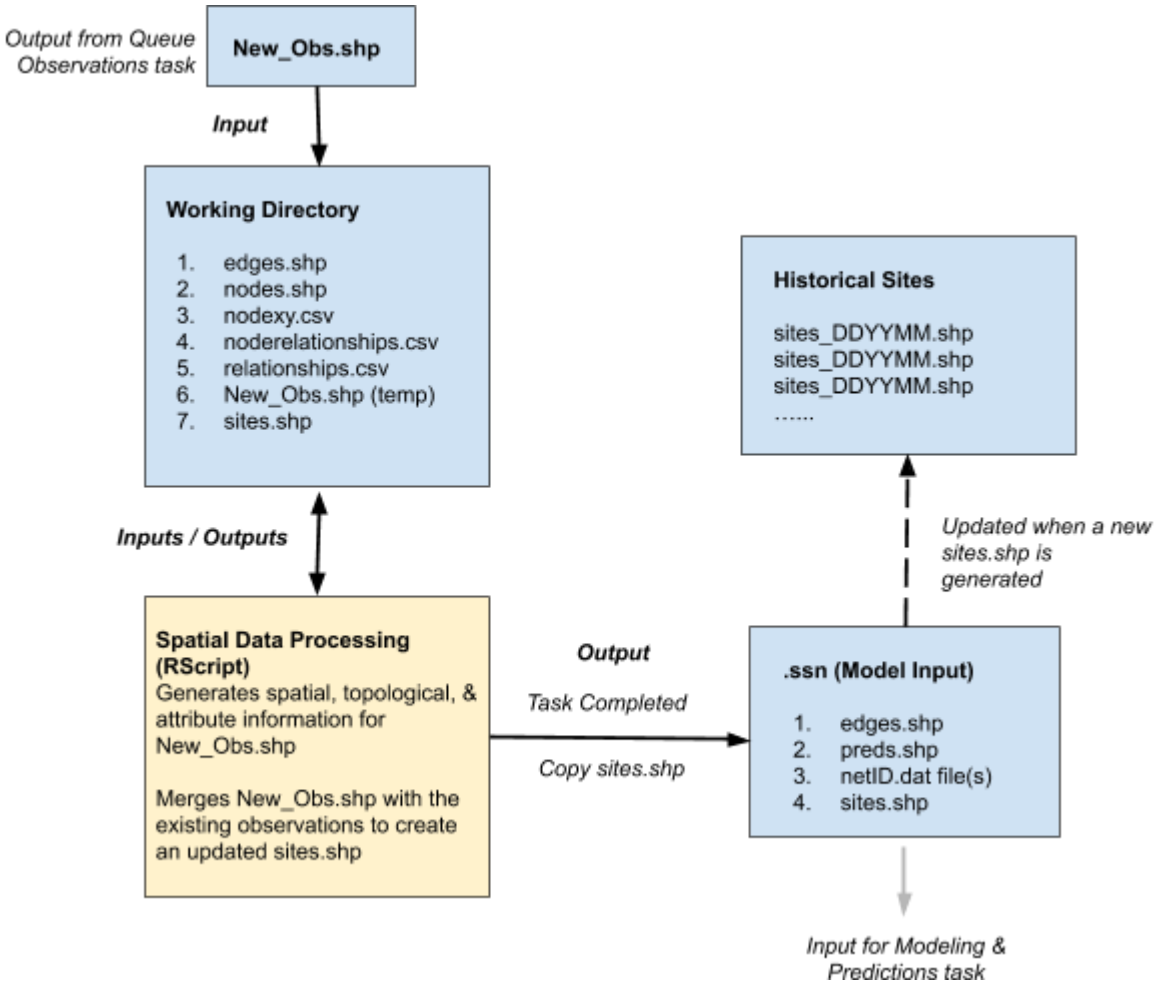

Figure 15. Spatial Data Processing task workflow.

Once the spatial information has been generated, the new observations are merged with the existing observations to create an updated sites shapefile and the temporary New_Obs shapefile is deleted from the working directory. The updated sites shapefile is then copied to the .ssn (Model Input) folder (Figure 15), and the previous copy of the file in that folder moved to a backup folder called Historical Sites. The file is renamed according to the current date and time.

The data in the updated .ssn (Model Input) folder now forms the input for the Modeling & Predictions task, and contains all of the observation data uploaded to FDAT to date.

5.1.2.2.3 Modeling & Predictions

The Modeling & Predictions RScript task fits spatial statistical stream-network models to the data found in the .ssn (Model Input) folder, generates predictions, with associated estimates of uncertainty at multiple scales, and formats these model outputs for use as visualization layers, database records, and publicly available data downloads (Figure 16). Although this task initiates on a schedule in the same way the previous RScript tasks do, the first part of the task is to check the date the sites shapefile was last modified to see if it was updated since the model was last executed. In the event that no new observations have been provided, the task will exit without running. If a recently modified sites shapefile is detected, the modeling process begins (see Section 4.5 for details).

The outputs for this process are fed back into the web application for visualization, downloads and to support dynamic web queries with additional predictions stored in the database (Figure 16).

1. Visualization Layers: contains the updated sites shapefile, so that users can view the observation data via the web interface. The shapefiles are converted to a format the mapping library supports (e.g., GeoJSON).
2. Database Records: New observations and predictions are stored in the database instance, which has been optimized for both spatial and temporal queries of massive datasets. This improves the efficiency of the dynamic web queries needed to support the on-demand summary figures generated for users in the web interface (please see Section 4.2.2).
3. Data Downloads: The predictions are exported as .csv files named based on the current date and time, which contain the predictions and associated estimates of uncertainty, at the point, population, and catchment scale, by species and age class. These data are then stored, along with the Model Input data and a pdf document describing the modeling methods in the Data Downloads directory.

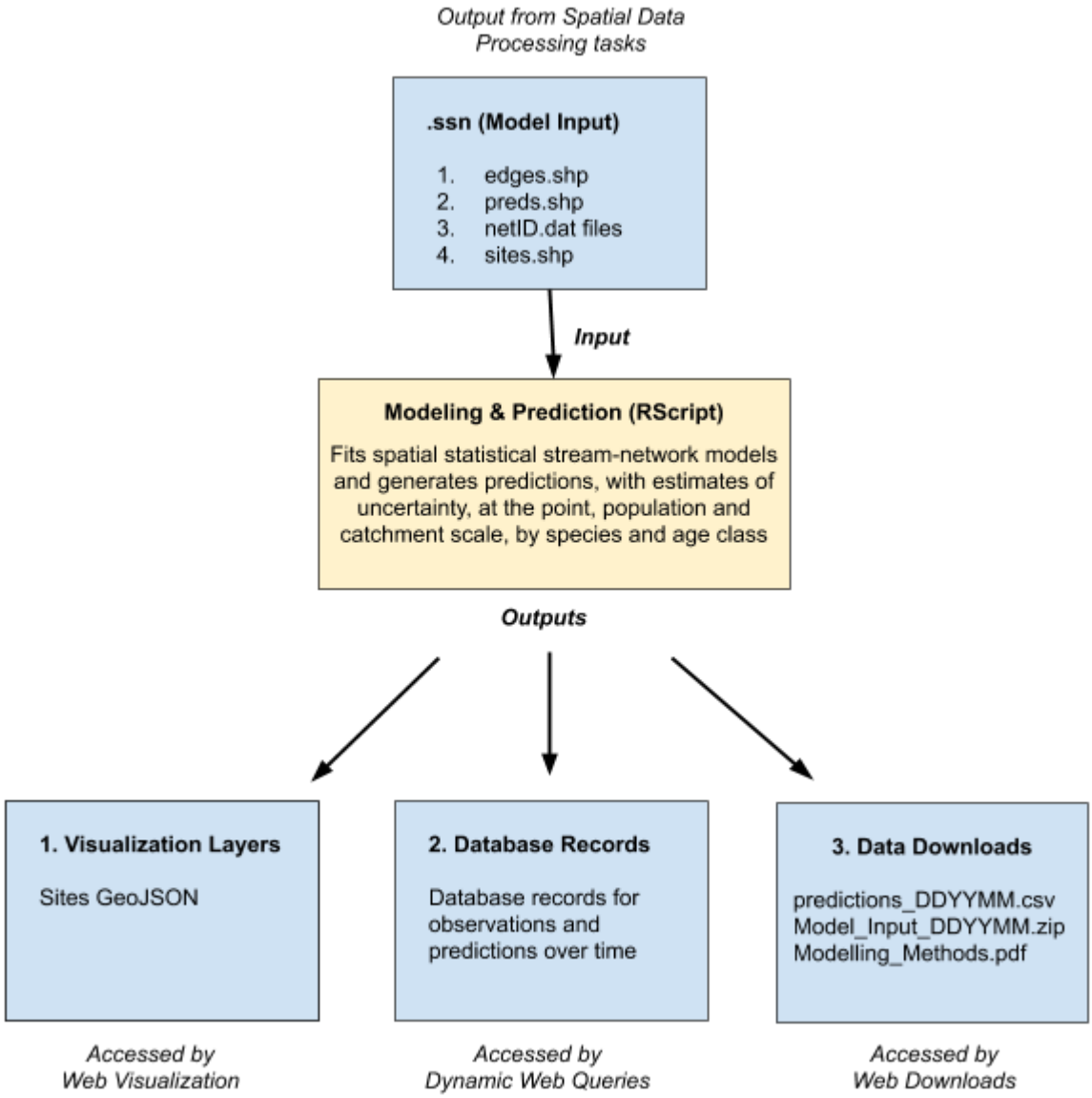

Figure 16. Overview of FDAT Modeling & Prediction workflow.

5.2 File Storage & Web Assets

The Web and RScript servers will depend on both static and generated file assets (Figures 13, 14, and 15). In this section we explore the types of files that are produced and provide recommendations for storage architecture.

Cloud computing provider AWS provides a relatively inexpensive file storage (Leoni 2019) and delivery service over the internet, referred to as S3 storage. It acts much like a traditional folder on a computer or server, but the underlying infrastructure is designed for multi-level redundancy and history, with easily managed security and access restrictions. This can be a cost effective mechanism for static file storage and delivery, which also has the ability to scale well beyond the anticipated needs of FDAT. FDAT files that would be served from S3 buckets or equivalent on-premises file storage include (Figure 16):

- Static web files, including JavaScript libraries, images, custom scripting files, and the mapping library framework used in the FDAT web interface;
- System generated shapefiles for user download;
- System generated CSV files for user download; and
- System generated files in the DES format containing updated observation coordinates, for user download.

The different file and folder requirements can be separated into those that are exposed to the internet and those that are only accessible from the RScript Server within the protected network (Figure 17). We discuss this in more detail in the following sections.

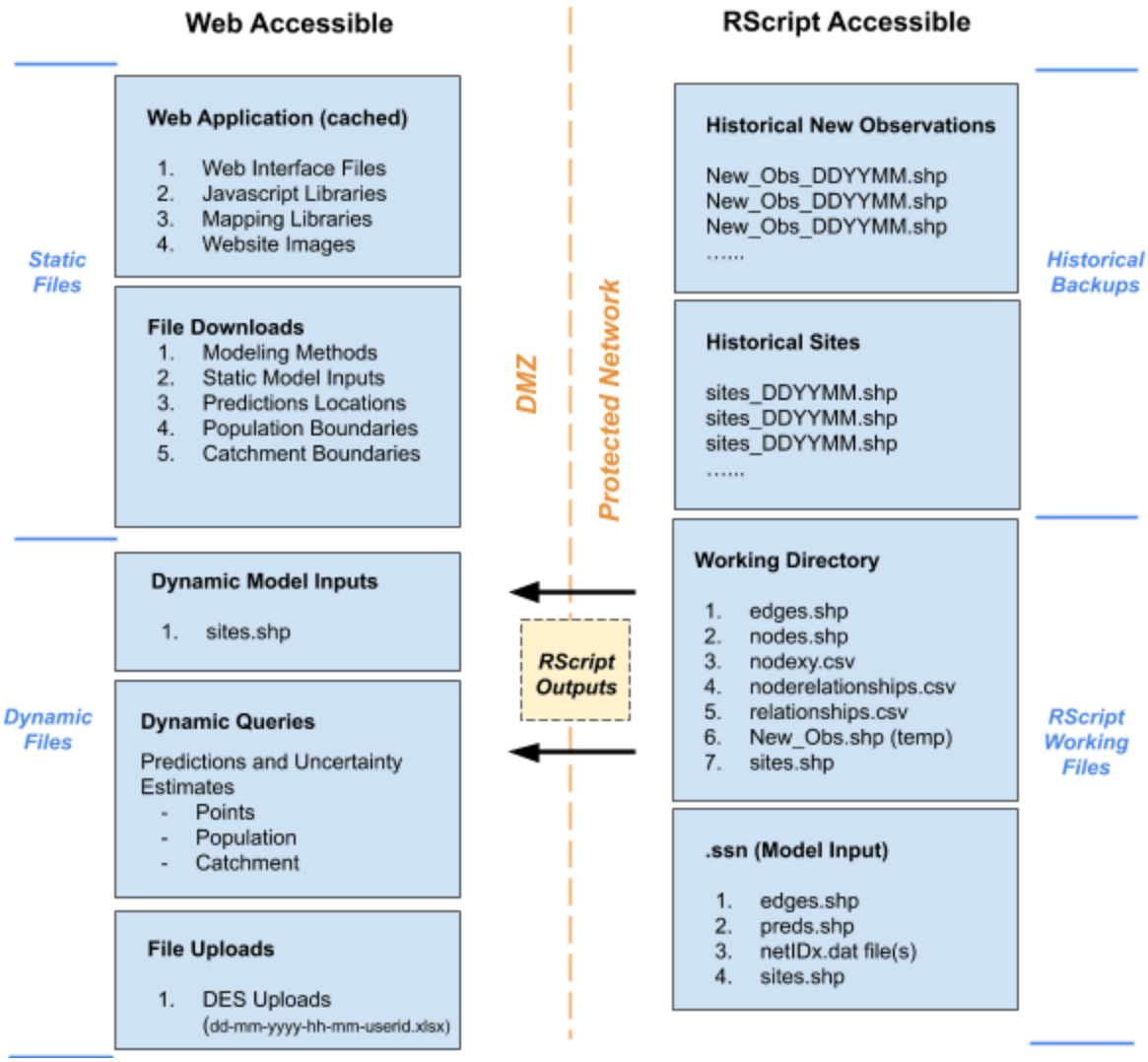

Figure 17. The FDAT storage architecture is divided into the demilitarized zone (DMZ) Web Accessible components and the secured RScript protected network. In the Web Accessible component, static files have read only access, while dynamic files are updated by the RScript server and require read/write access. All files that are RScript accessible require both read/write access internally, but are not accessible from the DMZ.

5.2.1 Static Files & Web Caching

Static files in a web application are typically files delivered to the client's web browser which do not change over time (Figure 17). Files and folders containing static files will be set with read-

only access within the server environments. There are occasions when core code libraries will be updated, but generally these files are also considered static. In this application, static files include:

Code / JavaScript Libraries:

- Mapping Provider JavaScript Library;
- HTML / NodeJS source files;
- CSS files / Framework;
- Custom FDAT JavaScript libraries; and
- 3rd party user interface libraries.

Images:

- Web Interface images.

File Downloads:

- Modeling methods (pdf).

Mapping Data

- Stream network shapefiles;
- Catchment boundaries shapefiles;
- Prediction sites; and
- Population boundaries.

Static files provide the opportunity to utilize caching within the Web server environment. In other words, static files can be served from memory rather than the disk itself, which improves the speed of data delivery and reduces server workload (Ellingwood 2015). This approach is considered best practice for delivery of web assets which do not change over time and require rapid delivery to the client. (Davison 2001) The caching of static files provides a significant improvement in file delivery speed in balance with the cost limitations of the project design.

5.2.2 Dynamic Files

The dynamic nature of FDAT means that some underlying files will change over time as new RScript tasks are completed. These files are delivered to the various read/write folders from the RScript Server, with appropriate permission required (Figure 17).

Registered users can also upload observation data to the web-based FDAT in the DES format. As mentioned previously, user uploaded files pose a security risk because they can be used to execute files within the Web server environment or gain access to secure areas within the system. We have addressed this in the design by setting read/write permissions only on the folders required, isolating user contributed files within their own folder located outside the

protected network, checking the multipurpose internet mail extensions (MIME) types of uploaded files, and renaming the files on receipt. The files will also be parsed to ensure they are Microsoft Excel files in the expected DES format. Any failure of these checks will result in the file being rejected and deleted from the storage folder. All successful uploads will be retained in the File Uploads folder, using a timestamp to rename them appropriately (Figure 17).

For the purposes of the Snap Points web interface (Section 4.3.2), the files are read into memory on the Web server and delivered back to the client via the data API services, which only requires the file to be read once. These files will not be cached, but instead stored in a slightly slower access file folder, accessible on the internet. Note that the speed of file access for this folder is not as important as speed of access for files served to the user via the web interface because they are not regularly accessed. The data files uploaded into the system in DES format will also be available for download, with two additional columns added describing the new coordinates for the observation sites, after the snapping has occurred. However, the data file is only made available to the user who uploaded it. These files can be generated dynamically and streamed to the user's browser session, which does not require file storage.

The ability to perform dynamic queries on the observation and prediction data is required to support interactions within the web interface. Dynamic queries are also necessary for interactive block kriging summaries within user defined areas, if that optional functionality is implemented in FDAT. Some of these functions will also create shapefiles, GeoJSON, or comma separated files (.csv) files dynamically, which will be served from temporary file folders accessible by the Web server, or alternatively streamed directly to the client session from memory.

5.2.3 Historical Backups

Several files will continue to be updated over time, as new observations are uploaded by users and new model outputs generated (Figure 17). When new observations are contributed, a shapefile containing the new observations is created during the Queue Observations RScript task (Figure 14), which is used as an input for the Spatial Data Processing task. Each New_Obs shapefile is stored in the Historical New Observations folder to maintain a full history of observations contributed to FDAT. When new observations are combined with the existing observations in the Spatial Data Processing task (Figure 15), the full set of observation data is stored in an updated sites shapefile. Again, we store the historical versions of this file in the Historical Sites folder (Figures 14 and 16) to allow for error tracking, or in extreme circumstances, the rollback of the base observations, in the event of a processing failure.

All user submitted files, which pass the DES validation steps will be retained in the File Uploads folder (Figure 17), applying a naming convention which identifies the date of upload and the user. These files will remain on the Web server and will only be read from their current location one time when they are parsed on upload to ensure they meet the DES requirements (Section 4.3.1 and Appendix 1).

5.2.4 RScript Working Files

The Working Directory and the .ssn (Model Input) folders (Figure 17) contain both the input and outputs for several tasks and so the system account operating the scheduled tasks requires appropriate read/write permissions. On completion of the final Modeling & Predictions task, output files are also copied to web accessible folders (Figure 16). Permissions in this case will only allow the RScript server to write to the public folders, with no read or write access back to the protected network housing the RScript server (Figures 10 and 16).

5.3 Database Instances

The FDAT database instance will be an enterprise relational database, which provides both read and write functions, triggered by web application interactions and scheduled tasks from the RScript Server (Appendix 2). The instance itself will support partitioning, indexing and replicas for high performance, as well as future proofing for potential data scaling. We recommend using PostGreSQL, but real-world data usage patterns must be tested during development to ensure it is suitable. Other databases might also be considered for larger datasets such as temperature, which is collected at fine temporal scales. In cases such as this, an additional optimized time-series database may be needed to efficiently query data. This is not uncommon in larger scale solutions, where several database instance types are utilized depending on the specific data requirements. For these reasons, the database instance is a flexible component of the application, which can be swapped for another database type or tailored to data usage needs over time as the scope of FDAT grows and/or changes.

5.3.1 High performance replicas

The current FDAT design includes a single database instance, but multiple instances can be used in cases where usage or performance needs exceed these requirements. With modern database servers, additional synchronized copies of a master database (i.e., replicas) are relatively easy to create, especially in a cloud computing environment. These replicas provide additional server resources for handling database queries and are created to cater to the needs of the application, whether it be long running queries, intensive write operations or large-scale dataset delivery. Thus, database replicas optimize hardware for a specific use, providing a cost effective way to achieve greater performance, while minimizing costs.

When considering database performance, the topology of the database instances must reflect the manner in which the application will be used. The main FDAT operations that are computationally intensive include:

- Serving dynamic queries for map interactions, figures and downloadable data;
- Generating shapefiles and GeoJSON files dynamically for subsets of data; and

- Recreating preprocessed data views after completing modeling and prediction tasks.

A replica scenario in FDAT should separate the activities that create and write records from those that only read or scan data; thus assigning database resources to tasks that best suit their configuration (Figure 18). This proposed replica environment will ensure that there is no impact on the user experience in the web application when Spatial Data Processing and Modeling & Prediction tasks are being executed. It also opens up the possibility of deploying more database instances in a clustered environment to improve database performance as the application grows over time.

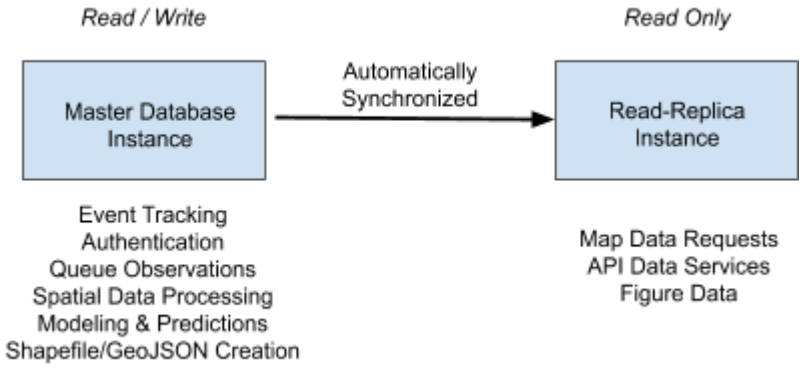

Figure 18. Database replica instances designed to separate read/write functionality from read only functionality.

Database performance and tuning is an important aspect of application design, especially in a data intensive environment such as the FDAT. We have not provided specific recommendations for database table design, indexing or partitioning here. Instead, a developer or database administrator will need to rigorously and iteratively optimize the database performance and underlying hardware allocations during FDAT development.

5.4 Infrastructure Environment

The Web and RScript servers, database instances, and file storage described in the previous sections can be hosted in a variety of ways to cater to the FDAT system architecture requirements. FDAT can be deployed on either a Windows or Linux variant environment, relying on interoperable software solutions such as NodeJS and R statistical software. While the implementation of scheduled processes to support the RScript Server differs slightly between operating systems, it is a routine task using Windows Scheduled Tasks or Linux Cron Jobs.

The implementation specifics are ultimately at the discretion of the developers and system engineers and will largely be driven by the feedback that the BPA provides on this proposed design. However, the FDAT architecture will require the following individual computing resources to serve the website, modeling and data delivery services:

- At least one database instance with large storage support, redundancy, and scalability;
- RScript server optimized for computationally intensive processing, scalability, and on-demand computing;
- Web server with low latency, cached and disk file delivery, and minimal compute; and
- File storage that is low cost, with reasonable performance, redundancy, and scalability.

5.5 Web Interface

The FDAT web application will be designed and built using a single-page application (SPA), which interacts with the web browser by dynamically rewriting the web page with new data from the Web server (Monterio 2014). The SPA design is preferred over a traditional web page since it behaves and responds more like a desktop computer application, but is delivered via a web browser (RubyGarage 2018).

The majority of web interface development will focus on the mapping and exploration interface, with minor supporting pages for authentication, user profiles and project information. The libraries and components to develop the web interface will include:

- Responsive CSS framework;
- Mapping library;
- Mapping provider;
- Graphing library;
- Custom snapping interface;
- Custom API / Data interactions;
- Custom dynamic query interface;
- Authentication and user profile forms; and a
- Data upload interface.

A modern responsive CSS framework will be used to implement the user interface design. The CSS framework allows for rapid development of a web application layout, which is suitable for different screen sizes, devices and browser environments with minimal effort from the developer. Many different CSS frameworks would be suitable for this task including Bootstrap (Otto 2020) and Foundation (Foundation 2020). Therefore, the choice will ultimately depend on the developers' familiarity with particular frameworks.

The mapping library performs the rendering and interactions of the map itself, while the mapping provider supplies the underlying basemap and aesthetic for the map (Appendix 3). We reviewed mapping libraries and providers (Appendix 3) and have recommended the MapBox or CesiumJS mapping library, with the MapBox mapping provider. The interface will include the ability to switch between different underlying basemaps (Appendix 3).

A graphing library will provide visual summaries of data (e.g., bar charts, line charts, etc.) associated with the observations, predictions and estimates of uncertainty at multiple scales. An API will deliver data to render the charts within the web browser dynamically. We reviewed graphing libraries such as Chart.js, Plotly.js and D3.js (Appendix 3), but a variety of different graphing libraries could be used to implement this functionality.

Existing software products will be used to develop FDAT where possible, but custom code is required to implement more specialized functionality. For example, the exact functionality needed for the snapping interface (Section 4.3.2) is not native to any mapping provider, but some provide native functionality that reduces the time needed to develop the custom code (Appendix 3). Thus, the snapping interface will be developed using a custom JavaScript overlay and interface that works in unison with the mapping library, as was the case in FDAT Phase 1 (Peterson et al. 2018; Appendix 3). Custom data API interactions will also be used to connect and deliver data to the mapping interface, graphing interface, file downloads and dynamic query tools. We recommend that all data services be delivered through asynchronous JavaScript requests to an underlying API, ensuring decoupling of the interface and data services. This decoupling of functionality and data supports more specialized development, resource allocation and improved user experience while catering to the exposure of the API as a useful independent service, separate from the web interface. Web form elements and feature selection mechanisms are also required for the custom dynamic query interface (Section 4.2.2.3) on the map itself. Finally, the Authentication and User Profile (Section 4.1) forms will require form elements with input validation and connections into data services to perform authentication and retrieve, update and show profile information.

5.6 Browser Support

Modern browser support is an important consideration in FDAT, given that the public access point for the application is a website. We expect users to have an up-to-date version of Google Chrome, Firefox or the latest Microsoft Edge browser to ensure the best performance and allow the developers to use the latest in browser programming technologies and standards. However, consideration should also be made for users who have limited access to the latest browsers in their computing environment, including advisory messages where this would in some way degrade the user experience. The BPA should provide guidance about browser support if there

are internal security restrictions or recommendations for particular browsers because it may affect the user interface (UI) functionality.

We do not recommend that FDAT be developed for use on mobile devices in Phase 3 due to limitations in performance of mapping libraries with large amounts of spatial data on mobile devices. While the latest JavaScript frameworks generally offer mobile support, it is not sufficient to ensure that the user experience on all mobile devices will be acceptable. Thus, users should be informed that a desktop or laptop device is the preferred method of access. Additional development could be undertaken to offer mobile support in future versions of FDAT, depending on feedback from the BPA and partner organizations regarding this requirement. However, additional software design considerations for mobile devices will likely increase development costs.

References

Access Programmers (2016). Corruption Issues. Access Programmers. [online] Available at: https://access-programmers.com/corruption-issues [Accessed 28 Apr. 2020].

Agafonkin, V. (2019a). Leaflet — an open-source JavaScript library for interactive maps. LeafletJS. [online] Available at: https://leafletjs.com/#features [Accessed 31 Jan. 2020].

Agafonkin, V. (2019b). Plugins - Leaflet - a JavaScript library for interactive maps. LeafletJS. [online] Available at: https://leafletjs.com/plugins.html [Accessed 31 Jan. 2020].

Amazon Web Services (2020a) AWS GovCloud (US) User Guide. [online] Available at: https://docs.aws.amazon.com/govcloud-us/latest/UserGuide/welcome.html [Accessed 29 Apr. 2020].

Amazon Web Services (2020b) Cloud Migration Customers. Amazon Web Services, Inc. [online] Available at: https://aws.amazon.com/cloud-migration/customers/ [Accessed 29 Apr. 2020].

ArcGIS for Developers (2018). Add shapefile | ArcGIS API for JavaScript 3.31. [online] Available at: https://developers.arcgis.com/javascript/3/jssamples/portal_addshapefile.html [Accessed 28 Feb. 2020].

ArcGIS for Developers (2020a). ArcGIS API for JavaScript 4.14. [online] Available at: https://developers.arcgis.com/javascript/ [Accessed 31 Jan. 2020].

ArcGIS for Developers (2020b). Pay as you go pricing. [online] Available at: https://developers.arcgis.com/pricing/credits/ [Accessed 31 Jan. 2020].

ArcGIS for Developers (2020c). Pricing. [online] Available at: https://developers.arcgis.com/pricing/ [Accessed 26 Feb. 2020].

Barbian, M.H. and Assuncão, R.M. (2017) Spatial subsemble estimator for large geostatistical data. Spatial Statistics, 22: 68-88.

Bhatia, S. (2019). PostGreSQL vs. MySQL: Everything You Need to Know. Hackr.io Blog. [online] Available at: https://hackr.io/blog/postgresql-vs-mysql [Accessed on 3 May 2020] .

Bostock, M. (2013). D3.js - Data-Driven Documents. D3js.org. [online] Available at: https://d3js.org/ [Accessed 4 Mar. 2020].

Bostock, M. (2020). Gallery / D3. Observable. [online] Available at: https://observablehq.com/@d3/gallery [Accessed 4 Mar. 2020].

Boston GIS (2008). Cross Compare SQL Server 2008 Spatial, PostGreSQL/PostGIS 1.3-1.4, MySQL 5-6. [online] Available at: http://www.bostongis.com/PrinterFriendly.aspx?content_name=sqlserver2008_postgis_mysql_compare [Accessed 3 Feb. 2020].

Boston GIS (2010). Compare SQL Server 2008 R2, Oracle 11G R2, PostGreSQL/PostGIS 1.5 Spatial Features. [online] Available at: http://www.bostongis.com/PrinterFriendly.aspx?content_name=sqlserver2008r2_oracle11gr2_postgis15_compare [Accessed 3 Feb. 2020].

Bradley, T. (2016). Leaflet to Mapbox GL | Fuzzy Tolerance. [online] fuzzytolerance.info. Available at: http://fuzzytolerance.info/blog/2016/03/16/Leaflet-to-Mapbox-GL/ [Accessed 31 Jan. 2020].

Cesium (2020a) Open Source 3D Mapping: CesiumJS. [online] Available at: https://cesium.com/cesiumjs/ [Accessed on 1 May 2020].

Cesium (2020b) Cesium ion. [online] Available at: https://www.cesium.com/cesium-ion/ [Accessed on 2 May 2020].

Cesium (2020c). Documentation | cesium.com. [online] Available at: https://cesium.com/docs/ [Accessed 28 Apr. 2020].

Cesium (2020d). Cesium ion Pricing Plans. Cesium. [online] Available at: https://cesium.com/pricing/ [Accessed 31 Jan. 2020].

Chart.js (2019). Chart.js | Open source HTML5 Charts for your website. [online] Available at: https://www.chartjs.org/ [Accessed 3 Mar. 2020].

Citus Data (2017). Fermi Estimates On Postgres Performance. [online] Available at: https://www.citusdata.com/blog/2017/09/29/what-performance-can-you-expect-from-postgres/ [Accessed 30 Apr. 2020].

Comeau, S. (2020). Microsoft Access Databases Are Still Popular. Hallam ICS. [online] Available at: https://www.hallam-ics.com/blog/microsoft-access-databases-are-still-popular [Accessed 28 Apr. 2020].

Cressie, N.A.C. (1993) Statistics for Spatial Data. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc. 900 pg.

DB-Engine. (2020). MariaDB vs. Microsoft SQL Server vs. MySQL vs. Oracle vs. PostGreSQL Comparison. [online] Available at: https://db-engines.com/en/system/MariaDB%3BMicrosoft+SQL+Server%3BMySQL%3BOracle%3BPostGreSQL [Accessed 31 Jan. 2020].

Database.Guide (2018). T-SQL vs SQL. [online] Available at: https://database.guide/t-sql-vs-sql/ [Accessed 19 Feb. 2020].

Davison, B.D. (2001). A Web caching primer. IEEE Internet Computing, [online] 5(4), pp.38–45. Available at: https://ieeexplore.ieee.org/document/939449 [Accessed 28 Apr. 2020].

Django Project (2020). GeoDjango Database API | Django documentation | Django. [online] Available at: https://docs.djangoproject.com/en/dev/ref/contrib/gis/db-api/#compatibility-tables [Accessed 28 Apr. 2020].

Ekenes, K. (2020) ArcGIS API for JavaScript: Mapping large datasets on the web. ArcGIS Blog, Environmental Systems Research Institute. [online] Available at: https://www.esri.com/arcgis-blog/products/js-api-arcgis/mapping/mapping-large-datasets-on-the-web/ [Accessed on 2 May 2020].

Ellingwood, J. (2015). Web Caching Basics: Terminology, HTTP Headers, and Caching Strategies. DigitalOcean. [online] Available at: https://www.digitalocean.com/community/tutorials/web-caching-basics-terminology-http-headers-and-caching-strategies [Accessed 28 Apr. 2020].

ESRI (Environmental Systems Research Institute) (2019). ArcGIS Desktop: Release 10.7.1. Technical report, Environmental Systems Research Institute, Redlands, California.

ESRI (Environmental Systems Research Institute) (2020a) ArcGIS API for JavaScript. Environmental Systems Research Institute, ArcGIS for Developers. [online] Available at: https://developers.arcgis.com/javascript/ [Accessed on 2 May 2020].

ESRI (Environmental Systems Research Institute) (2020b) ArcGIS Developer Subscription Pricing [online] Available at: https://developers.arcgis.com/pricing/ [Accessed on 2 May 2020].

ESRI (Environmental Systems Research Institute) (2020c) Create apps from maps, ArcGIS Online. Environmental Systems Research Institute. [online] Available at: https://doc.arcgis.com/en/arcgis-online/create-maps/create-map-apps.htm [Accessed on 2 May 2020].

ESRI (Environmental Systems Research Institute) (2020d). ArcGIS Living Atlas of the World. Environmental Systems Research Institute. [online] Available at:

https://livingatlas.arcgis.com/en/browse/#d=2&type=layers&cont=esriOnly&categories=Basemaps:11111 [Accessed on 2 May 2020].

FMS (2010). When and How to Upsize Microsoft Access Databases to SQL Server. [online] Available at: http://www.fmsinc.com/MicrosoftAccess/SQLServerUpsizing/how/index.htm [Accessed 28 Apr. 2020].

Foundation (2020). The most advanced responsive front-end framework in the world. | Foundation. [online] Available at: https://get.foundation/ [Accessed 28 Apr. 2020].

Geographic Information Systems Stack Exchange. (2013). How can I get a shapefile from a PostGIS query? [online] Available at: https://gis.stackexchange.com/questions/55206/how-can-i-get-a-shapefile-from-a-postgis-query [Accessed 3 Feb. 2020].

GitHub. (2019a). (ArcGIS API for JavaScript) Add Shapefile from disk to a map in 4.x. [online] Available at: https://gist.github.com/nommuna2/93bc71e90e44b8aad72291f41207fcd9 [Accessed 28 Feb. 2020].

Holzgraefe, H. (2016). GIS features in MariaDB and MySQL: What has happened in recent years? Free Software and Open Source Conference (FrOSCon), Sankt Augustin, Germany. August 20-21, 2016. Available at: https://programm.froscon.de/2016/system/event_attachments/attachments/000/000/388/original/MariaDB_MySQL_GIS_Features.pdf.

GDA/OGR Contributors (2020). GDA/OGR Geospatial Data Abstraction software Library. Open Source Geospatial Foundation. https://gdal.org.

Hsu, L.S. and Obe, R.O. (2019). What is a spatial database? Chapter 1 In: PostGIS in Action, Third Edition Manning Early Action Program(MEAP) [online] Available at: https://livebook.manning.com/book/postgis-in-action-third-edition/chapter-1/v-1/74 [Accessed 25 Feb. 2020].

ILAQH (International Laboratory for Air Quality and Health) (2020). Establishing advanced networks for air quality sensing and analysis. International Laboratory for Air Quality & Health. [online] Available at: https://research.qut.edu.au/ilaqh/projects/koala-sensors/ [Accessed 1 May 2020].

Isaak, D.J., Ver Hoef, J.M., Peterson, E.E., Horan, D., and Nagel, D. (2017). Scalable population estimates using spatial-stream-network (SSN) models, fish density surveys, and national geospatial database frameworks for streams. Canadian Journal of Fisheries and Aquatic Sciences 74:147-156.

Leoni, K. (2019). Cloud vs On-Premises Pros and Cons. Heroix. [online] Available at: https://blog.heroix.com/blog/cloud-vs-on-premises-pros-and-cons [Accessed 28 Apr. 2020].

Mapbox (2013). Visualizing 3 Billion Tweets - maps for developers. Medium. [online] Available at: https://blog.mapbox.com/visualizing-3-billion-tweets-f6fc2aea03b0 [Accessed 30 Apr. 2020].

Mapbox (2015). Massive data in Mapbox Studio Classic with PostGIS + SQL. [online] Medium. Available at: https://blog.mapbox.com/massive-data-in-mapbox-studio-classic-with-postgis-sql-ce554bcebd6a [Accessed 30 Apr. 2020].

Mapbox (2020a) Mapbox GL JS. Mapbox|Docs. [online] Available at: https://docs.mapbox.com/mapbox-gl-js/overview/ [Accessed on 1 May 2020].

Mapbox (2020b) Mapbox Maps. [online] Available at: https://www.mapbox.com/maps [Accessed on 2 May 2020].

Mapbox (2020c). Mapbox Pricing. [online] Available at: https://www.mapbox.com/pricing/ [Accessed 31 Jan. 2020].

Mapbox (2020d). How Mapbox Works. [online] Available at: https://docs.mapbox.com/help/how-mapbox-works/ [Accessed 31 Jan. 2020b].

Mapbox (2020e). Introduction - API Documentation. [online] Available at: https://docs.mapbox.com/api/ [Accessed 31 Jan. 2020].

Marsh, J. (2017). SQL vs. MySQL: Which Relational Database is Right For You? Upwork. [online] Available at: https://www.upwork.com/hiring/data/sql-vs-mysql-which-relational-database-is-right-for-you/ [Accessed 31 Jan. 2020].

McLaughlin, C., Hutson, H., De Vine, L., Woodley, A., Geva, S., Chappell, T., Kelly, W., Boles, W., Perrin, D. (2019). Change Detection over the State of Queensland using High Resolution Planet Satellite Mosaics, In: Proceedings of the 2019 Digital Image Computing: Techniques and Applications (DICTA), Perth, Australia. December 2-4, 2019.

Merenych, S. (2019). Relational vs non-relational databases: advantages and disadvantages - Clockwise Software. [online] Available at: https://clockwise.software/blog/relational-vs-non-relational-databases-advantages-and-disadvantages/ [Accessed 19 Feb. 2020].

Microsoft. (2010). Access 2010 web databases roadmap. [online] Available at: https://support.microsoft.com/en-us/office/access-2010-web-databases-roadmap-f778cc30-bfa3-4be4-bab2-55ef4b50e814 [Accessed 28 Apr. 2020].

Microsoft (2020) Microsoft Bing Maps. [online] Available at: https://www.microsoft.com/en-us/maps/web [Accessed on 2 May 2020].

Miller, T. (2018). How Does Indexing Work. Chartio. [online] Available at: https://chartio.com/learn/databases/how-does-indexing-work/ [Accessed 3 Mar. 2020].

Monteiro, F. (2014). Learning Single-page Web Application Development.

Packt Publishing. 214p. Available at: https://www.packtpub.com/web-development/learning-single-page-web-application-development

MySQL (2020a). 5.1.8 Server System Variables. [online] Available at: https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_max_connections [Accessed 25 Feb. 2020].

MySQL (2020b). MySQL 8.0 Reference Manual, 15.18.2 InnoDB Recovery. [online] Available at: https://dev.mysql.com/doc/refman/8.0/en/innodb-recovery.html#innodb-crash-recovery [Accessed 28 Apr. 2020].

Netek, R., Brus, J., and Tomecka, O. (2019). Performance Testing on Marker Clustering and Heatmap Visualization Techniques: A Comparative Study on JavaScript Mapping Libraries. International Journal of Geo-Information, 8(8):348-362.

OneFishTwoFish (2019). Juvenile and Adult Density Distributions. [online] Available at: http://www.onefishtwofish.net/viz/JuvAdultDensity.html [Accessed 21 Feb. 2020].

Opengate Software (2010). When Should I Use MS Access, Getting Started With Access. [online] Available at: https://www.opengatesw.net/ms-access-tutorials/What-Is-Microsoft-Access-Used-For-2.htm [Accessed 28 Apr. 2020].

OpenJS Foundation (2020). Node.js. [online] Available at: https://nodejs.org/en/ [Accessed 27 April 2020].

OpenLayers (2019). Ol-Cesium. [online] Available at: https://openlayers.org/ol-cesium/ [Accessed 21 Feb. 2020].

OpenLayers (2020). OpenLayers v6.2.1 API. [online] Available at: https://openlayers.org/en/latest/apidoc/ [Accessed 2 Mar. 2020].

OpenStreetMap Contributors (2020). OpenStreetMap. [online] Available at: https://www.openstreetmap.org/ [Accessed 2 Mar. 2020].

OpenStreetMap Foundation (2020). OpenStreetMap Foundation, Operations Working Group. [online] Available at: https://operations.osmfoundation.org/ [Accessed 3 May 2020].

Otto, M. (2020). Bootstrap. [online] Available at: https://getbootstrap.com/ [Accessed 28 Apr. 2020].

Otwell, J. (2018). MySQL in 2018: What's in 8.0 and Other Observations. Severalnines. [online] Available at: https://severalnines.com/blog/mysql-2018-what-s-80-and-other-observations [Accessed 24 Feb. 2020].

Peterson, E.E., and Ver Hoef, J.M. (2014). STARS: an ArcGIS toolset used to calculate the spatial information needed to fit spatial statistical models to stream network data. Journal of Statistical Software, 56:1–17.

Peterson, E.E., Isaak, D.J., Ver Hoef, J.M., Nagel, D., and Woodley, A. (2018). Analysis of spatial stream networks for salmonids. Technical report to the Bonneville Power Administration, BPA Contract #77234, 77246, and 46273 REL 13. Queensland University of Technology, USDA Forest Service, and the US National Oceanic and Atmospheric Administration. 56p.

Pineault, D. (2016). Why MS Access isn't a Secure Database. DEVelopers HUT. [online] Available at: https://www.devhut.net/2016/08/19/why-ms-access-isnt-a-secure-database/ [Accessed 28 Apr. 2020].

Planet (2012) Geometries, Chapter 9 In: Introduction to PostGIS [online] Available at: http://postgis.net/workshops/postgis-intro/geometries.html [Accessed 30 Apr. 2020].

Plotly (2020). Plotly JavaScript Graphing Library. [online] Available at: https://plot.ly/javascript/ [Accessed 4 Mar. 2020].

PostGIS (2020) PostGIS: Spatial and Geographic objects for PostGreSQL. [online] Available at: https://postgis.net/ [Accessed on 1 May 2020].

PostGreSQL (2020a). Section 19.3. Connections and Authentication, Chapter 9 Server Configuration, In: PostGreSQL 12.2 Documentation. [online] Available at: https://www.PostGreSQL.org/docs/current/runtime-config-connection.html [Accessed 25 Feb. 2020].

PostGreSQL (2020b) PostGreSQL: The World's Most Advanced Open Source Relational Database. [online] Available at: https://www.PostGreSQL.org/. [Accessed on 1 May 2020].

PostGreSQL Tutorial (2020). PostGreSQL vs. MySQL: A Comprehensive Comparison. [online] Available at: https://www.PostGreSQLtutorial.com/PostGreSQL-vs-mysql/ [Accessed 3 Feb. 2020].

Potter, J. (2020). NPM Trends: Compare NPM package downloads. [online] Available at: https://www.npmtrends.com/cesium-vs-mapbox-gl-vs-mapbox.js-vs-openlayers-vs-leaflet [Accessed 30 Apr. 2020].

Powell, J., Sager, N., and Gup, A. (2020) What's New in ArcGIS API for JavaScript (April 2020). ArcGIS Blog. [online] Available at: https://www.esri.com/arcgis-blog/products/js-api-arcgis/announcements/whats-new-in-arcgis-api-for-javascript-april-2020/ [Accessed on 2 May 2020].

QUT Design Lab (2020). RAISE: Rapid Analytics Interactive Scenario Explorer Toolkit – QUT Design Lab. [online] Available at: https://research.qut.edu.au/designlab/projects/raise/ [Accessed 30 Apr. 2020].

R Core Team (2020). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org

Richardson, B. (2018). Managing Maximum Number of Concurrent Connections in SQL Server. MSSQL Tips. [online] Available at: https://www.mssqltips.com/sqlservertip/5766/managing-maximum-number-of-concurrent-connections-in-sql-server/ [Accessed 25 Feb. 2020].

RubyGarage (2018). What's the Difference Between Single-Page and Multi-Page Apps.RubyGarage. [online] Available at: https://rubygarage.org/blog/single-page-app-vs-multi-page-app [Accessed 28 Apr. 2020].

Sarig, M. (2019). MariaDB vs MySQL in 2019: Compatibility, Performance, and Syntax. Panoply. [online] Available at: https://blog.panoply.io/a-comparative-vmariadb-vs-mysql [Accessed 31 Jan. 2020].

Seeger, M. (2018). Visualizing Large Scale 3D Terrain with Open Source Tools. HERE Developer. [online] Available at: https://developer.here.com/blog/visualizing-large-scale-terrain-with-open-source-tool-tin-terrain [Accessed 30 Apr. 2020].

Shabalin, A.A. (2018). Filematrix: file-backed matrix class with convenient read and write access. R package version 1.3. https://CRAN.R-project.org/package=filematrix

Soto, C. (2019) 12 of the biggest spreadsheet fails in history. Oracle Small-to-Medium Business Blog. [online] Available at: https://blogs.oracle.com/smb/10-of-the-costliest-spreadsheet-boo-boos-in-history [Accessed 1 May 2020]

Sql Programmers. (2014). Disadvantages of Microsoft Access. SQL Programmers. [online] Available at: https://sql-programmers.com/disadvantages-of-access [Accessed 28 Apr. 2020].

Stähli, L. (2017). Cesium vs. ArcGIS API for JavaScript. Institute of Cartography and Geoinformation, Swiss Federal Institute of Technology, Zürich, Switzerland. 20 p. http://www.ika.ethz.ch/studium/cartography_lab/2017_staehli_report.pdf

Stack Overflow. (2014). GIS: PostGIS/PostGreSQL vs. MySql vs. SQL Server? (Answer). [online] Available at: https://stackoverflow.com/a/22576304 [Accessed 3 Feb. 2020].

Stack Overflow (2016a). Mapbox Icons/Markers "BearingSnap" or Snap to Position. [online] Available at: https://stackoverflow.com/questions/39417535/mapbox-icons-markers-bearingsnap-or-snap-to-position/44798270 [Accessed 21 Feb. 2020].

Stack Overflow (2016b). javascript - Mapbox GL JS vs. Mapbox.js. [online] Available at: https://stackoverflow.com/questions/35069753/mapbox-gl-js-vs-mapbox-js [Accessed 19 Feb. 2020].

Stack Overflow. (2017). javascript - CesiumJS: Draggable points jump to static point locations. [online] Available at: https://stackoverflow.com/questions/43666879/cesiumjs-draggable-points-jump-to-static-point-locations [Accessed 28 Feb. 2020].

Stack Overflow (2018). sql - Understanding MySQL licensing. [online] Available at: https://stackoverflow.com/questions/49888007/understanding-mysql-licensing [Accessed 31 Jan. 2020].

Stepnov, E. (2019) Top JavaScript Maps API and Libraries. Flatlogic. [online] Available at: https://flatlogic.com/blog/top-javascript-maps-api-and-libraries/ [Accessed on 2 May 2020].

Quinn, S., and Dutton, J.A. (2018). What is a web mapping API? Open Web Mapping. [online] Available at: https://www.e-education.psu.edu/geog585/node/763 [Accessed 31 Jan. 2020].

U.S. Department of Labor Statistics (2019). Occupational Employment and Wages, May 2019: 15-1244 Network and Computer Systems Administrators. [online] Available at: https://www.bls.gov/oes/current/oes151244.htm [Accessed 30 Apr. 2020]

Virtual Reef Diver (2020). Virtual Reef Diver. [online] Available at: https://www.virtualreef.org.au/ [Accessed 30 Apr. 2020].

Ver Hoef, J.M. (2008). Spatial methods for plot-based sampling of wildlife populations. Environmental and Ecological Statistics 15:3-13.

Ver Hoef, J.M., and Peterson, E.E. (2010). A moving average approach for spatial statistical models of stream networks. Journal of the American Statistical Association 105: 6–18.

Ver Hoef, J.M., Peterson, E.E., Clifford, D., and Shah, R. (2014). The SSN Package: An R package used to fit spatial statistical models to stream network data. Journal of Statistical Software, 56(3), 1-45.

Ver Hoef, J.M., Peterson, E.E., and Isaak, D.J. (In Prep). Data partitioning for spatial linear models when analyzing large datasets. To be submitted to: Journal of Agricultural, Biological, and Environmental Statistics.

Ver Hoef, J.M., Peterson, E.E., and Theobald, D.M. (2006). Spatial Statistical Models that Use Flow and Stream Distance, Journal of Agricultural, Biological and Environmental Statistics, 13, 449-464.

Wikibooks (2020). JET Database/Creating and connecting. Wikibooks, open books for an open world. [online] Available at: https://en.wikibooks.org/wiki/JET_Database/Creating_and_connecting [Accessed 28 Apr. 2020].

Wilhelmsen, C. (2015). Table Partitioning in SQL Server - The Basics. [online] Available at: https://www.cathrinewilhelmsen.net/2015/04/12/table-partitioning-in-sql-server/ [Accessed 4 Mar. 2020].

Woodley, A., McLaughlin, C., Hutson, H., Geva, S., Chappell, T., Kelly, W., Perrin, D., Boles, W., De Vine, L. (2019). High resolution change detection using Planet mosaic. Proceedings of the 2019 IEEE International Geoscience and Remote Sensing Symposium, Yokohama, Japan. July 28-August 7, 2019. Pages 6578-6581.

Ziemann, M., Eren, Y., and El-Osta, A. (2016). Gene name errors are widespread in the scientific literature. Genome Biology, 17, 177.

ZWorkstations (2020). HP Z Workstations Customized for Ultimate Performance. [online] Available at: https://zworkstations.com/ [Accessed 30 Apr. 2020].

Appendix 1. Data Exchange Standard

The following section documents the general field and fielder headers (i.e., column names) contained within the draft data exchange standard (DES) Excel file, as well as the type of data contained in each column, whether it is required, and a description. The details are provided here because it influences the data entry needs of the application.

Project Information

General Field: Org

1. Field header: “Org”
    - Value type: Text
    - Required: No
    - The name of the organization that conducted the work.

General Field: Program/Project name

1. Field header: “Program name”
    - Value type: Text
    - Required: No, but preferred
    - The name of the program or group in the above organization that conducted the work.
2. Field header: “Project name”
    - Value type: Text
    - Required: No, but preferred
    - The name of the project that involved completing the work.

General Field: Program / Project ID Number

1. Field header: “Program / Project ID number”
    - Value type:
    - Required: No
    - The reference number for the contracted work.

General Field: Biologist name/Data owner

1. Field header: “Biologist name / Data Owner / Contacts”
    - Value type: Text
    - Required: Yes
    - The name of the party or parties associated with this data collection event (DCE) that are the main point of contact.
2. Field header: “Contact phone”
    - Value type: Number (formatted as a phone number)

    - Required: No
    - The phone number that the above person can be contacted with.
3. Field header: “Contact email”
    - Value type: Text (formatted as an email address)
    - Required: No
    - The email address that the above person can be contacted with.

General Field: Data access

1. Field header: “Download URL”
    - Value type: Text
    - Required: Yes
    - Equation: =VLOOKUP(J6,Data_System!$A$2:$B$325,2,FALSE)
    - An URL that points to a downloadable version of the dataset containing the measurement data for this DCE, if one is available. This can be supplied by entering the URL, or selecting it from a list of values in the “Data_System” worksheet.
2. Field header: “Data repository name”
    - Value type: Text
    - Required: Yes
    - The name of the data system storing the data related to this DCE.

General Field: Protocol

1. Field header: “Protocol / Method name”
    - Value type: Text
    - Required: Yes
    - The name of the protocol used for the work from monitoringresources.org, or a citation of a document that details the protocol.
2. Field header: Protocol reference
    - Value type: Text
    - Required: No
    - An URL that points to the above protocol document.
3. Field header: Survey method
    - Value type: Text
    - Required: No
    - The name of the method used to perform the work. This will be selected from a list of options.

General Field: Sample Design

1. Field header: “Sample design reference”
    - Value type: Text

    - Required: No
    - An URL that points to a document detailing the sample design, or a citation of a document that details the sample design.
2. Field header: “Sample design type”
    - Value type: Text
    - Required: No
    - The name of the sample design used for the work. This will be selected from a list of options.
3. Field header: “Survey type”
    - Value type: Text
    - Required: No
    - (?) This will be selected from a list of options.
4. Field header: “Related report link”
    - Value type: Text
    - Required: No
    - An URL that points to a document containing an (annual) report of the work, or a citation of a document that contains the (annual) report.

Site/reach metadata

General Field: Stream/Site name

1. Field header: “Stream/Site name”
    - Value type: Text
    - Required: Yes
    - The common name for the stream where the DCE took place in, or the given name for the site if there is no stream name.

General Field: Geospatial coordinates

1. Field header: “Latitude decimal degrees”
    - Value type: Number
    - Required: Yes
    - The latitude of the site in degrees.
2. Field header: “Longitude decimal degrees”
    - Value type: Number
    - Required: Yes
    - The longitude of the site in degrees.

General Field: Date

1. Field header: “Survey date”
    - Value type: Date (DD/MM/YYYY format)

- Required: Yes
- The date on which the DCE took place.

General Field: Survey Type

1. Field header: “Survey type”
    - Value type: Text
    - Required: Yes
    - (?) This will be selected from a list of options.

Species type

General Field: Species code

1. Field header: “Species code”
    - Value type: Text
    - Required: Yes
    - Equation: =VLOOKUP(Y6,Species_Codes!$C$2:$D$125,2,FALSE)
    - The species code of the species of fish that was measured in the DCE. Can be selected from a list of options from PTAGIS, or through a list of values in the “Species_Codes” worksheet. If a value is entered in the “Species common name: Sampled/observed” column first, this value will be automatically populated.

Species sampled descriptive name

1. Field header: “Species common name: Sampled/observed”
    - Value type: Text
    - Required: Yes
    - Equation: =VLOOKUP(X5,Species_Codes!$A$2:$B$125,2,FALSE)
    - The name of the species of fish that was measured in the DCE. Can be selected from a list of options from PTAGIS, or through a list of values in the “Species_Codes” worksheet. If a value is entered in the “Species code” column first, this value will be automatically populated.

General Field: Species scientific: (Common name)

1. Field header: “Species scientific: Common name)”
    - Value type: Text
    - Required: Yes
    - The general species names for the fish measured in the DCE; made up of its scientific name followed the common name in brackets. This will be automatically populated based on the contents of the “Species code” column.

Size class

General Field: Size classification

1. Field header: “Size classification”
    - Value type: Number (range)
    - Required: Yes
    - The size range of the fish measured in the DCE in centimeters, from minimum to maximum. A set of recommended bins are provided and are preferable to use, but custom values can also be entered.

Abundance

General Field: Number counted

1. Field header: “Number counted”
    - Value type: Number
    - Required: Yes
    - The number of fish measured at the site during the DCE.

General Field: Type: Primary recaptures

1. Field header: “Type: Primary recaptures”
    - Value type: Text
    - Required: No
    - Documentation in text if the above count of fish includes the number of fish that were recaptured (recaptures require separate lines).

General Field: Correction factor/observer error

1. Field header: “Correction factor/observer error”
    - Value type: Number
    - Required: No
    - The multiplier which was used as a correction factor to account for observer error and to support the estimated value of fish.

General Field: Number estimated

1. Field header: “Number estimated”
    - Value type: Number
    - Required: No
    - The estimate of the number of fish extrapolated from the measurements in the DCE.

Sample unit size

General Field: Sample unit size

1. Field header: “Channel class/type”
    - Value type: Text
    - Required: No
    - This column may be used to provide summaries for sites or “channel units”. Larger sites may have more than one classification, for which the value here should be “mixed”. Some data sets may also have more specific habitat classes. If this is left blank, “mixed” is assumed. This can be selected from a list of options.
2. Field header: “Site/reach length”
    - Value type: Number
    - Required: Yes
    - The average length of the site in meters.
3. Field header: “Site/reach average width”
    - Value type: Number
    - Required: Yes
    - The average width of the site in meters.
4. Field header: “Site/Reach average depth”
    - Value type: Number
    - Required: No
    - The average site depth in meters.

Observed count

General Field: Fish/meter

1. Field header: “Fish/meter”
    - Value type: Number
    - Required: Yes
    - Equation: =(AB5/AG5)
    - The number of fish observed per meter. It can be entered manually or calculated based on the “Number counted” column divided by the “Site/reach length” column.

General Field: Fish/meter2

1. Field header: “Fish/meter^2”
    - Value type: Number
    - Required: Yes
    - Equation: =(AB5/(AG5*AH5))

- The number of fish observed per meter squared. It can be entered manually or calculated based on the “Number counted” column divided by the product of the “Site/reach length” column multiplied by the “Site/reach average width” column.

General Field: Fish/meter3

1. Field header: “Fish/meter^3”
    - Value type: Number
    - Required: Yes
    - Equation: =(AB5/(AG5*AH5*AI5))
    - The number of fish observed per meter cubed. It can be entered manually or calculated based on the “Number counted” column divided by the product of the “Site/reach length” column multiplied by the “Site/reach average width” column multiplied by the “Site/reach average depth” column.

Estimated abundance

General Field: Fish/meter

1. Field header: “Fish/meter”
    - Value type: Number
    - Required: Yes
    - Equation: =(AE5/AG5)
    - The number of fish estimated to exist per meter. It can be entered manually or calculated based on the “Number estimated” column divided by the “Site/reach length” column.

General Field: Fish/meter2

1. Field header: “Fish/meter^2”
    - Value type: Number
    - Required: Yes
    - Equation: =(AE5/(AG5*AH5))
    - The number of fish estimated to exist per meter squared. It can be entered manually or calculated based on the “Number estimated” column divided by the product of the “Site/reach length” column multiplied by the “Site/reach average width” column.

General Field: Fish/meter3

1. Field header: “Fish/meter^3”
    - Value type: Number
    - Required: Yes
    - Equation: =(AE5/(AG5*AH5*AI5))

- The number of fish estimated to exist per meter cubed. It can be entered manually or calculated based on the “Number estimated” column divided by the product of the “Site/reach length” column multiplied by the “Site/reach average width” column multiplied by the “Site/reach average depth” column.

User defined fields

The user will also be allowed to add their own columns to the end of the spreadsheet, but these data will not be ingested into FDAT.

Appendix 2. Database Formats

A2.1 Microsoft Access

The Landscape Network (LSN) created by the Spatial Tools for the Analysis of River Systems (STARS) ArcGIS custom toolset (Peterson and Ver Hoef 2014) is stored as a personal geodatabase, which is in Microsoft Access format. However, we made the decision to move away from using ESRI ArcGIS (ESRI 2019) for spatial data processing in future versions of FDAT because the software is proprietary and the personal geodatabase format is not supported anymore. We considered whether Access would be a suitable database for storing FDAT observations and predictions, but decided against it because:

- Microsoft Access is not considered a general Enterprise solution for multi-user architecture, especially for those delivered over the internet. Instead, it is designed for installation on client computers, while Enterprise database solutions are designed for high performance and accessibility and deployed onto a server environment. (Opengate Software 2010)
- Access is incompatible with MacOS and Linux operating systems. Even within Windows, accessing the database with different versions of Windows or Microsoft Access may cause database corruption (SQL Programmers 2014; Access Programmers 2016).
- Access is a file-based database rather than a client server-based database (Comeau 2020). Users must be able to access the database file itself, which is achieved by copying the database file to multiple computers or having the database file available on a shared network drive. Depending on the size of the database, transmitting data over the network can be slow (SQL Programmers 2014). This also means that Access databases can not be (reliably) accessed over the Internet. Access 2010 provided the ability to create a web database, but this function will be discontinued in February 2020 (Microsoft 2010).
- Another consequence of being file-based is that Access databases are more vulnerable to corruption in the event of a hardware or software problem. If Access is unexpectedly closed while data is being written to the database file, the database will enter a corrupted state and will need to be repaired (Access Programmers 2016). Client-server databases do not encounter this problem as often, as the database is separate from the client application on the user's machine, and they automatically perform changes as "transactions" which are rolled back in the event of a problem before they are complete (MySQL 2020a).
- Access databases do not automatically free up space over time if records are deleted. This can cause the database file to become much larger than is required as old deleted

records build up. A “compact” operation must be performed manually in order to free up the space occupied by the deleted records (Wikibooks 2016).

- Access can support multiple users accessing the same file at one time, but the number of users that can access it at once before performance begins to suffer can be relatively low (SQL Programmers 2014); although this depends somewhat on the design of the database (FMS 2010).
- Security on an Access database is not very robust. While users can be given specific restrictions in more complex Access databases, it is easy to open the database file in another program and view the contents, including passwords. Since users must be able to access the file, it is vulnerable to unauthorized copying (Pineault 2016).
- Most client-server databases provide logging of operations automatically. If an incorrect change is made to the database, when that action was performed and by who can be found in the log. Access does not automatically log changes; although this ability can be added manually (FMS 2010).

A2.2 Enterprise Databases

Enterprise level database solutions are designed for large data storage, high performance, and generally have a relational database schema. In this section we explore four popular Enterprise level database solutions, with an emphasis on the characteristics important for the FDAT design and architecture (Table A2.1). All four options could be used for FDAT since they have the ability to support simultaneous users and functionality for partitioning, as well as support for large table and file sizes (Table A2.1). Therefore, we will focus on the licensing and support for spatial data as their main point of difference.

Table A2.1. A comparison of Enterprise databases based on licensing, the ability to export shapefiles, the maximum number of simultaneous users connected to the database, maximum database table size, the ability to partition data, and file size limits.

Database	License	Spatial Support	Maximum # of Users	Maximum Table Size*	Partitioning	File Size Limit*
MySQL/ MariaDB	Open source	Yes, with GDAL	Default 151, maximum 10,000 (MySQL 2020b)	No internal limit, except for MyISAM 256TB[v]	Yes, for InnoDB and NDB[+]	No limit
MS SQL Server	Proprietary	Yes, with GDAL	Default 32767 (Richardson 2018)	No internal limit	Yes	16TB
PostGreSQL	Open source	Yes, with PostGIS	Default 100 (PostGreSQL 2020a)	16TB, can be increased to 64TB	Yes	No limit

* Database file size is still limited by system configuration (e.g., a FAT32 filesystem will limit database files to 4GB).
[v] Can be increased.
+ Network database (NDB)

MySQL

Originally created in 1995 by MySQL AB, MySQL was purchased by Sun Microsystems in 2008, which was then taken over by the Oracle corporation in 2010 (Sarig 2019). Even after this acquisition, MySQL remains available for free and is open source. However, Oracle chooses to keep its own development decisions closed to the public (Sarig 2019). Additional support from Oracle can be purchased, as well as an Enterprise edition that includes tools suitable for the needs of large companies. A license must also be purchased to release MySQL with non-open source software (Stack Overflow 2018).

MySQL comes with some support for spatial data built in (Table A2.1). However, their offering in the past was considered to be inferior to SQL Server and PostGreSQL combined with PostGIS (Stack Overflow 2014), and it was also slower for spatial queries. A representative from MariaDB also suggested that PostGIS (with PostGreSQL) was better for new projects that required spatial data (Holzgraefe 2016). MySQL 8.0 does include improvements to spatial support, such as choosing a spatial reference identifier or coordinate system for spatial data (Otwell 2018). Note that the third-party Geospatial Data Abstraction Library (GDAL; GDA/OGR Contributors 2020) library is required to export spatial data as ESRI shapefiles from MySQL.

Pros:

- Open source, can be modified as required and the modifications distributed for free;

- Different database engines available, such as MyISAM and InnoDB (DB-Engines 2020); and
- Spatial data support is built in.

Cons:

- A commercial license is required if MySQL is to be included in the distribution of a non-open source product (not relevant to FDAT);
- Support from Oracle and Enterprise features must be purchased;
- Dependent on Oracle development decisions, so new features are slow to appear;
- Not as compliant with SQL standards as PostGreSQL (Bhatia 2019); and
- Spatial data support features are considered inferior to that of SQL Server and PostGreSQL + PostGIS.

MariaDB

MariaDB is a version of the MySQL database, created in 2010 after the acquisition of Sun Microsystems by Oracle (Table A2.1). For the most part, MariaDB can function as a drop in replacement for MySQL, but with the added advantage of being open source so that it can be reviewed by the community (Sarig 2019). The description of MySQL spatial support also applies to MariaDB, except that MariaDB had full compliance with the OpenGIS standard version 10.1 compared to MySQL (Holzgraefe 2016).

Pros:

- Unlike MySQL, it is not dependent on development by Oracle.
- Spatial data support is greater for MariaDB version 10.1 compared to MySQL version 5.7.

Cons:

- Enterprise features, 24/7 support, etc. must be purchased
- Potential for divergence between MySQL and MariaDB in the future.

Microsoft SQL Server

Microsoft SQL server is specifically designed for easy integration with applications developed for the Windows platform, such as .net programs (Marsh 2017; Merenych 2019; Sarig 2019). It is a proprietary product and as such, cannot be modified by developers to better fit their needs. The licensing is also stricter and relatively expensive, with only a heavily restricted, single database available for free (Boston GIS 2010; Marsh 2017). However, SQL Server also makes use of Transact-SQL, an extension of regular SQL that provides extra features such as advanced processing of text, mathematical data and dates (Database Guide 2018).

Microsoft SQL Server is similar to MySQL as it includes support for spatial data out of the box. Also like MySQL, an external program such as the GDAL library is required to save spatial data as ESRI shapefiles. However, it has less spatial support features than PostGreSQL combined with PostGIS. For example, support for rasters (e.g., images) and functions for processing GIS data are lacking (Hsu and Obe 2019).

Pros:

- Made to interface with Microsoft .net programs
- Transact-SQL provides additional features for queries not available in other databases
- Better support for spatial data than MySQL.

Cons:

- Must purchase a license to use more than the restricted free version and/or more than one database;
- Not open source, so cannot be modified if required to suit the application;
- No ability to choose a different database engine (DB-engines 2020); and
- Spatial support inferior to PostGreSQL + PostGIS.

PostGreSQL

PostGreSQL (PostGreSQL 2020b) or 'Postgres' was first developed in 1989 and is the oldest of the relational databases discussed here. It has the greatest compliance to SQL standards (Bhatia 2019), a large range of features, and full open sourced development, like MariaDB. This database remains the most advanced of all relational databases due to the rapid, open source development by the Postgres community (Bhatia 2019).

Unlike the previous two databases, PostGreSQL does not come with support for spatial data on its own. Instead, the PostGIS extension (PostGIS 2020) is required to store and use spatial data. However, this is not a disadvantage; PostGIS is the most powerful offering for spatial databases, including the full OpenGIS standard and beyond (Django Project 2020; Holzgraefe 2016). It also includes support for exporting spatial data to ESRI shapefiles without additional libraries like GDAL (Geographic Information Systems Stack Exchange 2013).

Pros:

- Open source, can be modified as required and the modifications distributed for free;
- Better range of features, including enterprise level features such as data analysis, without requiring an enterprise license;
- Greatest support for spatial data when used with PostGIS; and
- Greater compliance with SQL standards (160 out of 179 standard features implemented) compared to MySQL Server/MariaDB.

Cons:

- No ability to choose a different database engine (PostGreSQL Tutorial 2020)
- Spatial support using PostGIS must be installed separately, unlike with MySQL and SQL Server.

A2.3 Indexing and Partitioning

With any choice of enterprise database, the creation of indexes and partitioning within a database design allow for greatly improved performance, depending on the nature of the queries performed. Indexing is intended to speed up the process of retrieving data from the database. When database tables are unindexed, queries on that table must traverse every row to determine which of those rows match the query's conditions. Adding an index on a table column essentially allows that column to be sorted when the table is queried, decreasing the time needed for the query (Miller 2018). The FDAT web interface is a read-only experience, except when new data are uploaded by the user. Thus, indexes will be created to improve read performance, along with a clustered database environment splitting the read-only queries into a separate duplicate database to improve overall system performance and allow granular monitoring of processing resource requirements.

The database tables containing point prediction information for the Columbia River Basin are expected to be large, in the order of millions of rows. Queries on this size table can take a relatively long time to complete and so partitioning will be used to improve performance. Partitioning splits a table into several logical sections, depending on the value of a column (e.g., date). If data from a certain year or month needs to be retrieved, only those partitions relevant to the query will accessed and examined, decreasing the time required to obtain results (Wilhelmsen 2015).

All of the enterprise databases considered here support indexing and partitioning. However, it is difficult to determine a priori which indexing system will be most suited for the FDAT system, given that queries will be needed in both space (e.g., summarize over areas) and time (e.g., summarize data at one location). Thus, further database query performance tests will be undertaken in the FDAT implementation phase to help identify which indexes will best be used.

Appendix 3. Mapping functionality

A web-based mapping provider is a service that allows a developer to view and manipulate data within a map, rather than having to develop their own solution from scratch. These services provide an Application Programming Interface (API), which provides access to common spatial functions such as placing elements on the map or calculating the distance between two points. Being web based, these maps can then be served to users in different locations and on various devices, such as a desktop PC, tablet or phone (Quinn and Dutton 2018).

A3.1 Difference between providers and libraries

There is an important distinction to be made between a mapping provider and a mapping library. A mapping *library* is a software product that supports the manipulation of maps, such as adding data, turning layers on and off, and zooming in and out, and so on. In contrast, a mapping *provider* provides access to a map layer itself, usually through its API. While some providers also provide their own libraries, such as Cesium and Mapbox, there are standalone libraries available like Leaflet and OpenLayers. These libraries still require a map provider before they can be used, and each library will only provide support for a subset of providers.

A3.2 Mapping libraries

CesiumJS

CesiumJS is an open source mapping library developed by the creators of Cesium ion (Table 3.1; Cesium 2020a). Features on a CesiumJS map can be associated with popups using the description tag in the code. This description can contain HTML code as well as plain text, so complex elements can be displayed using these popups such as graphs and figures. The visibility of features on the map can be toggled on and off, which allows certain map elements to be hidden. Features can also be assigned to a “data source” which itself can have its visibility toggled. It is also possible to drag elements around the map using this library, though it is not an inbuilt function (Stack Overflow 2017). There is also no native snapping function, but custom code could be developed to implement this in CesiumJS.

The CesiumJS library can be used with a variety of mapping providers other than Cesium ion (Cesium 2020b), including Bing Maps (Microsoft 2020), Mapbox and OpenStreetMaps (Table A3.1; Cesium 2020c). Although it does not support ESRI shapefiles natively, it does support GeoJSON, KML or TopoJSON files.

Pros:

- Open source and can be modified as needed;

- Allows for popups that contain (complex) HTML;
- Variety of mapping providers can be used;
- Simple to hide and show features; and
- Support for GeoJSON, KML or TopoJSON data formats.

Cons:

- No inbuilt dragging or snapping functions; and
- Can be ongoing costs depending on the mapping provider (Section A3.3).

Table A3.1. A comparison of mapping libraries and the characteristics relevant to FDAT, including whether the software is open source (versus proprietary), the map providers it supports, the existence of a native snapping function, the ability to enable or disable spatial layers in the view, and detailed popups, which allow contain complex HTML elements to support graphs and figures.

Mapping Library	Open Source	Supported Map Providers	Snapping Function	Enable/ Disable Layers	Detailed Popups
CesiumJS	Yes	Cesium Ion, OpenStreetMap, Bing Maps, ArcGIS, Google Earth*, MapBox	No native function for dragging and snapping	Yes	Yes
ArcGIS API JavaScript	No	ArcGIS[+]	Present in version 3, but not 4	Yes	Yes
Leaflet	Yes	OpenStreetMap by default. Plugins available for Bing Maps, ArcGIS, Google Maps, and Mapbox^	Yes, through Leaflet.Snap plugin	Yes	Yes
Mapbox GL JS	Yes	Mapbox	No specific function[#]	Yes	Yes
OpenLayers	Yes	OpenStreetMap, ArcGIS, Cesium Ion (with a library (OpenLayers 2019)), Bing Maps	Yes	Yes	No

* Requires a Google Earth Enterprise server.
[+] ESRI provides its own versions of OpenStreetMap.
^ Official Mapbox plugin for Leaflet is depreciated after Mapbox GL JS development (Bradley 2016).
# Other functions can be used to implement this functionality (Stack Overflow 2016a).

ArcGIS API for JavaScript

The Environmental Systems Research Institute (ESRI) provides several solutions for creating web-based applications that display spatial data. ArcGIS Online provides tools for those with no programming experience to make maps embedded in web pages, or full “apps” using a selection of widgets. However, these do not allow extra code to be added for custom modifications and so its usefulness for FDAT is limited (ESRI 2020c).

For developers, the ArcGIS API JavaScript (ESRI 2020a) can be used to create web applications using ArcGIS map services (Table A3.1). However, this API cannot be modified and as such, it is not an open source library (ArcGIS for Developers 2020a). The JavaScript library does include functions for snapping points to lines during editing, but this feature is only available in version 3 (ArcGIS for Developers 2018) and is not yet implemented in version 4 (ArcGIS for Developers 2020a). The ability to move points is still available in version 4, so it is possible to write custom code to snap the points to nearby lines.

Spatial data layers can also be added either from an online source or created from elements, such as a database within FDAT. These layers can be turned on and off, similar to CesiumJS. The ArcGIS library also supports widgets (i.e., extensions), which can support complex elements such as map popups (e.g., figures and charts) or tools for adding and editing points on a map (ArcGIS for Developers 2020a). Both versions 3 and 4 support the upload of ESRI shapefiles, as long as they are in a zip archive which includes all the associated files (GitHub 2019a). A connection to the ArcGIS REST API is also needed because the process that sets up files for viewing requires functions on the server (ArcGIS for Developers 2018; GitHub 2019a).

The ArcGIS API for JavaScript can be used with a free Essentials plan for developers of non-revenue generating apps, but it is not available for government use. Instead a commercial deployment plan must be purchased for a monthly subscription fee. The Builders plan ($125 per month minimum) includes access to all ArcGIS APIs and software development kits (SDKs), 1 million connections for basemaps and geocoding per month, and content hosting on ArcGIS servers for access, analysis and displaying on maps. It also provides 50 credits that can be exchanged for various services like routing, demographics and data storage. More importantly, it includes technical support from ESRI and access to additional tools such as the AppStudio (ArcGIS for Developers 2020c).

Pros:

- Support for uploading and displaying ESRI shapefiles in a zip archive;
- A large selection of widgets can be added to a map, including editing tools;
- Allows for popups that contain (complex) HTML code; and
- Simple to hide and show features on a map.

Cons:

- Software is proprietary and so there is a cost to using it;

- JavaScript API cannot be modified and so is not open source;
- Feature snapping is not yet implemented in version 4 or the API, but it was present in version 3; and
- Only supports ESRI basemaps.

Leaflet

Leaflet is an open source JavaScript library for mapping, which is not associated with any particular mapping provider (Table 3.1). The library is intended to be lightweight and has the basic features for drawing maps on a webpage, but is also extendable with a large range of plugins contributed by other authors (Agafonkin 2019a). For example, there are plugins available for adding ESRI shapefiles (e.g., Leaflet.Shapefile) and geodatabases (e.g., Leaflet.FileGDB).

Leaflet itself does not include the snapping functionality, but there are at least two plugins that add this functionality (Table 3.1). Leaflet.Geoman is a comprehensive plugin that provides many editing functions such as the ability to add and remove points, while Leaflet.Snap relies on other plugins to implement additional editing functionality. Leaflet also allows elements on a map to be grouped, as well as a simple way to control switching between basemaps and toggling layer visibility. Popups can be quickly added to elements and include HTML, so that complex popups including tables or graphs are possible (e.g., OneFishTwoFish 2019).

By default, Leaflet draws maps using the open source OpenStreetMaps as the basemap. However, a suite of plugins make it possible to use other mapping providers, such as ESRI's ArcGIS and Bing Maps (Agafonkin 2019b). Interestingly, Mapbox also used a modified version of Leaflet for their map service before developing their own library.

Pros:

- Open source and can be modified as needed;
- Allows for popups to contain (complex) HTML;
- Variety of different mapping providers can be used;
- Simple to hide and show features using an inbuilt control;
- Not tied to a paid mapping service like Cesium or ArcGIS; and
- Many plugins available for additional features, such as snapping and uploading shapefiles.

Cons:

- Lightweight library means that many advanced features are relegated to third party plugins.
- Leaflet has performance issues when displaying greater than 100,000 data points (Netek et al. 2019).

Mapbox

As with Cesium and ArcGIS, the Mapbox (Mapbox 2020a) service provides its own library to embed maps into websites (Table A3.1). Originally, Mapbox was built as a plugin for Leaflet, but this is now deprecated in favor of their own library, Mapbox GL JS. This library is open source under the 3-clause Berkeley Software Distribution (BSD) license.

Mapbox GL JS is specifically designed to work with styles created by Mapbox's online tool, Mapbox Studio (Stack Overflow 2016b). It is intended to streamline the process of designing the basic properties of a map, including the color of land areas, font, color, position and orientation of labels, and other elements. Style can also vary based on zoom level. Maps are rendered on the client side using WebGL (Mapbox 2020d.). The Mapbox GL JS library also supports popups containing HTML elements, meaning that a Mapbox map can show tables or graphs. Creating a control to hide and show layers is also possible; albeit not as simple as Leaflet's implementation. Mapbox does not include the snapping functionality, but it is possible to write custom code to do this (Stack Overflow 2016a).

Mapbox GL JS is intended to work with Mapbox basemaps, which include four core sets of data referred to as tilesets. These include Streets, which contain street data and administrative boundaries based on OpenStreetMap; Outdoors containing elevation, terrain, and roads and natural features; Satellite imagery from various sources; and continually updated Traffic information shown on the Streets tileset (Mapbox 2020b).

Pros:

- Various map data tilesets available;
- Online tool Mapbox Studio to allow for maps to be themed as desired; and
- Allows for popups that contain complex HTML.

Cons:

- By default, library requires a Mapbox account for an access token, which means it is subject to Mapbox's usage and payment conditions; though using Mapbox GL JS changes how pricing works;
- Uses Mapbox basemaps only; and
- No inbuilt snapping function.

OpenLayers

OpenLayers is another open source library (under 2-clause BSD) which, like Leaflet, is not tied to a particular mapping provider (Table A3.1). Layer visibility can be easily toggled using a one line function, and so hiding and showing layers is simple after adding the elements to different

layers. While OpenLayers does not provide an inbuilt control for changing layer visibility on the map, the addition of custom controls could be tailored to do this.

Strictly speaking, OpenLayers does not directly supply the functionality for popups. Instead, they are technically overlays, which are triggered when the user clicks on elements. However, there are tutorials that show how to implement this functionality on the OpenLayers website. Unlike the other mapping libraries reviewed here, snapping is supported directly, meaning that a plugin is not required for this functionality.

The OpenLayers library supports several map providers natively, such as ArcGIS (for images), Bing Maps and OpenStreetMaps (OpenLayers 2020). Mapbox maps can also be used in OpenLayers through a generic API call (Mapbox 2020e) and there is an additional library available for integration with Cesium ion (OpenLayers 2019).

Pros:

- Not tied to a proprietary mapping provider and a variety of different mapping providers can be used;
- Simple to show and hide layers; and
- Snapping features included.

Cons:

- No inbuilt control for toggling layers;
- Popups are supported but are more complicated to define compared to some other libraries; and
- OpenLayers has performance issues when displaying datasets with more than 100,000 data points (Netek et al. 2019).

A3.3 Mapping providers

Most data displayed on FDAT will be stored on an FDAT server, along with scripts that process data, and so most additional services that map providers offer are not required. Instead, the main function of the mapping provider will be to provide basemaps that form the background setting so that users can orient themselves on the map easily. Of the four mapping providers we reviewed (Table A3.2), all but OpenStreetMaps provides a wide variety of basemaps that would be suitable for FDAT (Figures A3.1-A3.4). Thus, the main aspect distinguishing the different providers is the cost (Table A3.2). Some providers are free, or free until the number of FDAT users accessing and viewing maps exceeds a threshold, while others have a fixed price per month.

Table A3.2. Mapping providers, billing methods, licensing, and cost estimate in US dollars.

Mapping Provider	Billing method	Licensing	Cost estimate
Cesium ion	Fixed price per month	Commercial	$149+ per month
ArcGIS	Fixed price per month	Commercial	Builders License: $125 per month
Mapbox	Number of requests per month	No separate licensing	Cost varies with use but free for low volume services
OpenStreetMap	Free	Open source	Free

Cesium ion

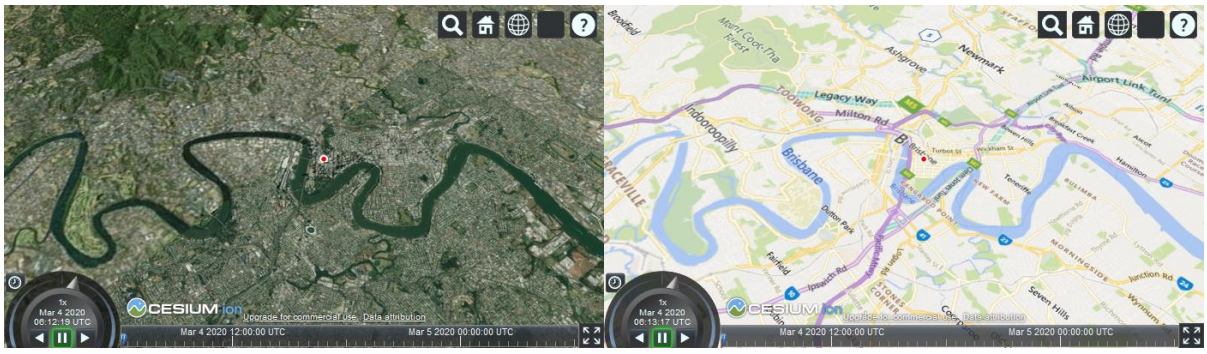

Figure A3.1. An aerial (left panel) and road (right panel) basemap offered by Cesium ion.

Cesium ion (Cesium 2020b), like ArcGIS, is more than a proprietary software service. The Cesium ion engine specializes in the provision of hosted 3D data streaming services, which delivers uploaded data assets as a tiled service. This ensures that a responsive performance is maintained for the end user, while also handling the data delivery, caching and construction of the underlying tiled 3D and geospatial data. Commercial plans are available at a fixed price per month (Table A3.2; Cesium 2020d). Note that there is a free Community plan available, but government agencies do not qualify for it. All commercial plans offer support via email, and depending on the plan, higher limits for data streaming, and the use of Bing Maps imagery and geocoding. Cesium ion also provides customized payment plans, which are directly negotiated with their sales team (Cesium 2020d). Cesium ion does provide access to 5000 free Bing Maps

sessions as part of the fixed monthly price (Table A3.2; Microsoft 2020). However, Bing Maps and 3D visualization are not critical to the development of FDAT. For example, Bing Maps can also be accessed and used directly, without a license for Cesium ion. In addition, FDAT does not currently utilize 3D visualization; although the use of Cesium ion would provide the opportunity to deliver 3D visualization in the future. (Figure A3.1).

ArcGIS

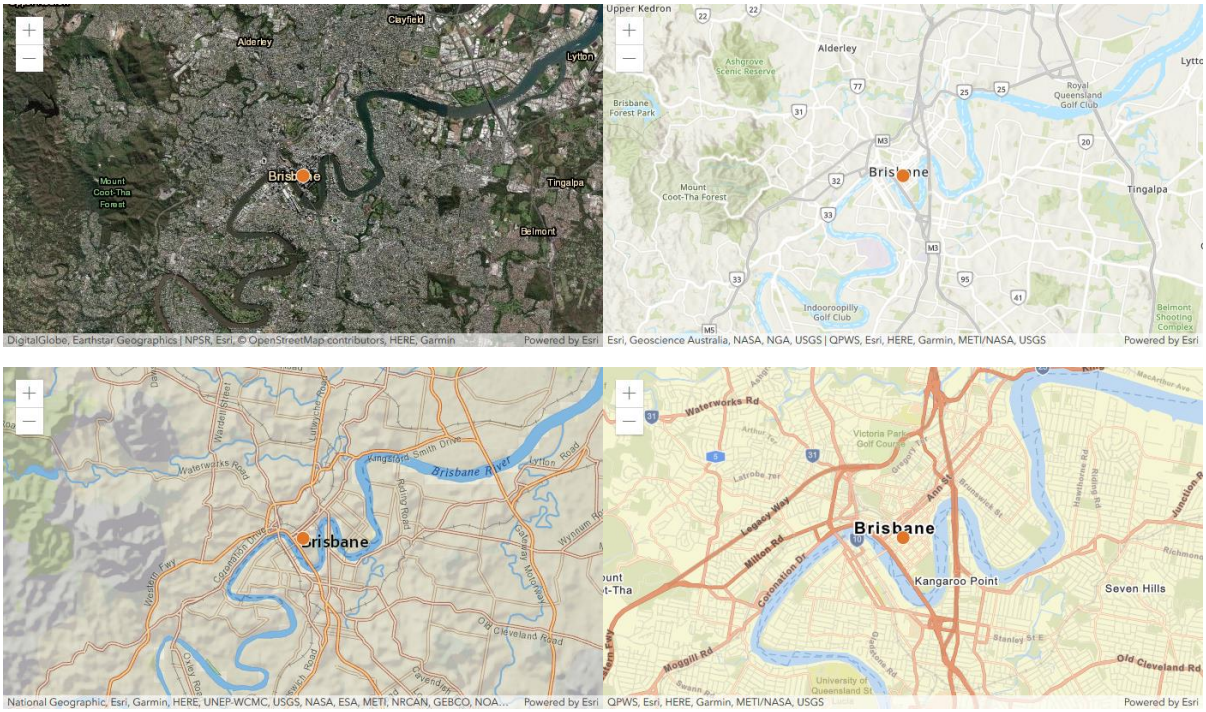

Figure A3.2. Four basemaps offered by ArcGIS including satellite with vector overlay (top left), topographic vector (top right), National Geographic style (bottom left) and the streets vector (bottom right).

The Builders commercial plan (ESRI 2020b) is required for use with the ArcGIS API JavaScript mapping library and the plan includes access to ESRI basemaps (Table A3.2). There are a large number of ESRI basemaps available for use in the ArcGIS API JavaScript (ESRI 2020d), including their own version of OpenStreetMap, topographic basemaps, and a National Geographic style layer, among many others. However, a license is needed to use ESRI basemaps.

Mapbox

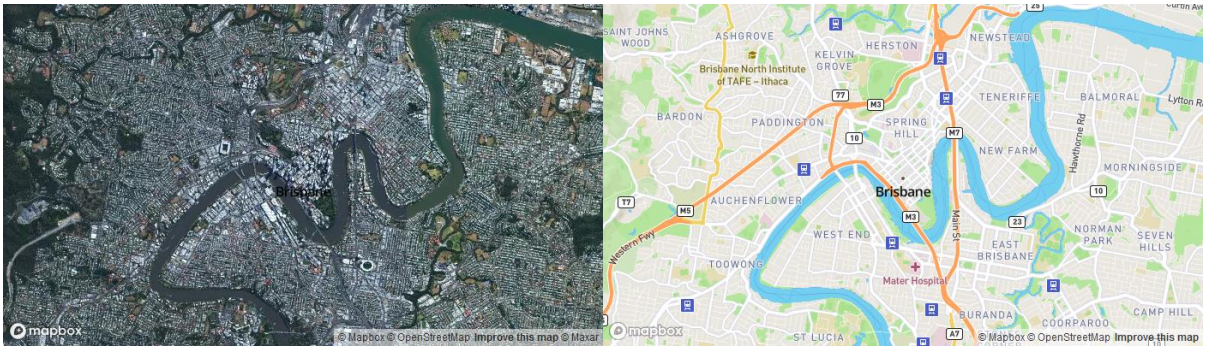

Figure A3.3. The satellite (left) and streets (right) basemaps, or styles, offered by Mapbox.

The Mapbox provider provides a large suite of basemaps that would be suitable for FDAT, including satellite imagery, terrain maps, and street maps (Mapbox 2020b). Unlike some of the other mapping providers, Mapbox maps can be used with Mapbox GL JS or another mapping library. Another advantage is that Mapbox pricing is based on the number of requests, rather than a fixed price per month (Table 4.2; Mapbox 2020c). The way in which the requests are counted depends on how the service is being accessed. If the Mapbox GL JS library (≥ version 1.0.0) is being used, each map load is counted. A map load occurs when a map in the library is initialized, and requests tiles from the map provider service. No additional requests (and therefore, costs) are counted when zooming and panning the map or toggling layers. A map load is only considered active for 12 hours, and so maps viewed for longer than 12 hours count as a new map load. Up to 50,000 requests are provided for free each month, with the next 50,000-100,000 requests costing $5, 100,000-200,000 requests costing $4, and an additional $3 for every 100,000 requests after that (Mapbox 2020c).

If the service is accessed through an older version of Mapbox GL JS or a different mapping library, such as Leaflet, the number of tile requests is measured for pricing (Mapbox 2020c). Therefore, scrolling or zooming on the map increases the number of requests. In addition, raster and vector sources are counted separately, and unless tile sources are composed using Mapbox Studio, the number of requests will increase with multiple layers of Mapbox tiles. However, vector and static tiles (i.e., tiles being generated from GL styles) are free up to 200,000 requests, and raster tiles are free up to 750,000 requests (Mapbox 2020c).

OpenStreetMap

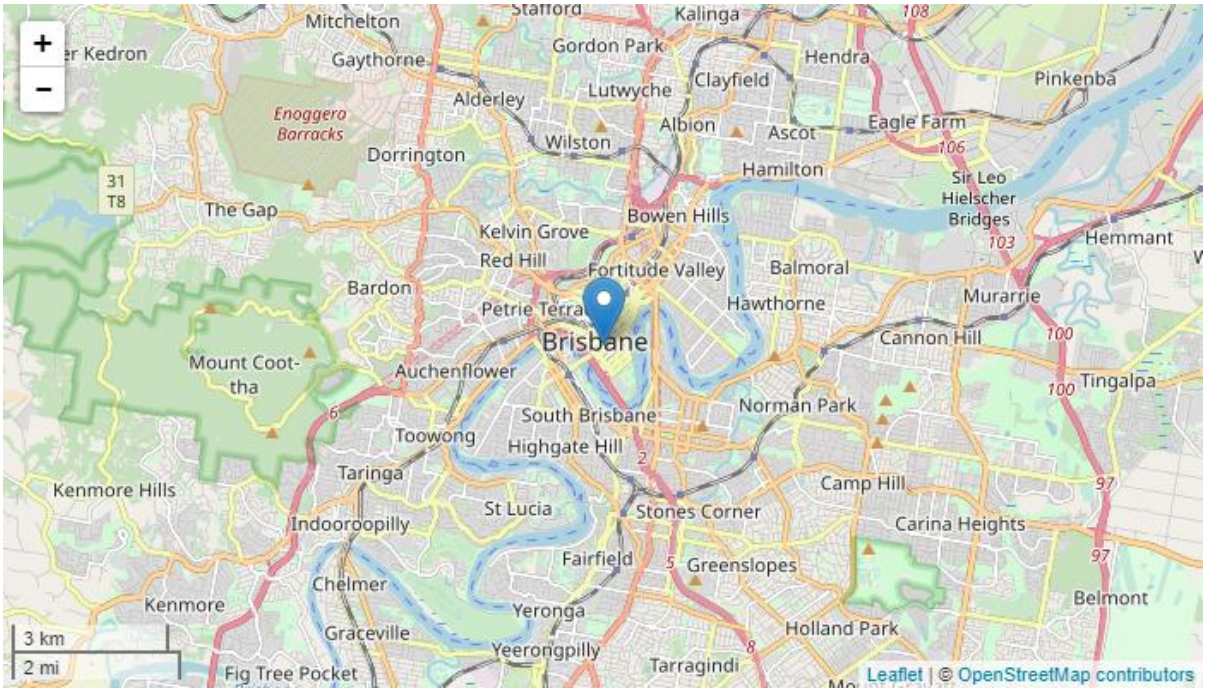

Figure A3.4. A map using OpenStreetMap as the provider, viewed through the Leaflet library. Adapted from an example on Switch2OSM (2020).

OpenStreetMap (OpenStreetMap Contributors 2020) is supported by the mapping libraries discussed so far and is a default for Leaflet. Unlike the previous three map providers, OpenStreetMap is open source and can be used for free (Table A3.2). There are only two conditions; first, correct attribution must be displayed on the map and second, modifications to the maps must also be distributed using the Open Data Commons Open Database License.

Although the name implies street maps, the maps include other features such as country and state boundaries, parks, rivers, bus stops and buildings. In some cases, the data are not complete due to the open source nature, but missing features can be added by anyone so that the maps are continually improved through community input. However, OpenStreetMap does not provide any other services, and more importantly, does not provide satellite imagery. If those are required, then one of the mapping libraries should be used; though OpenStreetMap can be used as an alternate basemap.

It is also important to note that while OpenStreetMap tiles and data are free, access to the tiles from their servers are not (OpenStreetMap Foundation 2020). For production use, either a copy of the OpenStreetMap database must be downloaded and rendered on servers, or another

service that provides the tiles needs to be selected (Switch2OSM 2020). For example, ESRI provides a basemap using OpenStreetMaps tiles.

A3.4 Graphing libraries

Visual data summaries (i.e., graphs) of data over time form an important aspect of the interactive web mapping interface. All of the mapping libraries we reviewed support popups with HTML content and so there are issues with compatibility between he graphing and mapping libraries; though there will be differences in how a graph is shown in each library. Therefore, the graphing library can be selected based on graph appearance and the familiarity of the developer with each library.

Chart.js

Chart.js is designed to make implementation simple, while also offering flexibility in the way graphs are rendered. There are many different graph types that can be used, such as the regular bar, line and pie graphs, and more uncommon types like radar and bubble graphs (Figure A3.5). Some graph types can also be combined into a single graph. In addition, graphs automatically scale to fit the browser window as it is resized.

There are several different configuration options in Chart.js related to graph characteristics such as the color of data sets, the location of the legend and the type of axis scale. The graphing library includes tooltips for data points as well, and the content of these can be modified, so the uncertainty value of predictions can also be displayed with predictions.

To create a graph, a canvas HTML element is required, so popups on the map will need to include such an element to display graphs from this library. Finally, Chart.js is open source under the MIT license and so modifications are allowed if required (Chart.js 2019).

Figure A3.5. A selection of graphs implemented using Chart.js (Chart.js 2019).

D3.js

D3.js is a general library used to add data into webpages, rather than a dedicated graphing library (Figure A3.6). Nevertheless, D3.js provides a suite of tools to draw graphs since they are one of the most common ways to visualize data. D3.js provides a great deal of control over elements created and selected on a web page, as well as attributes such as their color. Thus, it can potentially be used to create unique custom graphs that also include functions such as zooming and panning; though the process of doing so may be complex.

The D3.js library does not directly support tooltips on the data points within graphs it creates, which could be used to display uncertainty estimates along with predictions. However, this functionality can be achieved using alternative methods such as creating an element to show and position data when the user hovers or clicks on part of the graph. Graphics can also be added to different types of elements on a webpage, so D3.js is not restricted to div or canvas HTML elements, like the previous two libraries (Bostock 2013).

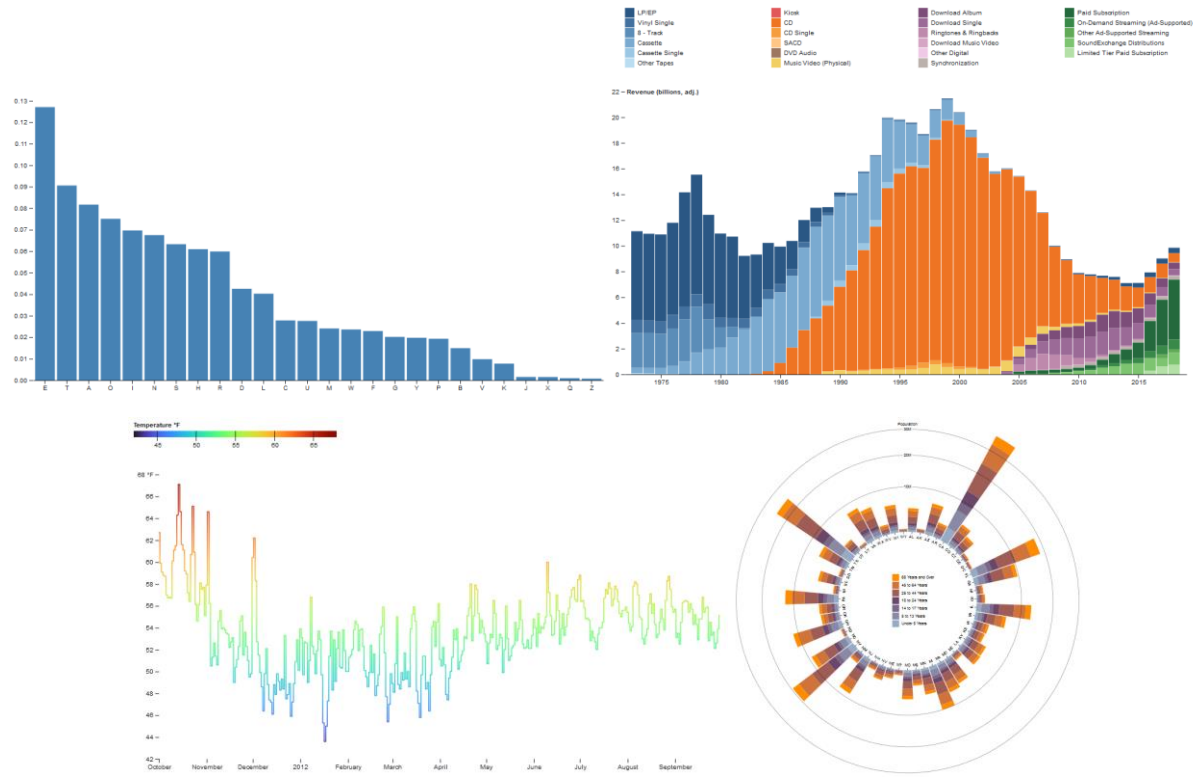

Figure A3.6. A selection of graphs generated using D3.js (Bostock 2020).

Plotly.js

Plotly.js is another open source offering, that has been built on other libraries such as D3.js, and is designed to draw SVG elements. Unlike Chart.js, Plotly.js can create 3D graphs as well as regular 2D graphs. Plotly.js also makes it simple to create a basic graph, while offering many additional customization options (Figure A3.7). There are also advanced options available that are not present in Chart.js (e.g., a selection of scientific and financial charts). Drop down menus can be added to graphs so that the user can select which data set to show, lasso selections can be used to obtain information about certain points, and a subplot can be added that provides additional or more detailed data. Plotly.js provides configurable popups as well, so the uncertainty value can be shown for prediction. However, a div HTML element is required, so map popups will need to include this element to display graphs from this library (Plotly 2020).

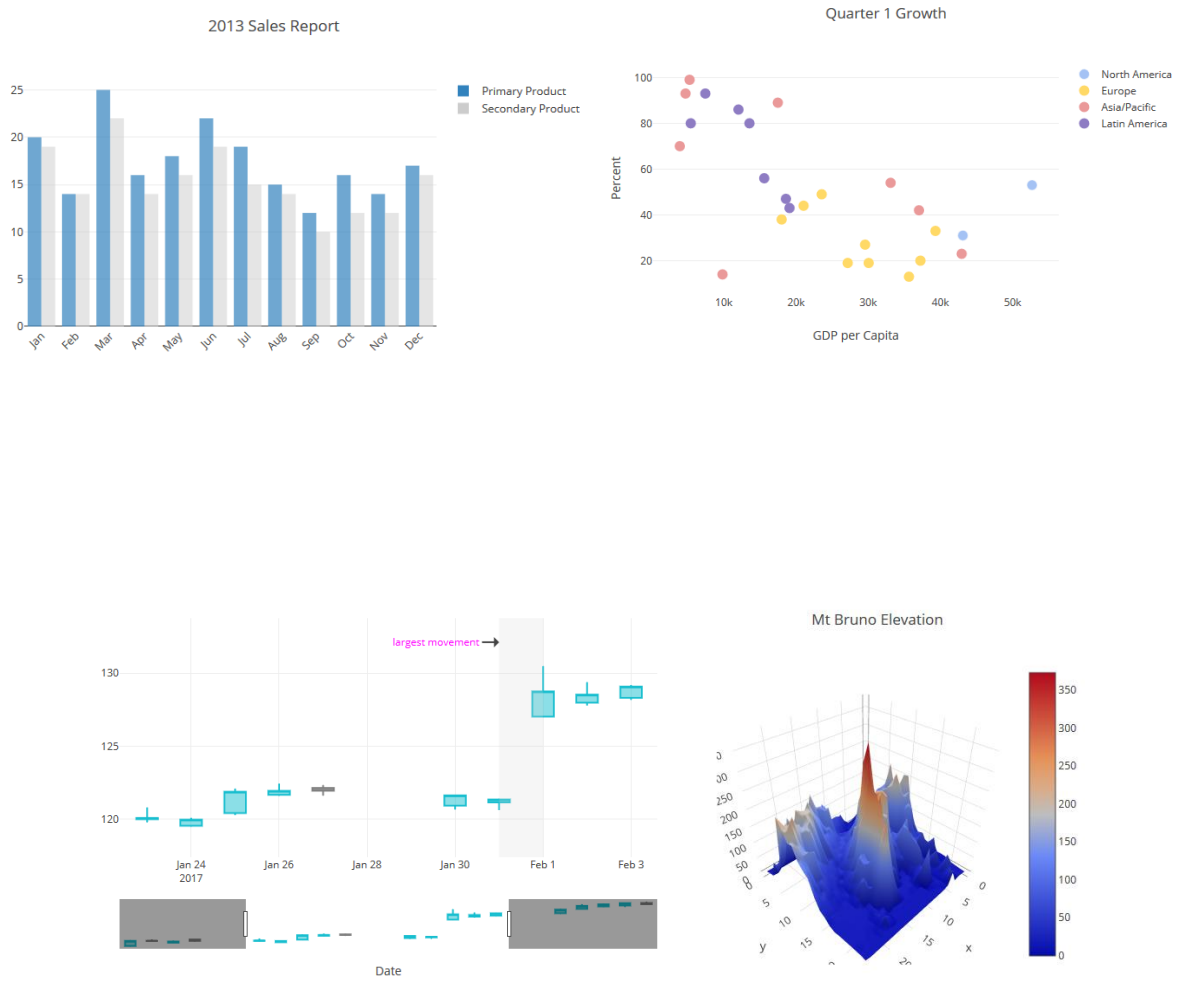

Figure A3.7. A selection of graphs implemented usingPlotly.js (Plotly 2020).

Appendix 4. Spatial data processing in R

In the prototype FDAT developed in Phase 1 of the project (FY18) the spatial data processing, modeling and prediction were undertaken in two steps using both proprietary and open-source software. Previously, the Spatial Tools for the Analysis of River Systems (STARS) tools (Peterson and Ver Hoef 2014), developed as ESRI ArcGIS (ESRI 2019) custom tool sets, coded in Python, for ArcGIS versions 9.3- 10.7[2], were used to generate the spatial, topological, and attribute information needed to fit spatial stream-network models to data collected on river networks. However, there are often changes in internal function syntax as new versions of ArcGIS are released, which means that they are rarely back compatible. In addition, ArcGIS is proprietary software and a full Advanced license is needed to run the STARS tools. To address these issues, we made the decision to convert the core STARS tools to scripts on the osSTARS package that can be executed in R statistical software (R Core Team 2020), which is free, open source software. In addition to cost savings, using open source software allows the pre-processing environment to be held relatively static within FDAT and maintained indefinitely, which significantly reduces ongoing maintenance. Finally, all of the statistical modeling and prediction takes place in R, creating a seamless data-analysis pipeline within one software product.

A4.1 osSTARS

The osSTARS package replicates the core functionality of the STARS custom toolkit needed to support the FDAT. Five new functions were created that have a 1:1 relationship with those found in STARS (Table 4.1). These functions convert the streams and observed sites to a Landscape Network (LSN) structure, which is the primary data format in STARS. There are also functions to calculate the upstream distance (upDist) from the stream network outlet to the upstream node of each line segment and to each individual observed site location. Additive function values are needed to fit spatial stream network models using the tail-up covariance function and these are generated using the additive.function function in the SSN package for R (Ver Hoef et al. 2014). Finally the pre-processed data are exported to a .ssn object, which is logically identical to .ssn objects created using the STARS tools. Thus, no data processing needs to be done to the .ssn object created by the CreateSSN_Obj function before it is imported into R using the importSSN function in the SSN package (Ver Hoef et al. 2014).

[2] https://www.fs.fed.us/rm/boise/AWAE/projects/SSN_STARS/latest_releases.html

Table 4.1. The relationship between STARS tools and equivalent functions within the osSTARS package. osSTARS functions are listed in the order that they will be called within FDAT.

STARS Tool Name	osSTARS Function Name
Pre-processing Polyline to Landscape Network Snap Points to Landscape Network	 polyline_to_network snap_pts_network
Calculate Upstream Distance – Edges Upstream Distance – Sites	 getUpstreamDist_Edges getUpstreamDist_Sites
Export Create .ssn object	 CreateSSN_Obj

The same topological conditions for streams required in the STARS tools must also be met in the osSTARS package in order to accurately generate hydrologic distances and describe spatial relationships; namely that the shapefile does not contain converging stream nodes, complex confluences, and or diverging nodes (Peterson and Ver Hoef 2014). STARS contains a number of tools that help users identify these topological conditions so that they can be removed, but these are not needed to run the FDAT since the stream network will be static and will not vary by user. Therefore, we did not include this functionality in the osSTARS package.

polyline_to_network

The polyline_to_network function converts a shapefile representing streams into a network structure that contains the same spatial information as the LSN utilized by the STARS tools (Peterson and Ver Hoef 2014). If it runs successfully, it produces a shapefile for nodes and edges, and three CSV files containing node-to-node, node-to-feature, and feature-to-feature relationships (i.e., nodexy.csv, noderelationships.csv, and relationships.csv).

snap_pts_network

The snap_pts_network function is used to incorporate observation sites into the network structure generated by the polyline_to_network function. If the sites do not lie directly on the stream network, they are moved to the nearest location on a stream segment (i.e., “snapped”) and the coordinates for the point feature are recalculated and added to the sites shapefile attribute table. Two additional columns are also added to the sites attribute table; the rid column contains the reach identifier for the line segment the site lies on, while the ratio column contains

the proportional distance of the site location from the downstream node of the line segment compared to the total length of the line (Peterson and Ver Hoef 2014). These two pieces of information are important because they describe which line segment each site resides on and where exactly on the segment the site lies.

getUpstreamDist_Edges and getUpstreamDist_Sites

The getUpstreamDist_Edges and getUpstreamDist_Sites functions are used to generate information needed to calculate hydrologic distances between sites along the stream network (Peterson and Ver Hoef 2014). The getUpstreamDist_Edges function calculates the distance from the stream outlet (i.e., most downstream location in the stream network) to the upstream node of each line segment. The values are then stored in a new column called 'upDist' in the edges shapefile attribute table. The getUpstreamDist_Sites function calculates the distance between the stream outlet to each site location based on data in the upDist column of the edges attribute table, along with information contained in the rid and ratio columns of the sites attribute table. A new upDist column is added to the sites attribute table and populated with the upstream distance values.

CreateSSN_Obj

Once the functions for pre-processing and calculating new data have run successfully (Table 4.1), the data must be converted into a format that can be imported using the SSN package (Ver Hoef et al. 2014) in R. The CreateSSN_Obj function is used to convert the LSN structure into a .ssn object as described in Peterson and Ver Hoef (2014). First, a new directory is created, with the naming convention lsn-name.ssn (i.e., lsn.ssn), which is referred to as the .ssn object. Next, a network identifier (netID) is assigned to the edges, observed sites, and prediction sites attribute tables. A location ID (locID) and a unique point identifier (pid) are assigned to the observation and prediction sites attribute tables in order to distinguish between repeated measurements at a single location. Binary identifiers (binaryID) are assigned to each line segment, which are stored along with the rid value as comma delimited text files in the .ssn object, with a separate file for each network. The naming convention for these files corresponds to the netID (i.e., net1.dat, net2.dat, etc.). Finally, the edges, sites, and prediction sites shapefiles are exported to the new .ssn object directory.