University of Western Ontario
Department of Electrical and Computer Engineering
Software Engineering Design Project (SE4450)

Forestcasting

Forest Fire Prediction Powered by Analytics

Faculty Advisor: Katarina Grolinger

Samantha Campbell
Shima Kananitodashki
Tharmiga Loganathan
Ivan Zvonkov

Abstract

Forest fire managers are responsible for strategically planning firefighting efforts to maximally prevent damages to communities and the environment. In recent years, the focus of forest fire management has shifted from suppressing forest fires after they break out, to proactive fire prevention through the use of analytics and data driven decision making. Pythia has developed Forestcasting, a forest fire management tool that provides vital information to aid firefighting resource allocation. The Forestcasing tool consists of (1) a machine learning model that calculates the probability a forest fire could occur in a given area, and (2) a mathematical damage algorithm that estimates the severity of damages caused by said fire. Forestcasting's results give forest fire managers the tools and data required to take appropriate precautions. As forest fire severity and frequency are increasing, fire management capacity is decreasing, and the expenditures of forest managers are rising due to higher forest fire suppression costs. With steadily rising costs, Canadian wildland fire management agencies are investing in new initiatives to prevent damages. Forestcasting aims to fill this gap by equipping forest fire managers with a tool to enhance decision making, consequently decreasing costs and damages.

Introduction

Background

Forest fire managers are responsible for planning firefighting efforts in case of fires in order to prevent damages to communities and the environment. Critical issues facing forest fire management include projected climate change impacts, increased public expectations, expanding economic development near forest areas, declining fire management capacity and declining experienced staff. [1]

Over the past decade, climate change effects have become more extreme and have increased the danger and scale of forest fires accordingly. As forest fire severity and frequency are increasing, fire management capacity is decreasing, and the expenditures of forest managers are rising due to higher suppression costs. With steadily rising costs, Canadian wildland fire management agencies are investing in new initiatives to prevent damages. Further, the agencies have realized that an attitude shift is necessary as current response strategies are not sufficient given the outlook of forest fire trends.[2]

Recently, the focus of forest fire management has shifted from suppressing forest fires after they break out to fire prevention through the use of analytics and data driven decision making.[3] Forest managers can prepare to respond to forest fires if they have a better understanding of the most vulnerable forest areas. This knowledge would enable them to allocate

investments in prevention and mitigation efforts for the areas at most risk, therefore reducing the potential damage to communities and critical infrastructure.

There is a sufficient amount of historical fire data and relevant factors affecting fire behaviour, such as weather and land composition. A number of government initiatives have made investments in technologies and research studies that leverage the available data to draw insights and direct management's decisions. However, the existing tools are focused on providing a single prediction regarding a single aspect of a forest fire. There is no focus on equipping forest fire managers with a comprehensive view of forest fire prediction and respective damages. Moreover, a critical element is missing from the existing solutions which is a measure of possible damages. As fire seasons become increasingly damaging, fire management costs to the government and the environment are steadily increasing.

Objectives

No existing forest fire management tool provides a comprehensive view of both risks and damages. Having identified this gap, *Forestcasting* aims to provide managers with a comprehensive tool that will help them predict fire risks and measure fire damages to be able to develop an appropriate response strategy. The solution focuses on equipping managers by building a comprehensive decision-making tool used to prioritize forest fire fighting efforts.

Constraints

The first constraint of this project was the availability of data for the prediction model and the damage algorithm. The data required includes historic forest fire data, current and historic weather data, and land features data. Having access to open source data with high quality was crucial to the project. This limitation meant accessing the datasets that the government agencies have made available online.

Furthermore, as the focus of this project was machine learning, computing power was another major constraint. The computation constraint was applicable when training and developing the machine learning models and performing large data operations. The models for Forestcasting were developed and tested using *Google Collaboratory* notebooks and therefore were limited by the provided processing capabilities. The team prioritized algorithm optimization to address the computing constraint and worked with *Google Collaboratory* when completing computationally heavy tasks.

Moreover, the software used to train the machine learning models and develop the application for *Forestcasting* had to be open source. Due to lack of budget for purchasing enterprise software, *Forestcasting* only used open-source software for testing and development and relied on the open-source community. This constraint was considered when picking technologies, languages, frameworks, and services such as APIs and data sources.

Finally, time was the greatest constraint when developing Forestcasting. With only seven months to design, develop, and test the application the team had to be strict with scope and prioritize core functionalities. The constraint was managed through flexible schedules to dynamically address unforeseen situations and revisit the importance of minor functionalities. The timing of the project was evaluated regularly, and priorities were continually updated.

Scope

The core goal of *Forestcasting* has been to provide an accurate prediction of the likelihood of a forest fire occurring and calculation of the potential damages in a region. Forest fire managers will use such information displayed in the application to make informed decisions about prioritizing certain regions and better allocate their resources. This is an extension of the existing tools in the market as we use analytics to predict likelihood of forest fires and provide unique information and insights by also calculating damage levels.

The product can be described as five subsystems: data ingest, machine learning model and damage algorithm, backend server, and user interface. The product's value is directly related to the prediction and calculation accuracy. A highly accurate model and damage algorithm are *Forestcasting's* main features, therefore they were prioritized. Increased user interface features and API functionalities are areas of growth in future releases.

Software Requirements Specifications (SRS) Document

Table of Contents

1 Introduction

Forestcasting is a machine learning application that given an area and date(s) will evaluate the area's:

(1) forest fire occurrence probability and;

(2) potential damage given a fire occurs.

Forest fire managers and government agencies will use the *Forestcasting* results to better assess areas at risk and optimally allocate their resources. This section will describe the scope of *Forestcasting* and provide information required for understanding the document's remaining sections.

1.1 Document Purpose

This document covers the software requirements of *Forestcasting* version 1.0. The application uses a machine learning model to score an area's forest fire occurrence probability, and an algorithm to calculate the level of damage a fire would cause in that area. The product can be divided into four main sections: data ingest, machine learning model and damage algorithm, application program interface endpoints, and user interface. For each product section the document outlines the requirements, constraints, and specifications with the goal of providing the reader with a complete understanding of *Forescasting's* requirements. Additionally, available technologies and solutions will be compared and decisions will be justified.

Considering *Forestcasting* as a "black-box" application, the product's safety, security, environment, and performance requirements are defined. Additionally, *Forestcasting's* target client is considered in this document, and their requirements and expectations of the application are listed. Any requirements not outlined in this document are still under review. The document's goal purpose is to provide a high-level, holistic understanding of *Forestcasting* 1.0.

1.2 Product Scope

The product can be described as four subsystems: data ingest, ML model and damage algorithm, API, and UI. The goal of *Forestcasting* is to provide a ML model that predicts the likelihood of a forest fire occurring and an algorithm to calculate potential damages given a region and date(s). Although *Forescasting* could be expanded to cover additional countries, the current implementation will only support Canadian locations. Forest fire managers will use information displayed in the UI to make informed decisions about prioritizing certain regions and better allocate their resources. Current products use analytics to predict likelihood of forest fires, *Forestcasting* provides unique information by also evaluating damage levels.

The product's value is directly related to the ML model's accuracy. Highly accurate models are *Forestcasting's* main feature, therefore the user interface and API aspects of the product will be minimal. Increased UI and API functionalities would be areas of growth in future releases. The data quality is directly related to a model's accuracy. Therefore, exploratory data

analysis, data cleaning, and data ingest are key aspects to creating the *Forestcasting* application. The data requirements and specifications are vital to *Forestcasting's* goals. With detailed requirements and specifications placed on the data and ML models, and core functionality from the UI and API, *Forestcasting* will provide the greatest benefit to its users.

1.3 Intended Audience and Document Overview

The intended audience for this report is the Canadian Forest Service, a subdivision of Natural Resources Canada, software testers, and software developers.

Reader 1: Canadian Forest Service

The most pertinent sections of the document to the Canadian Forest Service are *1 Introduction* and *2 Overall Descriptions*. These sections will provide a summary of *Forestcasting's* functionalities and benefits. Section three and four of the documents provide detailed descriptions of the product's software requirements and are intended for those with backgrounds in ML and software engineering.

Reader 2: Software Testers

The software testers should have an understanding of *Forestcasting's* functionalities and applications. The most pertinent sections of the document are sections *3 Specific Requirements* and section *4 Other Non-Functional Requirements*. These two sections will be crucial tools for developing complete test plans and test suites.

Reader 3: Software Developer

The software developers should have an understanding of *Forestcasting's* functionalities and applications. The most pertinent sections of the document are sections *3 Specific Requirements* and section *4 Other Non-Functional Requirements*. The software developers will use these sections as tools to validate the completed application meets all outlined requirements. Additionally, the developers can reference the sections during development to verify their implementation conforms to the designed functionalities.

1.4 Document Conventions

Formatting Conventions

This document follows standard IEEE formatting requirements. It uses the *Times New Roman* font, size 12 for all written texts. Minor section titles are bolded (i.e. see 'Formatting Conventions' subheading above) but are not present in the document's table of contents.

Naming Conventions

Company, team, and product names will be italicized. All major section names can be found in the table of contents, minor section names will be bolded but won't be in the table of contents (see above). Any other names (i.e. people, places) will not have special formatting and follow basic grammar capitalization criteria.

When referencing a section in the document, the section number and title will be used and be italicized. For example, *2.1 Product Perspective*.

Tables, charts, and diagrams will be named as: <section #>.<lower_letter>. For example, the architecture diagram in section *2.2 Product Functionality* is named *Figure 2.2.a*.

2 Overall Description

2.1 Product Perspective

Forestcasting is a standalone product that will add to Forest Fire Manager's current prediction techniques. Forest Fire Managers use various weather driven techniques to predict areas at risk of forest fires. However, these techniques do not include damage assessments of a potential fire. *Forestcasting* addresses this gap by providing a fire occurrence probability and damage score that can be used to prioritize relief resources. For example, a forest fire that occurs near a city could require evacuation protocols, whereas a forest fire in an unpopulated region could burn more freely. Various environmental factors will be considered to give a holistic damage score.

From a user perspective, *Forestcasting* will be a web application that allows region (*Figure 2.1.a*) and date selection (*Figure 2.1.a*) and displays the results in various forms (*Figure 2.1.c*). Forest fire managers will select areas of concern based on their current prediction technique and receive actionable results from the *Forestcasting* UI.

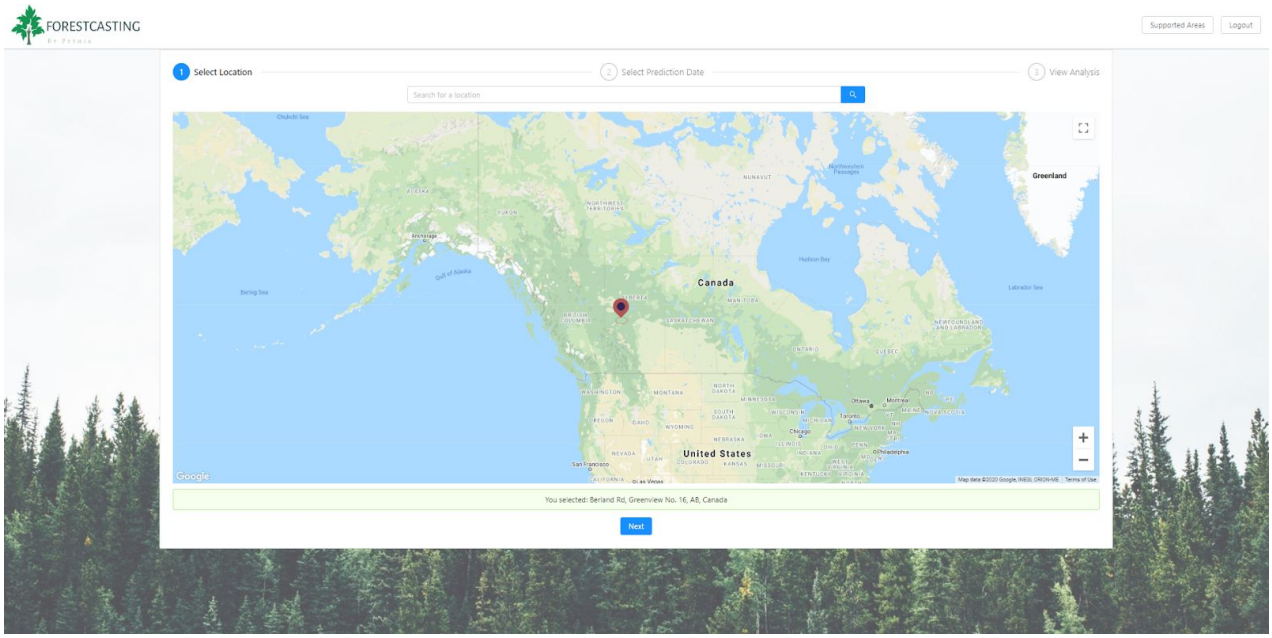


Figure 2.1.a

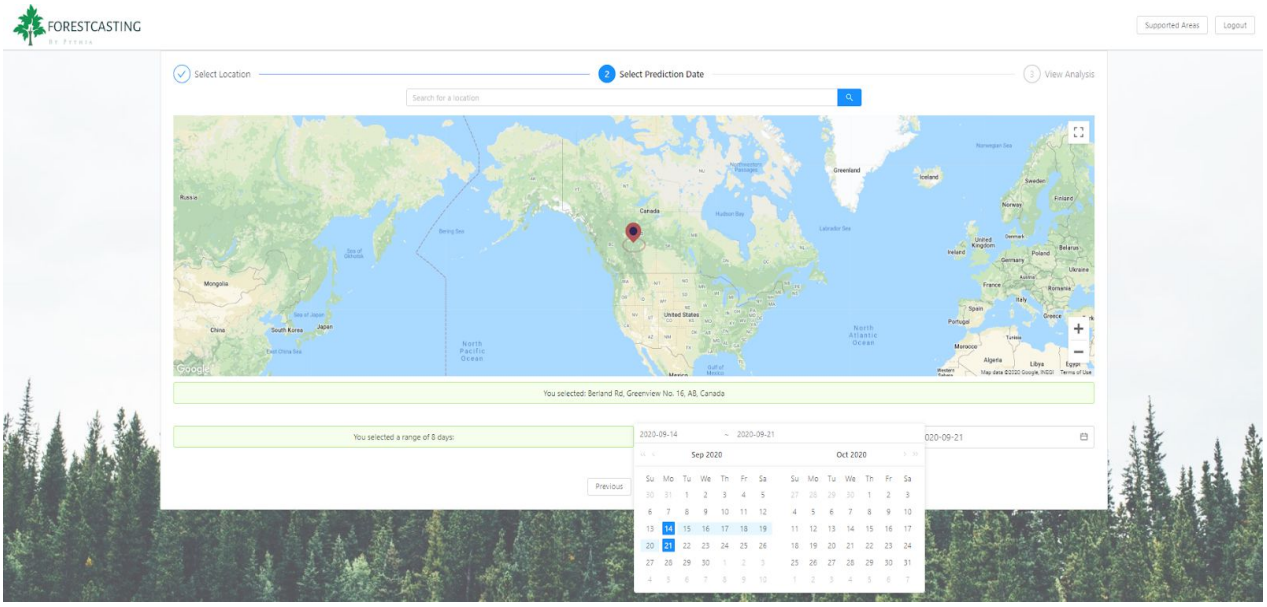


Figure 2.1.b

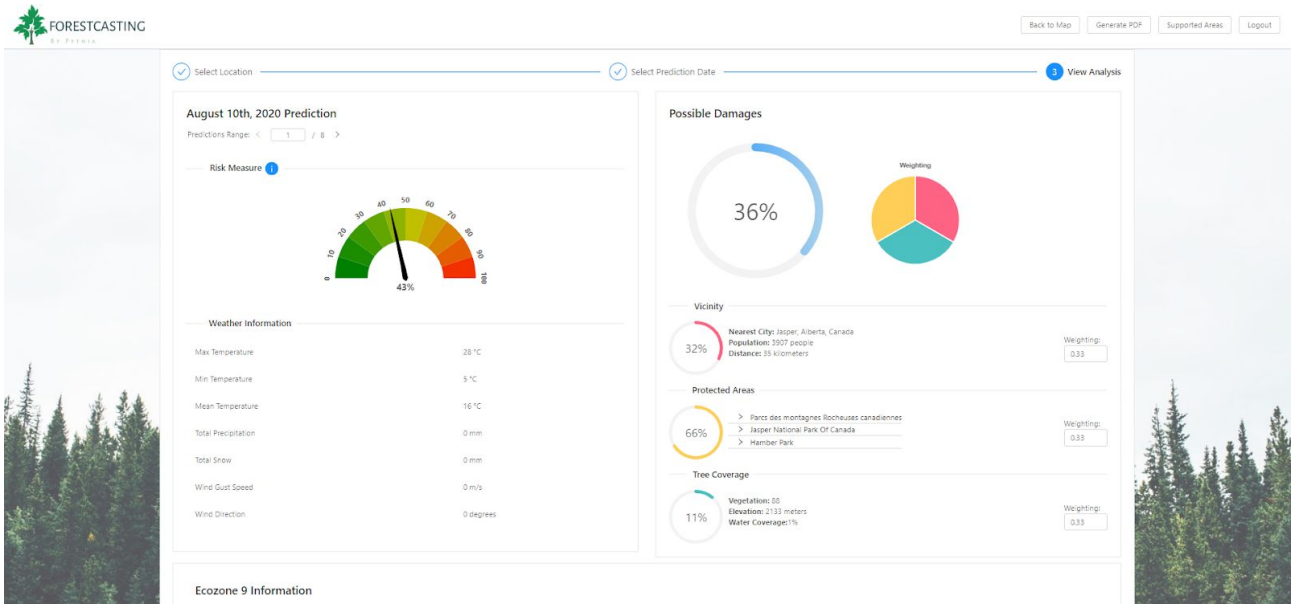


Figure 2.1.c

Once the user has selected a location and range of dates through the provided map and calendar, the region will be analyzed by the ML model and the damage algorithm, the results will then be returned to the UI. During this process the *Dark Sky Weather* API, *Wolfram Alpha* API and the database will be called to gather additional information on the selected region. *Figure 2.1.d shows* the dataflow process which is the core functionality of *Forestcasting* from the client's perspective.

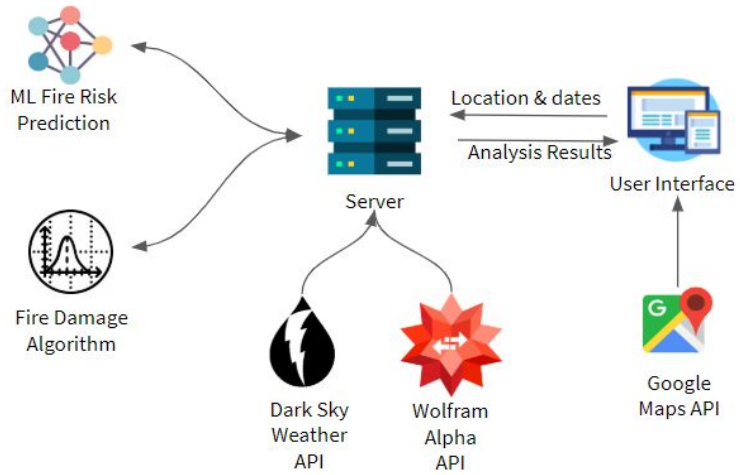


Figure 2.1.d

2.2 Product Functionality

Data Ingest Functions and Final Outputs

1. System will ingest 3 main data types:
    a. Historic Forest Fire Data
    b. Weather Data
    c. Environmental Data
2. The input to the ingest pipeline will be multiple data type CSVs
3. Each processing notebook will input and output a various number of CSVs that will be used for future process and/or uploaded to the application database for production use
4. The final output of the pipeline will be a trained machine learning model and calculated damage scores per location

ML Model and Algorithm Functions

1. Model 1 will predict the likelihood of a fire occurring in a given region on a range of given days
2. The algorithm will calculate the level of damages in a given region caused by a potential forest fire using three metrics: vicinity, tree coverage, and protected areas

Server Communication Functions

1. Communicate with the *Dark Sky Weather* API to get current weather data
2. Communicate with the *Wolfram Alpha* API to find the (1) population of the closest city and (2) population of said city given a selected location
3. Communication with cloud hosted database to gather additional information on for ML model and damage algorithm
4. Transport the user selected location to the ML model that is hosted on *AWS Lambda*
5. Transport the user selected location to the damage algorithm
6. Transport the ML models' results to the UI
7. Transport the damage algorithms results to the UI
8. Pass weather data, ecozone data, and historic statistics to the UI

UI Functions

1. Account login, and logout
2. Area selection via pin drop and search

3. Communicate with the *Google Maps* API for region selection
4. Date selection for any range of dates less than 1 year in the future
5. Display ML model's results with various graphs
6. Display the damage algorithm results with tuning capabilities for each metric used in algorithm
7. Display historic statistics and ecozone data for a selected region

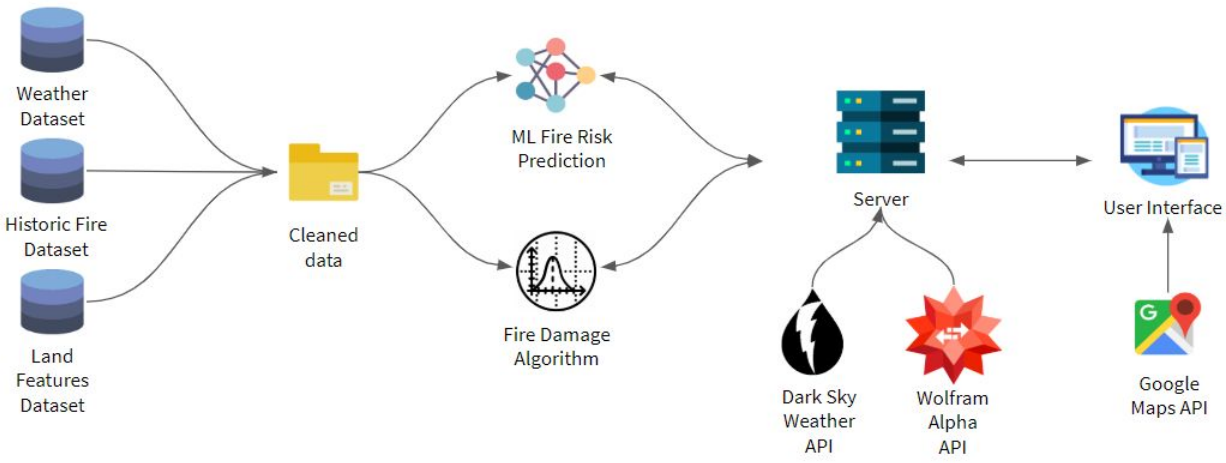


Figure 2.2.a

2.3 Users and Characteristics

User 1: Forest Fire Managers

The Forest Fire Manager will have access to the system's UI. They will be able to login, select regions to analyze, and view results. The Forest Fire Manager will not require an understanding of the models powering *Forestcasting* or any machine learning principles. They are expected to have an understanding of Canadian regions, forest fire causes, and forest fire management techniques to properly apply the information supplied by the UI.

2.4 Operating Environment

ML Models' Environment

The models will be created and trained on Google's *Collaboratory Notebooks* using *Sci-kit learn* with *Python 3.6*. *Keras* was initially considered for the ML model training, however the models available through *Sci-kit learn* were sufficient and lighter. Additionally, *Keras*' complexity is more ideal for neural networks whereas *Forestcasting*'s predictions require a basic ML model.

Collaboratory Notebooks gives access to GPUs that can be used to optimize ML tasks, whereas the team's local machines don't have access to GPU computation. Once trained, the models will be exported to JSON and H5 files and loaded onto *AWS Lambda*. The backend server will call the lambda function, similar to an API call, when analysis is requested.

Server Environment

The server will run in a *Node.js* environment that is hosted on a *AWS Linux* environment. The backend server will communicate with various APIs, a database hosted on *MongoDB Atlas,* the frontend UI, and the ML *AWS Lambda* function.

UI Environment

The UI is a web application and will be developed using *REACT*. The frontend server will communicate with the *Google Maps API* and use *Ant Design* for UI components. The server will be hosted on an *AWS Linux* environment and connect via proxy to the backend server.

Figure 2.4.a depicts the various environments required by *Forestcasting* and the corresponding technologies required.

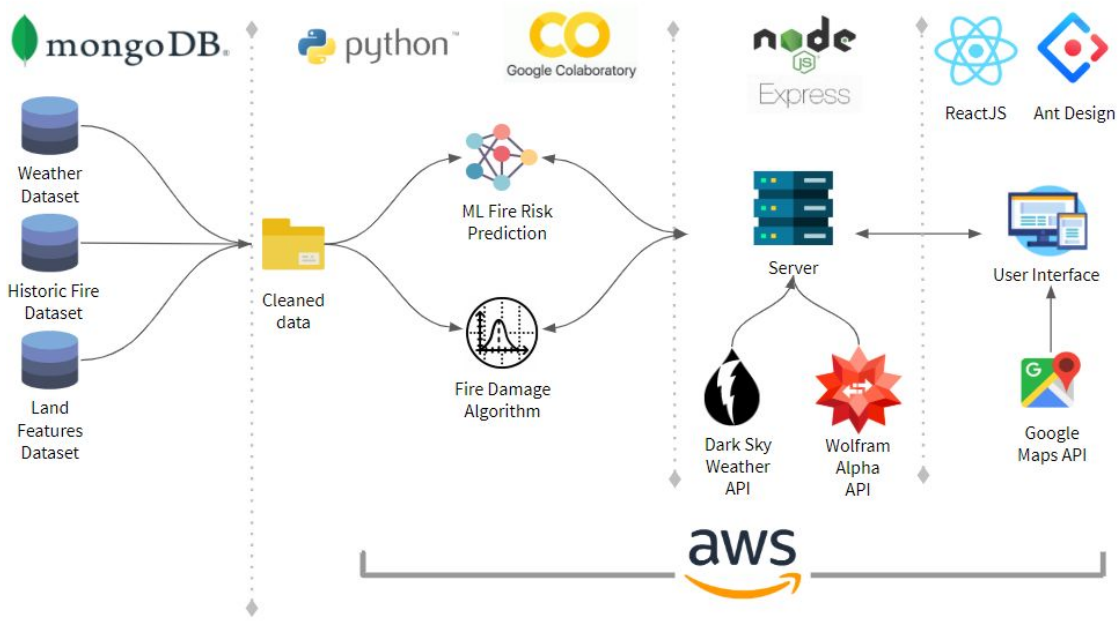


Figure 2.4.a

2.5 Design and Implementation Constraints

Data Availability

The first constraint *Forestcasting* must consider is data availability. The data types required are not sensitive and therefore anonymization won't be an issue. However, finding enough open-source data of high quality will be crucial. Without partnering with government agencies, the datasets must be available online, downloadable, and in a workable format. This constraint will be the main deciding factor for which area the application is applied. To apply the *Forestcasting* application to an area there must be historic forest fire data, current and historic weather data, and land features data available. This constraint will be addressed during the exploratory data analysis phase of the project.

Computing Power

The models for *Forestcasting* will be developed and tested using *Google Collaboratory* and will be limited by the provided CPU and GPU capabilities. The algorithms used for the ML models must function within the CPU and GPU limitations. Initially, testing and development was to be completed using the team's local machines, however *Google Collaboratory* provides higher computing power. Big data projects often take advantage of distributed systems and multi-node computing, however *Forestcasting* will be limited to single-node computations. The computation constraint will be applicable when training and developing the ML models and performing large data operations. The team will prioritize algorithm optimization to address the

computing constraint, and work with *Google Collaboratory* when completing computationally heavy tasks.

Open-Source

Forestcasting will only use open-source software for testing and development. The team does not have a budget for purchasing enterprise software and will rely on the open-source community. The constraint will be considered when picking technologies, languages, frameworks, and services. The open-source community around big data and ML software is quite robust, therefore this constraint is not a major concern.

Time

Time will be the greatest constraint when developing *Forestcasting*. With only seven months to design, develop, and test the application the team will have to be strict with scope and prioritize core functionalities. The constraint will be managed through flexible schedules to dynamically address unforeseen situations and revisit the importance of minor functionalities. The timing of the project will be revaluated regularly, and priorities will continually update.

2.6 User Documentation

Forestcasting will be a SaaS application (software as a service) where the UI is a web application and the backend and API are hosted on a server on *AWS*. Therefore, the client will not need an installation or upgrade guide. Upgrade documentation will be required for internal use but won't be customer facing. Customer facing tutorials on using the web application will be available via the website. Additionally, the website will provide a method for contacting the *Pythia* team to address questions and concerns. One of the benefits of SaaS applications is managing installs and upgrades internally therefore not requiring customer facing solutions.

2.7 Assumptions and Dependencies

Dependencies

Forestcasting only depends on widely accepted open-source projects that will not pose additional constraints on the application.

Assumptions

1. At any given time, there will be no more than 5 regions in a queue to be analyzed
2. Regions beyond Canada will not be processed
3. Current weather will be available for all regions selected

4. The models will not be retrained more than once a year
5. The API will process less than 1000 calls per day

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface should be compatible with all major browsers. It will be implemented using *React*. The application will have three main pages. The first page is a login screen. The second page is an interface where users can select a region and date(s) to analyze. Once the user selects a region to analyze, and a range of dates, the third webpage will be shown with the results of the analysis conducted. This third page will have a description of the analyses derived from the fire risk machine learning model and the damage algorithm along with graphed information representing historical data characteristics of the region selected.

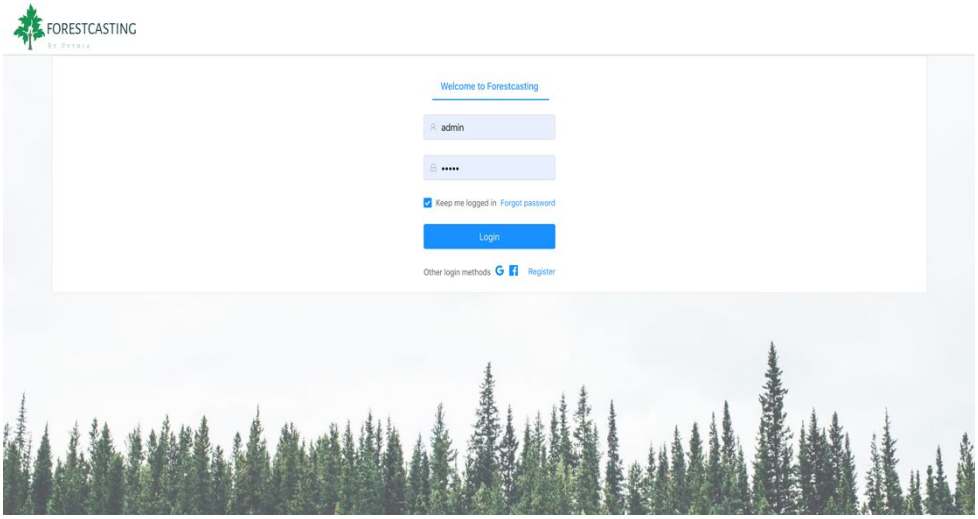


Figure 3.1.1.a - Login

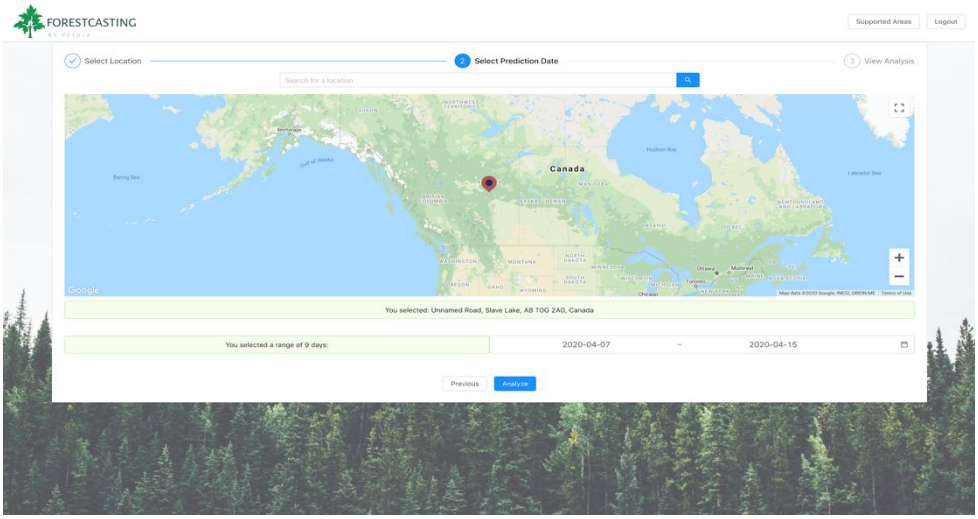


Figure 3.1.1.b - Search for Region - A user can select a location and the range of dates they would like to analyze

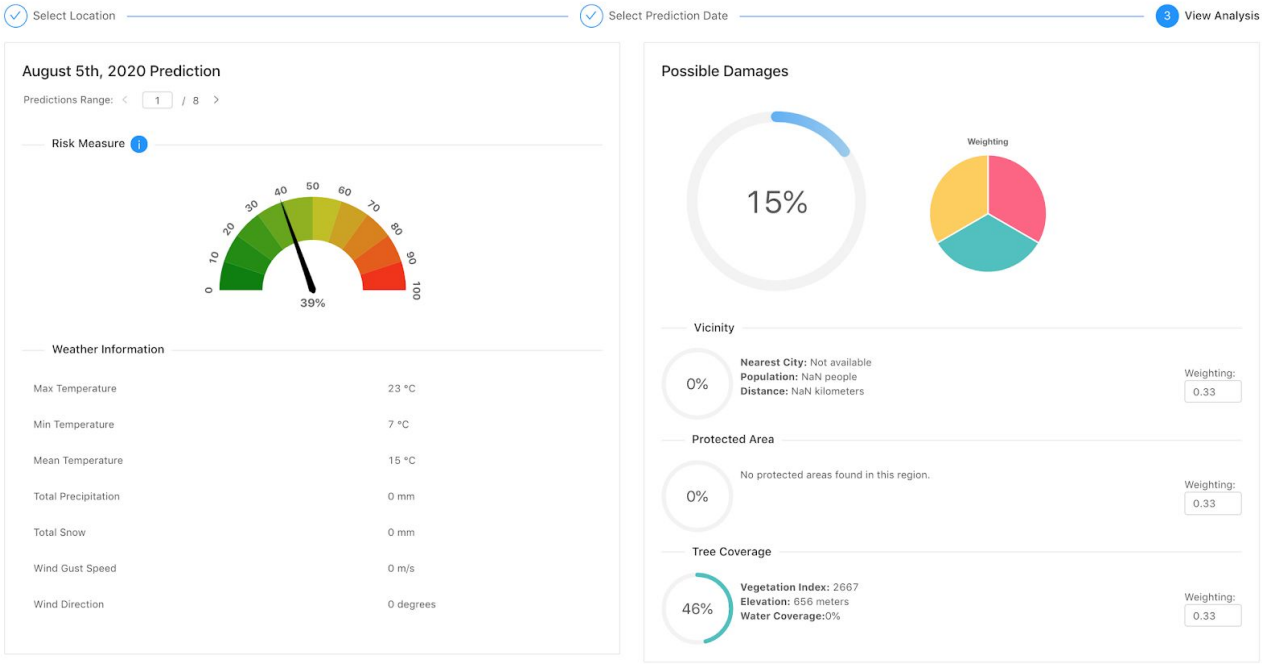


Figure 3.1.1.c - Analysis Results Part I - First half of the analysis webpage, shows fire risk, possible damages and related information.

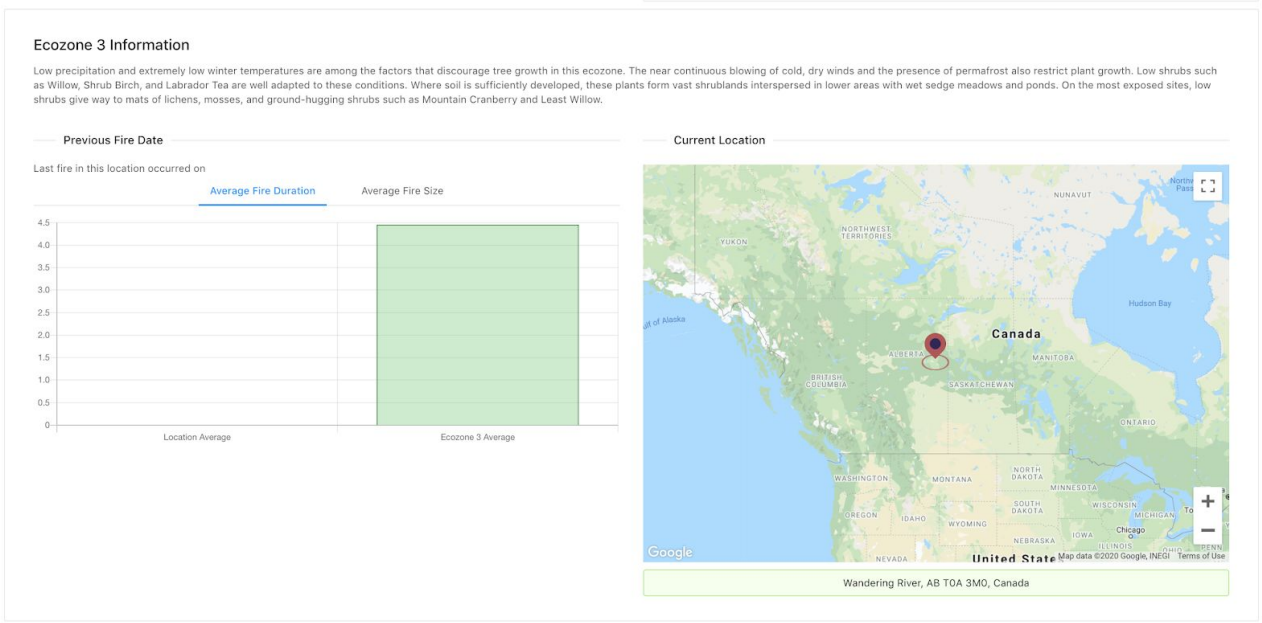


Figure 3.1.1.d - Analysis Results Part II - Second half of the analysis webpage, shows historic fire information for that location along with a map view of the selected location.

3.1.2 Hardware Interfaces

The only hardware we will require is a server to host our website on. We are using *Amazon AWS* to serve the website. We will not be using any special libraries to communicate with the software.

3.1.3 Software Interfaces

The website is hosted using an *Amazon AWS S3* container. The backend is hosted using *Amazon AWS Lambda*. The backend serves information to the frontend based on frontend queries to the backend for specific analysis. The ML model and damage algorithm is also hosted on the backend after it was trained offline. The application database is non-relational and hosted on the cloud using *MongoDB Atlas*. The frontend was created using *React*.

3.1.4 Communications Interfaces

The main communication will take place over HTTP using a web browser of the client's choice. The data will not be specially encrypted as it does not contain sensitive information.

3.2 Functional Requirements

Data Ingest Functions

System will ingest 3 data types:

a. Historic Forest Fire Data
   - Attributes: {FIRE, SIZE_HA, ECOREGION, ECOZONE, START_DATE, OUT_DATE}
   - Historic forest fire activity data is organized into predetermined regions and is fed into the models

b. Weather Data
   - Attributes: {MAX_TEMP, MIN_TEMP, MEAN_TEMP, TOTAL_PERCIP, DIR_OF_MAX_GUST, SPD_OF_MAX_GUST}
   - We combine region-specific weather data with region-specific historic forest fire data to determine the effect that weather has on forest fire activity
   - We then feed current weather data into the models to use as a factor in predicting forest fire risk today

c. Environmental Data
   - Environmental information acts as another parameter to determine forest fire risk

- Regions in the same environmental classification have similar characteristics that contribute to forest fire behaviour

The input to the ingest pipeline will be multiple data type CSV's and the output will be a trained machine learning model, calculated damage score metrics, and various CSVs that will be uploaded *Forestcasting's* cloud database to be used in production

ML Model's Functions

The machine learning model predicts the likelihood of a fire occurring in a given region on a given range of days. To do this, the model is specifically fed the following parameters during training:

1. Historic forest fire statistics (total duration, average duration, total size, average size)
2. Ecoregion information (ecoregion that belongs to each region)
3. Location (latitude and longitude)
4. Weather (temperature, wind speeds, humidity, precipitation)

Damage Algorithm

The damage algorithm predicts the level of damage in a given region that a potential forest fire can cause. To do this, the algorithm is specifically fed the following parameters:

1. Vicinity information (closest cities and corresponding population)
    a. Source: Wolfram Alpha API
2. Protected area information (regions that are a protected area)
    a. Source: Shapefile Database from Government of Canada Website
3. Tree coverage (percent tree cover)
    a. Source: Google Earth Engine

Frontend Functions

1. Communicate with the *Google Maps* API for region selection
    a. The map UI on the website is served using the Google Maps API
2. Communicate with the backend API to get results of ML model, damage algorithm and other characteristic information

Backend Functions

1. Communicate with the *DarkSky Weather* API to get current weather data
    a. DarkSky API is used to retrieve current weather data
2. Query database for damage algorithm results, weather averages, and information specific to the requested analysis

    a. Selected region will be queried for supplementary information that is presented on the analysis page

3. Query *AWS Lambda* for ML model results

UI Functions

The user will provide the system with the location and range of dates they would like to analyse, that location and range will then be passed through the ML model and damage algorithm to determine fire risk, and potential fire damage.

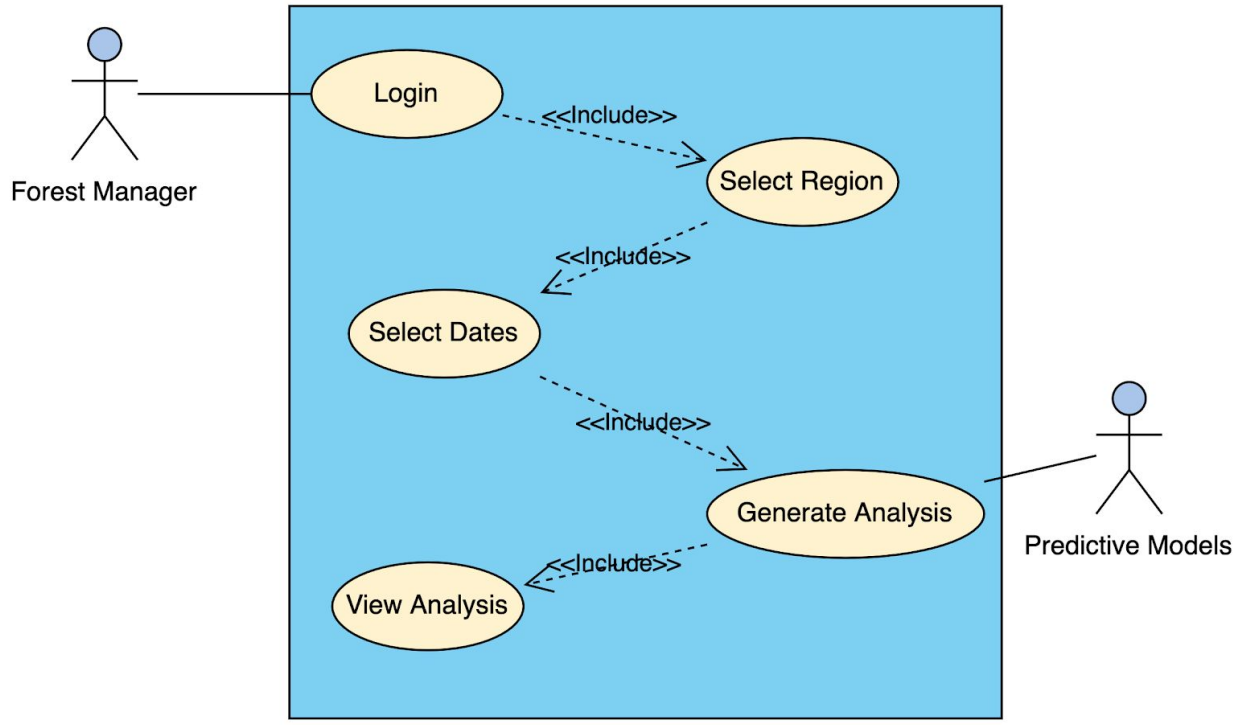
User Inputs: Location, Data Range to Analyze

System Outputs: (1) Fire Risk, and (2) Fire Damage

1. Account login, and logout
2. Area selection via pin drop or search
    a. Users will be able to use the embedded Google Maps UI on the website to select a location of their choice to be analyzed by our ML models
3. Date selection using a calendar feature
    a. Users will be able to use an embedded calendar on the website to select a range of data to be analyzed.
4. Display ML model and damage algorithm results with various graphs and supporting information
    a. The results that are outputted by the ML models will be visualized using Ant Design Charts
5. Damage algorithm will be tunable to reflect the user's priorities
    a. The damage score is calculated using a weighted sum with default weights of 0.33. The user will be able to dynamically alter these weights and have the damage score update to automatically
6. Display statistics for the selected region
    a. Statistics are computed by looking at the historic fire duration and fire size in that location
    b. Weather data in that region during the selected day range
    c. Additional information relating to protected area, land characteristics, and vicinity to large populations

3.3 Behaviour Requirements

3.3.1 Use Case View



4 Other Non-functional Requirements

4.1 Performance Requirements

1. Website will not take more than 3 seconds to load, assuming the client has a stable Internet connection
2. Communication with the *Google Maps API* will be established within 3 seconds of the webpage loading, assuming the client has a stable computer
3. Website will be optimized so that the client's computer will not require a significant amount of RAM (<100MB) or CPU usage (<20% for a typical CPU/GPU) to navigate the website
4. Once a region or location has been selected for analysis, the ML model and damage algorithm will take no longer than 10 seconds to provide the output in the backend

5. Once the backend has received the output of the ML model and damage algorithm, it will take no longer than 10 seconds to perform the necessary computations for displaying the analyzed results to the user

4.2 Safety and Security Requirements

1. There will be a login system in place for users to login.
2. We will also reduce the risk of any SQL injection attacks by sanitizing any request that is made to the server.
3. The website will be properly TLS/SSL certified which will ensure that secure/private communication between the web server and the client is provided

4.3 Software Quality Attributes

4.3.1 Accuracy

The product should be able to provide an accurate prediction of forest fire risk and potential forest fire data such as cause and property damage. This has been accomplished by completing thorough research on the best type of ML model for this task. The amount of time spent on data preparation and ML research was crucial to developing an accurate model.

4.3.2 Operability

The system is optimized for intuitive navigation. As shown by the screenshots included above, every page will be self-explanatory. This was accomplished by having user testing feedback to help and improve the operability of the website.

4.3.3 Time Behaviour

The website should be efficient as per section *4.1 Performance Requirements*. This was accomplished by not hindering the performance of the website with unnecessary resource intensive graphics. Optimized code and memory management standards were implemented to ensure that the website maintains fast performance.

Moreover, the ML model and damage algorithm must be able to make predictions within a timely manner. To do this, we ensured that the most time-consuming aspect of the models, i.e., the training of the models, does not take place on the website. The model and algorithm were saved and used for predictions on the website after they had been trained locally. This made the model's performance and hence the overall website's performance much more efficient.

4.3.4 Crash Frequency

This frequency will be less than 5 crashes per year. To do this, we implemented stress testing and debugging on the models and on the website. For example, if the website is undergoing a lot of traffic, we will ensure that server capacity is increased in order to accommodate an increased number of clients.

Software Design Specification (SDS) Document

Table of Contents

1. Introduction

1.1 Purpose of this Document

The main objective of the Software Design Specification is to provide a comprehensive description of the software product from a technical perspective. The Software Requirements Specification will be used to construct an accurate and clear representation of the structure of the project. The scope and common definitions will be outlined to put the architecture in the context of the bigger picture.

1.2 Scope of the Development Project

This Software Design Document is for a Forest Fire Prediction application to be used by forest managers across Canada. Base level functionality of the application will aid forest managers in predicting and prioritizing forest fires in relevant regions. The focus of the application is the development of a Machine Learning model to provide accurate predictions of fire occurrence and the calculation of potential damages.

1.3 Overview of Document

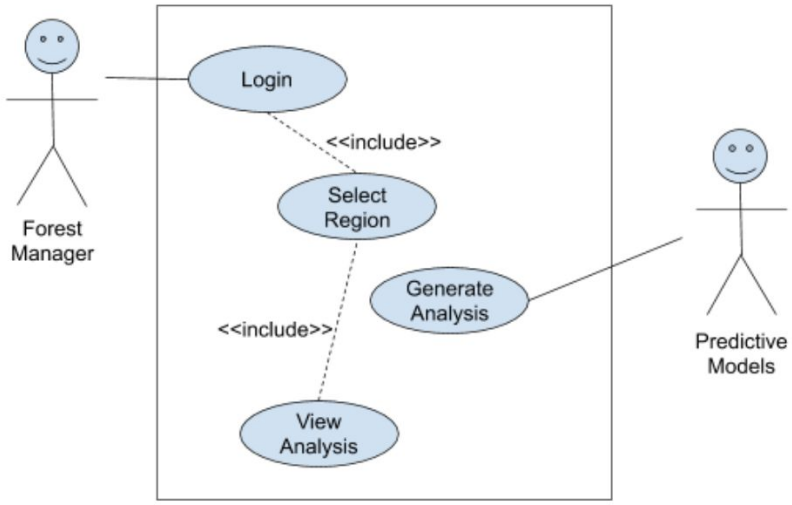
The Software Design Specification is broken down into several distinct parts. The Logical Architecture part presents a view of the overall architecture and top-level communication between various components. The next section provides a detailed description of components and their purpose and functionality. The Design Rationale will provide arguments for selecting the chosen design against other options by outlining the advantages and disadvantages of the selected design structure.

2. Logical Architecture

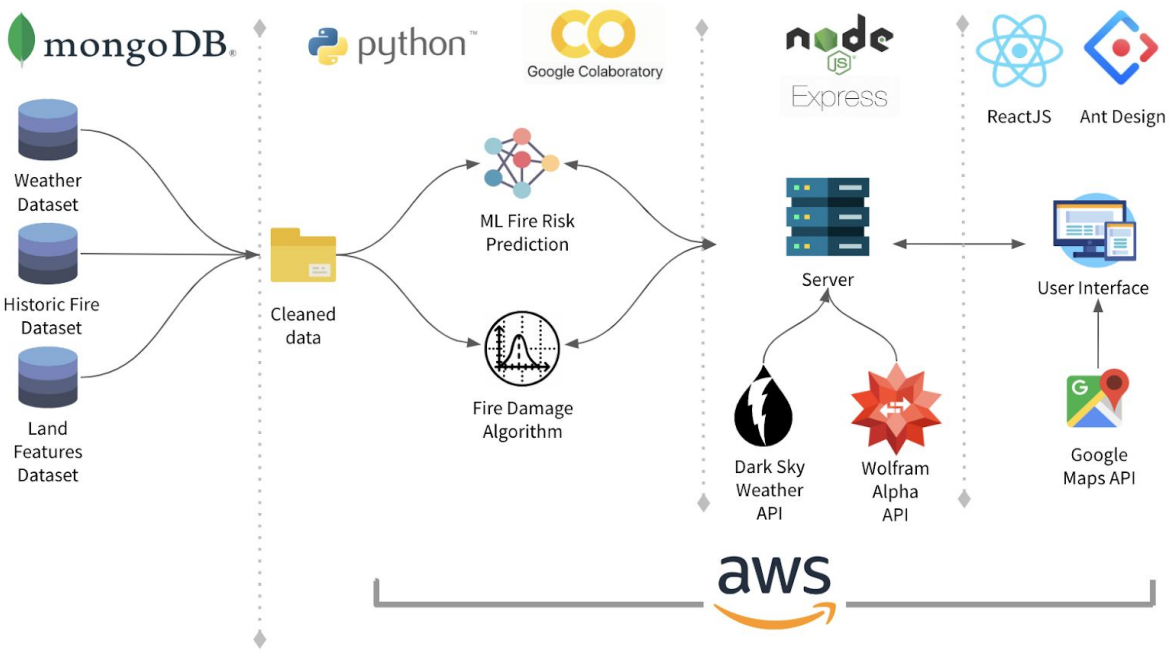
2.1 Overview

From a user and user interface perspective the system is fairly straightforward. On loading the application, the user can login, select a region and prediction date(s) for analysis and explore the analysis report once it is generated. The complexity of the application comes from the predictive model and algorithm behind the user interface which generates the analysis report.

2.1a Use Case Diagram



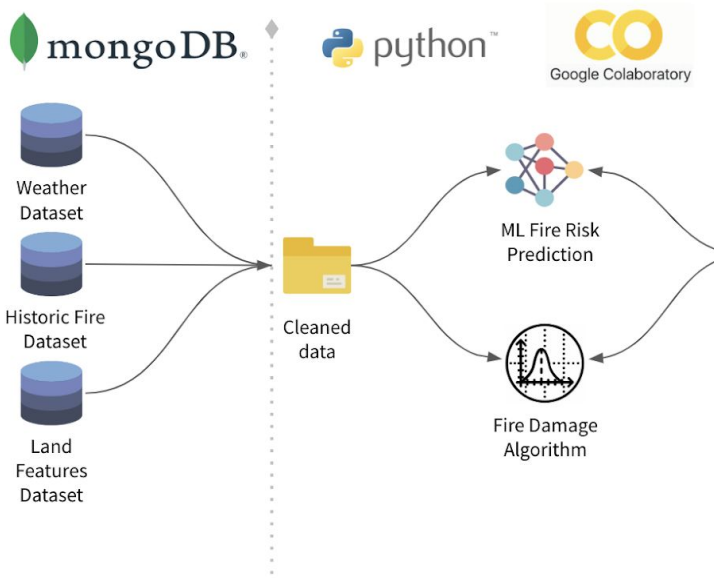
2.1b Application Architecture Diagram



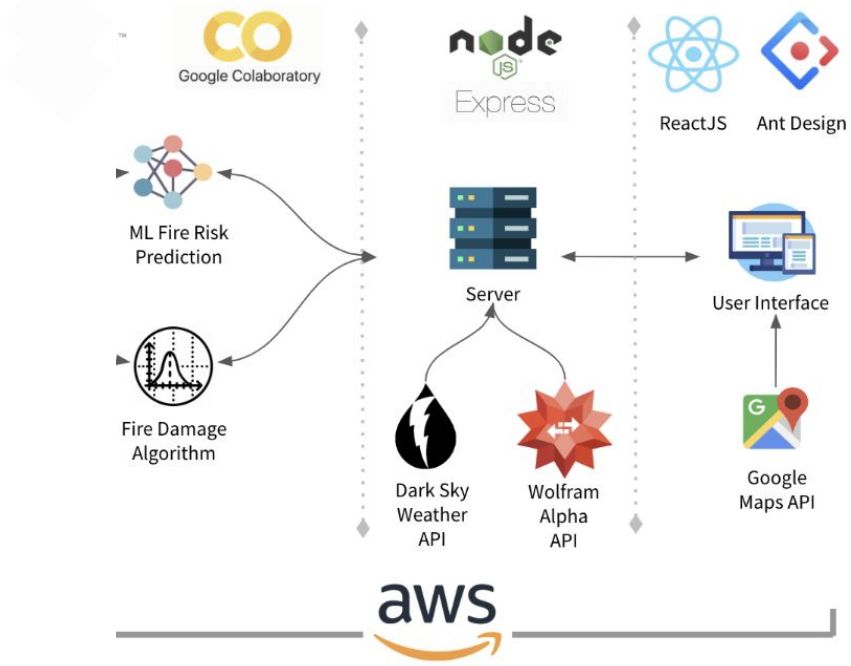
The Application Architecture Diagram describes all the high-level components involved in the application. It can be split into training and production portions for a clearer picture of how components contribute to the functionality of the application. The data pipeline and training portion includes the datasets, a fire risk machine learning model and a damage algorithm which are used to generate the analysis reports. The production portion includes the server and the UI

which represent the web application. The specifics of the architecture will be described in the following section.

2.1c Application Architecture Diagram (Data Pipeline and Training Portion)



2.1d Application Architecture Diagram (Production Portion)



The data pipeline and training and production portions of the application can be thought of as separate packages albeit with some limitations.

2.2 Data Pipeline and Training Package

The data pipeline training portion of the application architecture [2.1c Application Architecture Diagram (Training Portion)] is naturally more of a process than a package. It is done as part of the development process and therefore a package diagram does not make sense to demonstrate the functionality. The steps of the fire risk prediction machine learning process (common to many Machine Learning projects) can be split into a data pipeline (illustrated in diagram 2.2a) and the training portion.

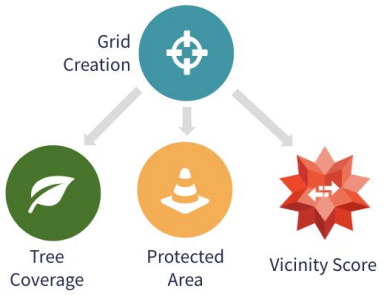
2.2a Fire Prediction Data Pipeline

Follow the pipeline the steps are:

1. Wrangling the unified data table to make all data Machine Learning ready.
2. Splitting data into testing and training sets
3. Using the above sets as inputs to train and test the accuracy of a Machine Learning model.
4. Tuning parameters of the model to achieve the best results through cross validation.

The damage algorithm has a slightly modified process. The data pipeline is illustrated in 2.2b.

2.2b Fire Prediction Data Pipeline



The damage algorithm is then developed by wrangling the available data and combining it to output a metric for each combination of data points.

Both of these processes can be directly mapped to the application architecture portion shown in (2.1c) and are run on Google Collaboratory using Python. Google Collaboratory provides access to GPUs (Graphical Processing Units) and TPUs (Tensor Processing Units) which greatly speed up data processing. Further, many useful libraries are available through Python such as Pandas Data Analysis library and scikit-learn. These processes do not run in production. After all processing is complete, the fire prediction model is outputted as a joblib.z file which is then made into a production API and the damage algorithm calculation is outputted directly to production by encoding it in the backend code.

2.3 Production Package

2.3.1 Package Documentation:

Name: UI
Visibility: Private
Relationships: Dependent on Server, Google Maps API

Name: Google Maps API
Visibility: Public
Relationships: UI dependency

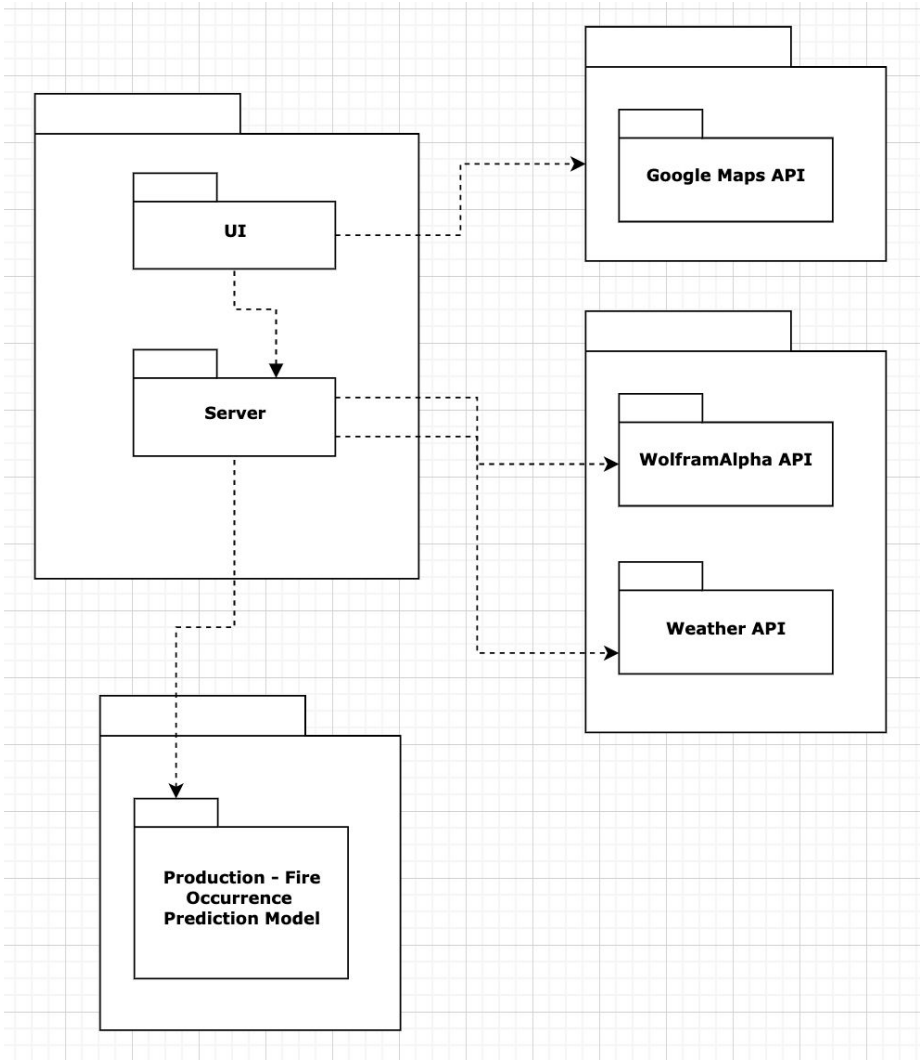
Name: Server
Visibility: Protected
Relationships: Dependent on Production ML Model, dependent on external APIs, UI dependency

Name: Weather API
Visibility: Public
Relationships: Server dependency

Name: WolframAlpha API
Visibility: Public
Relationships: Server dependency

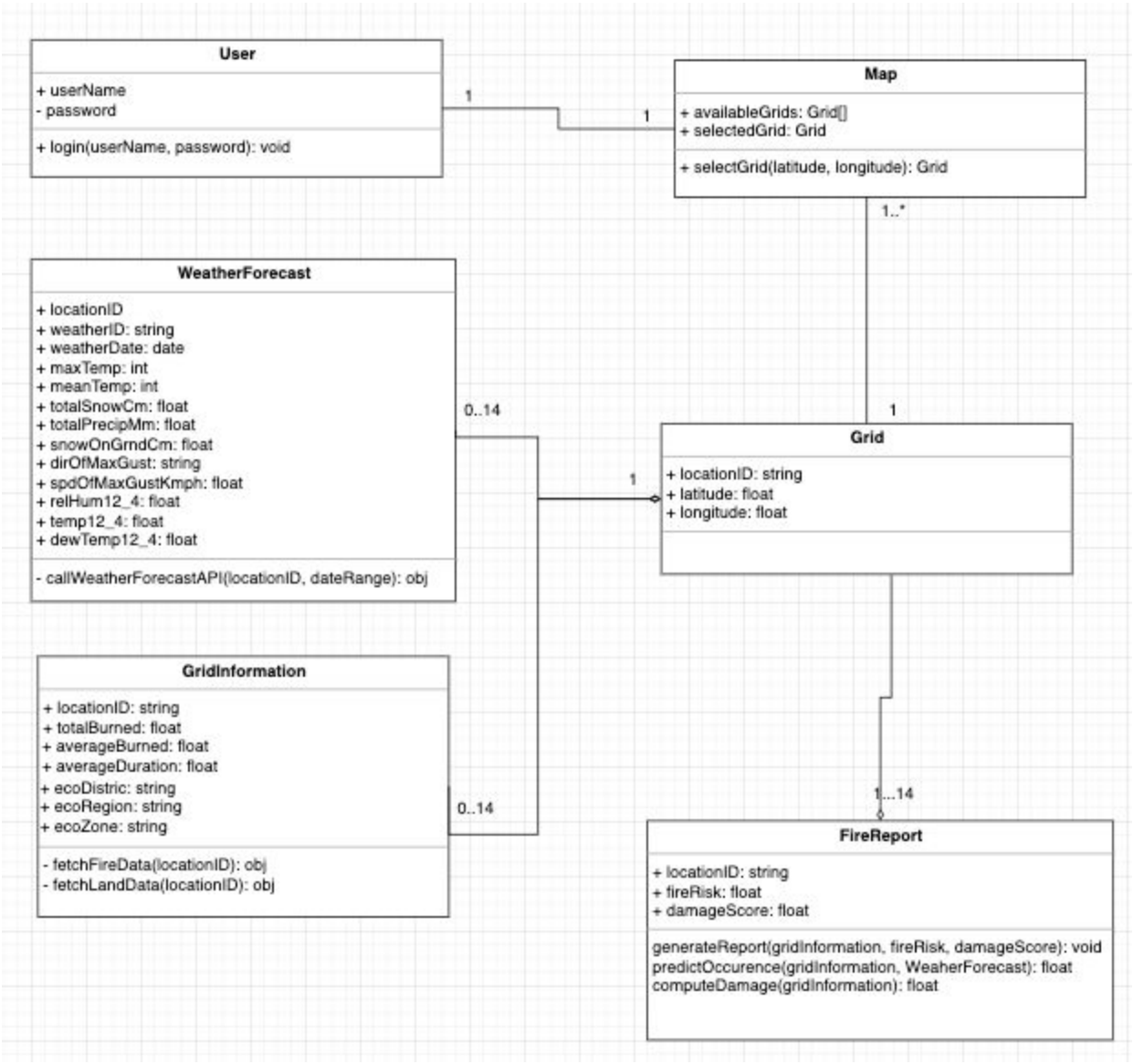
Name: Production - Fire Occurrence Prediction Model
Visibility: Protected
Relationships: Server dependency

2.3a Package Diagram



3. *Detailed Description of Components*

3.1 Production Application Class Diagram



3.1.1 User

The user class will hold the basic information for each user.

3.1.1.1 User Attributes

Name	Type	Description
userName	string	Name of the user
password	string	Password of the user

3.1.1.2 User Operations

Name	Description
login(userName, password)	Function to authenticate users and give access to app functionality.

3.1.2 Grid

The grid class will hold the basic information for each grid location.

3.1.2.1 Grid Attributes

Name	Type	Description
locationID	string	ID representing one grid location on map
latitude	float	The South-East latitude point of the grid location
longitude	float	The South-East longitude point of the grid location

3.1.2.2 Grid Operations

Name	Description
N/A	

3.1.2.3 Design Specification/Constraints

- Latitude and longitude attributes must constitute a valid location within Canada

3.1.3 WeatherForecast

WeatherForecast retrieves and stores all necessary weather information for a specific grid location.

3.1.3.1 WeatherForecast Attributes

Name	Type	Description
locationID	string	ID representing one grid location on map
weatherID	string	ID representing weather station
weatherDate	date	Date for the retrieved weather forecast
maxTemp	int	Max temperature for specified location and day
meanTemp	float	Mean temperature for specified location and day
totalSnowCm	float	Total snow (cm) for specified location and day
totalPrecipMm	float	Total precipitation (mm) for specified location and day
snowOnGrndCm	float	Total snow currently on ground (cm) for specified location and day
dirOfMaxGust	string	Direction of the maximum gust of wind for specified location and day
spdOfMaxGusKmph	float	Speed of maximum gust of wind (km/h) for specified location and day
relHum12_4	float	Average relative humidity measure between 12:00 pm - 4:00 pm
temp12_4	float	Average temperature between 12:00 pm - 4:00 pm
dewTemp12_4	float	Average dew point temperature between 12:00 pm - 4:00 pm

3.1.3.2 WeatherForecast Operations

Name	Description
callWeatherForecastAPI(locationID, dateRange)	Uses the Dark Sky Weather API to get weather information for a specified location and day

3.1.3.3 Design Specification/Constraints

- weatherDate must be in the future with a limit of one month

3.1.4 GridInformation

GridInformation holds additional information about a specified grid location. It is only retrieved for a selected grid location.

3.1.4.1 GridInformation Attributes

Name	Type	Description
locationID	string	ID representing grid location
totalBurned	float	Total historically burned area (hectares) for specified grid location
averageBurned	float	Average historically burned area (hectares) for specified grid location
averageDuration	float	Average duration (days) of previous fires in specified grid location
ecoDistric	string	The ecological district class of the specified grid location
ecoRegion	string	The ecological region class of the specified grid location
ecoZone	string	The ecological zone class of the specified grid location

3.1.4.2 GridInformation Operations

Name	Description
fetchFireData(location ID)	Retrieves historical fire data from database
fetchLandData(locatio nID)	Retrieves land data from database

3.1.5 Map

Map holds all available grid locations that user can select.

3.1.5.1 Map Attributes

Name	Type	Description
availableGrids	Grid[]	Array of available grid objects
selectedGrid	Grid	The grid object that user selects to analyze

3.1.5.2 Map Operations

Name	Description
selectGrid(latitude, longitude)	Function to allow user to select a grid from the list of available grid locations on a valid map

3.1.6 FireReport

Class that creates report based on prediction values.

3.1.6.1 FireReport Attributes

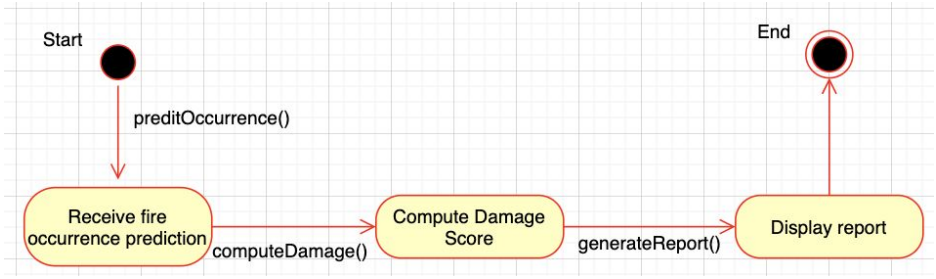
Name	Type	Description
locationID	string	ID representing one grid location
fireRisk	float	A value between 0 and 10 indicating risk of forest fire
damageScore	float	A value indicating a damage metric (percentage scale with 100% being maximum damage possible to

		community and environment).

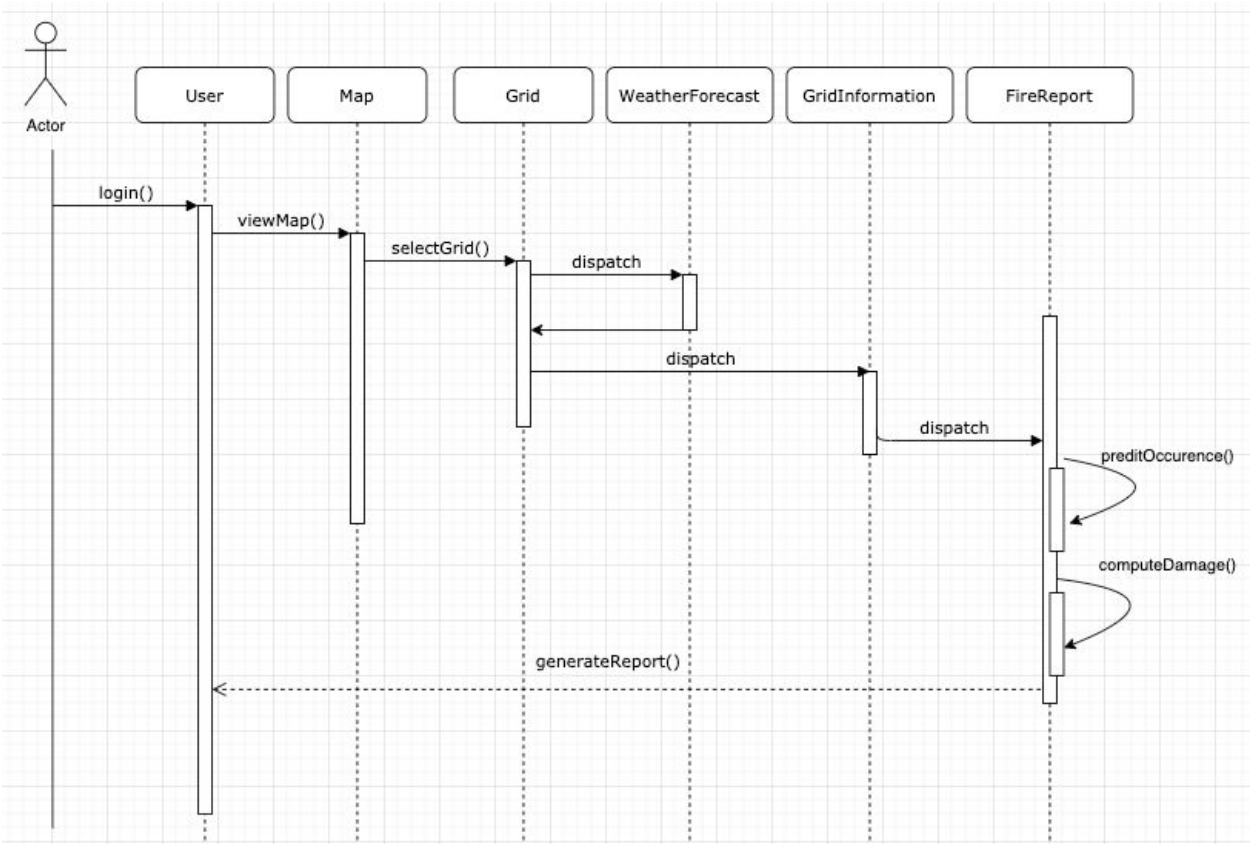
3.1.6.2 FireReport Operations

Name	Description
predictOccurrence()	Make a call to the fire occurrence model. Use gathered grid location, fire, weather and land information as inputs for the model.
computeDamage()	Compute damage score. Use vicinity, land data, and tree coverage data as input to the algorithm.
generateReport()	Combine features and results of predictions into a comprehensive report object to be displayed to the user.

3.1.6.3 FireReport Operations



3.2 Interaction Diagrams



3.3 Relational Database Schema

The production portion of the application will contain several database tables.

3.3.1 Production Database Schema

The ‘Users’ data table will be used to keep track of users who have access to the Forestcasting system. When a given user selects a valid grid on the map, the system will query the associated information from the ‘Land Data’, ‘Historical Fire Data’, ‘Average Daily Weather’, ‘Protected Area Data’, ‘Tree Coverage Data’, ‘Land Data’ tables. This information will then be used as inputs to the fire prediction model and the damage algorithm. All other data relating to the production use of the application will be stored in the application state.

In the training portion of the application, it is exactly the training/testing data table schema that is continuously improved and adapted. It is also the only database table that is stored and can be denoted by the schema in Figure 3.3.2.

3.3.2 Machine Learning Model Database Schema

TrainTestData
KEY, string [key] LOCATION KEY, string MAX_TEMP: float MIN_TEMP: float MEAN_TEMP: float TOTAL_RAIN: float TOTAL_SNOW: float TOTAL_PRECIP: float SNOW_ON_GRND: float DIR_OF_MAX_GUST: float SPD_OF_MAX_GUST: float TEMP_12_4: float DEW_TEMP_12_4: float REL_HUM_12_4: float LATITUDE: float LONGITUDE: float TOTAL_SIZE_HA: float AVERAGE_SIZE_HA: float TOTAL_DURATION: float AVERAGE_DURATION: float ECOZONE: float ECOREGION: float ECODISTRICT: float MONTH: float DAY: float

4. Design Rationale

To explain the rationale behind the selected design it will be necessary to discuss the production and data pipeline plus training portion of the design separately. The training portion which includes data analysis, data wrangling, model training, model testing and model selection follows the standard approach to solving complex problems with machine learning. The design therefore did not have many alternatives.

With regards to the production portion of the application, at first a simple report was considered as a means of portraying the capabilities of the Machine Learning models. Through discussions with the team and supervisor, it was established that a more comprehensive product would be needed to more effectively demonstrate the aforementioned capabilities. A web application was chosen over a native desktop application for many reasons. First, a web application is not platform dependent and therefore much more accessible. Additionally, a web application implies that the product would be Software as a Service (SaaS) meaning the server code and database would live independently of the clients. This setup allows for a much easier way to provide maintenance to the product in the future. Also, in the event that the Machine Learning models are made adaptive (meaning they can continuously re-train themselves based on incoming data), having a SaaS product will yield huge benefits as every user will improve the same model ultimately leading to higher accuracy.

Once a web application was decided, a simple database, server, UI design was chosen following the Model-View-Controller (MVC) framework. This design was chosen primarily for its simplicity. Since the focus of this project is the development of useful ML models it is important to keep complexity in the production application to a minimum. This is also the reason that external weather and maps APIs will be used to aid in developing the UI.

Overall this design provides a straightforward UI which allows access to the Machine Learning model in the backend. Data flow can be easily traced and as a result the product as a whole should not be difficult to test. In conclusion the described design is a foundation of a complete and robust application.

Implementation

Architecture

Python and Google Collaboratory

One of the first decisions that had to be made in the infancy stages of the *Forestcasting* application was which programming language to use for the data analysis and machine learning aspects of the project. The contenders were *Python* and *R*. While *Python* presented a non-existent learning curve for developers, provided better support for model deployment, and was easier to maintain, R had been a tried and true language for statistical analysis with more libraries. Ultimately the presence of an environment that made code easily shareable (Jupyter Notebook), the fact that the gap in statistical analysis tools was closing between the two languages, and the group's more substantial previous experience working with *Python* made it the most practical choice for this application. While it may be useful to have more libraries at the group's disposal, it was unlikely given the scope of the application that the group would exhaust the machine learning libraries currently available with *Python*. Moreover, deployment of the model was very important to the group so it made *Python* an even more desirable choice for the purposes of this application. Later on, the decision to go with Python was further reinforced when the group learned about *Google Collaboratory*, a Jupyter notebook environment that runs on the cloud and gives access to powerful GPUs for free. The ability to share and collaborate on notebooks, and integration with *Google Drive* was an important deciding factor in choosing to use *Google Collaboratory*. The bulk of the hours put into the *Forestcasting* application were spent using *Python* with *Google Collaboratory* for the data analysis, data pre-processing and machine learning stages of the application.

Frontend

Since the bulk of the work in developing the *Forestcasting* application was attributable to data processing and machine learning, it was important to use a frontend framework that allowed the group to efficiently develop the website. The group chose *React* as it is a lightweight library that was easy to learn for group members that haven't used it before and allowed access to many popular user interface and visualization libraries.

Backend API and Database

The Forestcasting application is structured such that the *ReactJS* frontend never communicates directly to the database and external APIs. The backend *Forestcasting* API is queried from the *ReactJS* frontend each time a user chooses to analyze a region for forest fire risk. The frontend specifically sends latitude, longitude, and a range of dates to the API. The API responds with the results from the machine learning model, damage algorithm, and data from the

database. The machine learning model itself is on a standalone API built on Flask. The backend API queries the machine learning *Forestcasting* API to get results. The backend API also sends additional information such as: average and total fire durations in that region, ecozone information, predicted weather for each of the days in the range, closest city, list of protected areas, elevation, vegetation and water cover information. The additional information is displayed as supporting information for the user to see on the results page. Weather information the API returns is pulled from the *DarkSky Weather API* for predictions seven days from the current date, the rest of the information is pulled directly from the *MongoDB Atlas* cloud database.

Data Pipeline

Grid Creation: Weather, Fire, Stats, Ecoregion Information

For the purposes of the Forestcasting application, the map of Canada needed to be divided into distinct regions so that it would be easy for the machine learning model to distinguish between different areas in Canada. The various regions in Canada could then be assigned a forest fire risk value depending on the characteristics of that specific region. It was also important that the regions do not get larger as the area becomes remote, meaning census or county divisions could not be used. Census and county divisions typically group very large remote areas of provinces into the same region. Many of the regions in which forest fires happen often would be in more remote locations and it was important that the data contains characteristic information on similarly sized regions. As a result, a grid system was developed that divides all of Canada into 20km by 20km squares, where each square's location key is generated by rounding the latitude and longitude down to the nearest 0.2. This location key generation method gives an O(1) lookup time when given a user selected latitude and longitude, improving the performance of the analysis API call. In order to incorporate weather data into the analysis, a weather station needed to be mapped to each grid location. Closest weather stations were mapped to each grid by calculating distance using the haversine function. The government of Canada provides forest fire data for all of Canada. This information was mapped along with ecoregion information to each location key in the grid system. This resulting joined dataset forms the basis of the unprocessed data.

Historic Statistics and Filling in Missing Data

In order to relate historic frequency and severity of forest fires to the possibility of a forest fire happening today, four different statistics were developed to associate with each region. These statistics aggregate historic forest fire data in a region based on *Total Duration, Average Duration, Total Size, and Average Size*. However, some of the regions in the forest fire data that was collected did not have end dates, this created a problem for developing the statistics. The solution to this problem was to fill in the blank data using some form of prediction. Initially

correlations were explored between forest fire duration and forest fire size using multiple different types of correlations such as linear, polynomial and exponential. When there was no strong correlation, it was determined that filling in missing values using a distribution was the best option. Taking a look at the data showed that forest fire duration data is heavily skewed to the right. Normal distributions, lognormal distributions, poisson distributions and alpha distributions were all explored to determine the best fit. In the end, the alpha distribution best fit the data and this distribution was used to fill in missing duration values.

Preprocessing

In the pre-processing step, the resulting dataset from the above two steps was modified to fill in null values, create new features and modify the data so that it will be best interpreted by the machine learning model. For example, our research showed that there can never be a fire when there is snow on the ground, so the value in the dataset for the amount of snow on the ground was reduced to a boolean value that represents whether or not snow is on the ground. It was also important to convert 1-12-month values into a cyclical form of representation in order to allow the machine learning model to understand that the 1st month follows the 12th month. Month values were transformed using a sine and cosine curve to take on cyclical values. After preprocessing, the dataset was ready to be inputted into the machine learning mode.

Machine learning

Dealing with Large Dataset

After processing data for British Columbia (BC), Alberta (AL), Saskatchewan (SK), Manitoba (MB), Ontario (ON), Quebec (QB) all the data was concatenated into a single dataset. This multi-province dataset contained 16,409,058 rows. *Google Collaboratory* provides a fast Tensor Processing Unit with 35GB of RAM and 108GB of memory, however many operations on the multi-province data caused the backend to run out of RAM. Thus, a randomly sampled subset, 20%, of the multi-province dataset was taken. Originally the dataset has 30 columns, however the Ecozone, Ecoregion, and Eco district are categorical features and thus must be transformed into one hot encoding features and so the column number is expanded to 458. This is specifically the reason that the entire multi province dataset cannot be easily processed and thereby used for training and testing.

Dealing with Imbalance

Naturally, the dataset is imbalanced, there are many more day-grids with no forest fires than day-grids with forest fires. In the sampled dataset referenced above, 0.102% of all days have a fire. There are multiple techniques which can be used to address dataset imbalance, in this case an abundance of data makes down sampling a valid option. Down sampling involves removing some amount of the data points that make up the majority of the dataset. Down sampling would

only be done on the training dataset, so that the metrics gathered from predicting on the test set still reflect real world performance.

Since the training dataset size would be reduced through down sampling, only 10% of the original sampled dataset was set aside as the test set. The remaining 90% was denoted as the training set and down sampled. Originally the training set was down sampled to have a 60/40 split; 40% fire data, 60% no fire data. Later in the process after model selection different ratios of fire to no-fire were tested using cross validation such as making the fire data points make up 1%, 5%, 10%, 20%, and 40% of the training dataset. Using cross validation, the best result was achieved with down sampling the dataset to have 5% fire data.

Model Optimization

Apart from negative log likelihood loss, there are several metrics that can be used to evaluate the model. The following metrics were considered in the decision process:

1. Accuracy is not useful because the dataset is highly imbalanced. Classifying each example as 'no fire' would yield a very high accuracy as there are many 'no fire' examples, however this model would be useless.
2. Precision ensures that 'fire' will only be predicted if the example is almost certainly a fire. This means many 'fire' examples will be misclassified however for the context of forest fires it is better to have a false alarm than missing a forest fire so precision is not sufficient.
3. Recall ensures that most 'fire' examples will be predicted but also that many 'no fire' examples may be predicted as 'fire'. This metric is forgiving to false alarms but not complete misses and therefore makes more sense because in preventing forest fires it's best to be overly cautious.
4. Negative Log Likelihood - Essentially ensures the model fits data as best as possible. Log likelihood does not consider any business context and so does not consider any one class with higher importance.
5. F1 Score is the harmonic mean of precision and recall and ensures there is a good balance between precision and recall.
6. AUC ROC (Area Under Curve) can be interpreted as the probability that a day-grid with fire will receive a higher risk score than a day-grid with no fire. AUC is classification threshold invariant so the prediction probabilities will be measured regardless of the threshold set.

The AUC ROC was selected to evaluate the model as it was suitable for imbalanced datasets and also allowed for individual forest fire managers to select a decision threshold as suitable for them using the ROC curve.

Model Selection

After the training and test sets were complete several classifier models from various libraries were trained on the training set and evaluated using AUC ROC. The best results through cross validation with default hyper parameters were achieved with the following models:

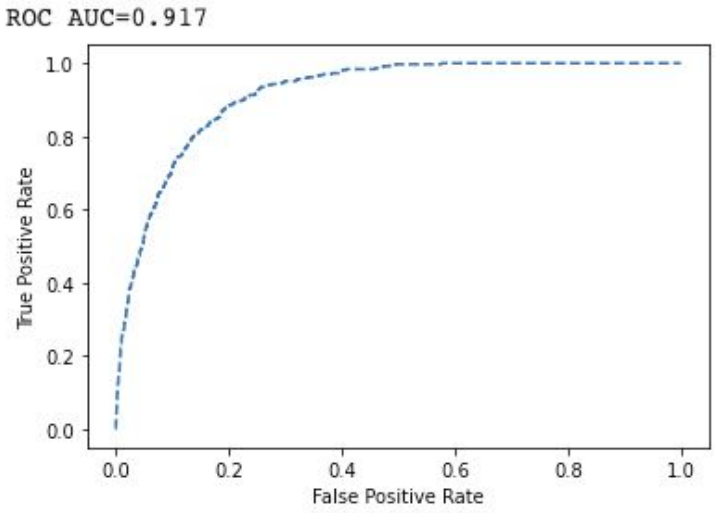
- Random Forest Classifier: 0.9058
- XGBoost Classifier: 0.9093
- CatBoost Classifier: 0.9170
- LightGBM Classifier: 0.9156

Each of these models was run through RandomSearch with cross validation to test various hypermaters. RandomSearch finds the best parameters by randomly sampling the available hyperparameters and narrowing down the list by checking the scoring metric on each iteration. Unlike GridSearch, RandomSearch is not exhaustive, however it is much faster than GridSearch and very often achieves the same result as GridSearch therefore it was ideal for using on Google Colab with limited processing time. After hyperparameter tuning was complete the models had the following AUC ROC scores:

- Random Forest Classifier: 0.9165
- XGBoost Classifier: 0.9170
- CatBoost Classifier: 0.9173
- LightGBM Classifier: 0.9164

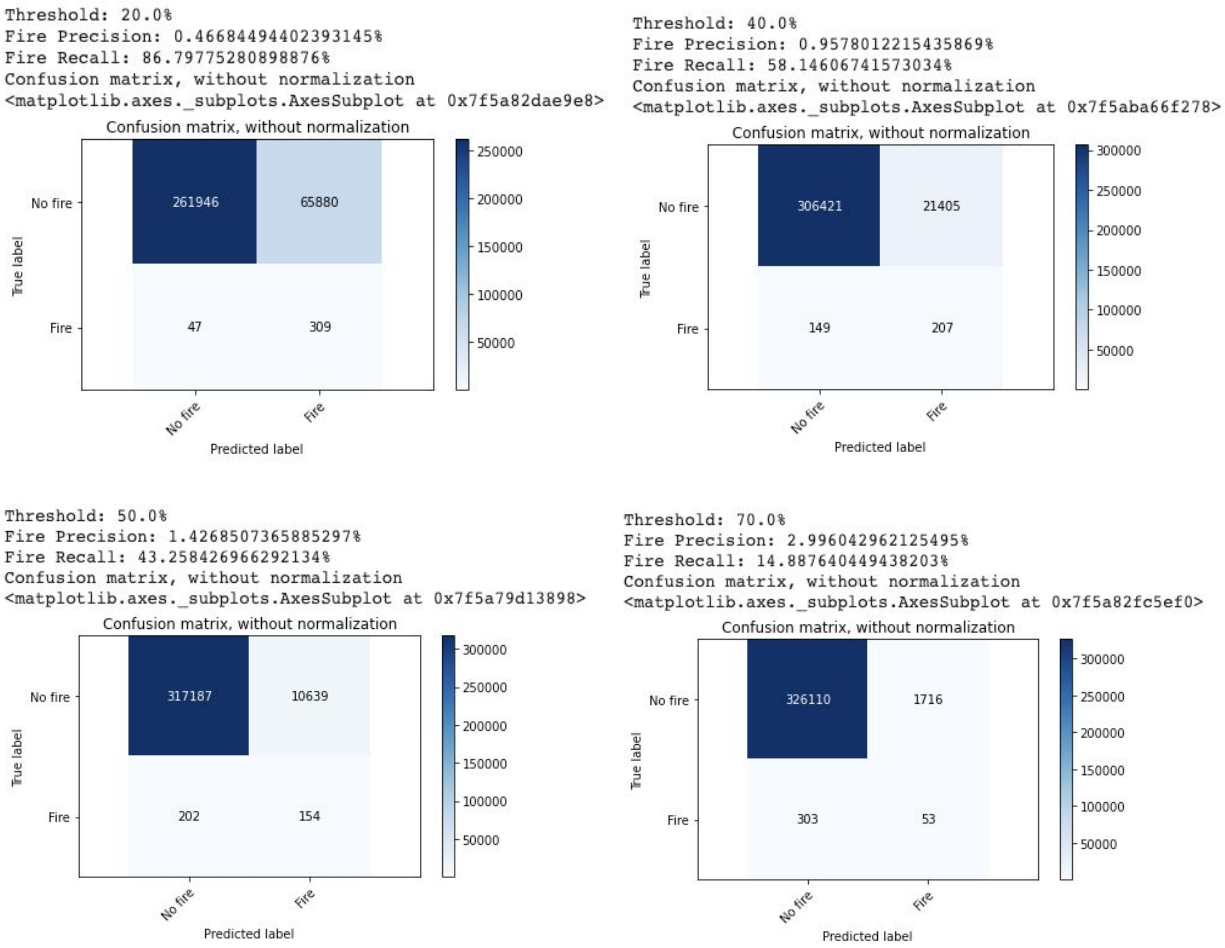
The AUC ROC scores of each model were nearly the same so the Random Forest Classifier was chosen for its relative simplicity. A simpler classifier meant less chance of overfitting and a smaller production model size.

Figure 4.3.1 ROC curve on cross validated model Random Forest Classifier



Using the confusion matrices at different thresholds a forest fire manager could decide which precision and recall ratio is best suitable for their needs.

Figure 4.3.2 Confusion Matrices at Different Decision Thresholds (on Test Set)



Deployment

After training and testing was complete the trained model was exported as a .joblib.z file. Then a Python Flask Rest API was created which imported the trained model and allowed a "/predict" endpoint. This Flask Rest API was then deployed as an *AWS Lambda* function and made accessible through *AWS API Gateway*.

Damage Algorithm

Damage Score Purpose

Forestcasting provides two main metrics:

(1) forest fire occurrence probability and;
(2) potential damage given a fire occurs.

The second metric is referred to as the damage score. The purpose of the damage score is to provide a numerical representation of the damage level a fire would cause in the selected location. The algorithm evaluates a given area and produces a percentage that represents the severity of damages relative to the worst case scenario. The damage score gives forest fire managers essential information for allocating their relief efforts as each location will have a unique damage score.

Terminology

Metric: The raw data value before normalization and/or scaling is applied
Score: The value after normalization and/or scaling is applied

Data Pipeline

Initially, the damage score would be calculated using a machine learning model, however without formal tags to train the model a mathematical algorithm was developed instead. The damage algorithm considers three metrics: number of protected areas, tree coverage percentage, and vicinity score.

(1) Protected Areas Metric

The protected areas dataset included a shapefile of all protected areas in Canada. This included areas such as national parks, conservation areas, critical habitats, and historical sites. The purpose of the protected areas score is to quantify at-risk areas identified by current governing bodies that would be put at risk by a fire in the selected location. Each location key was mapped to the protected areas within their grid resulting in a number of protected areas per grid. These numbers range between zero and seven where a higher number of protected areas results in a higher protected area score.

(2) Tree Coverage Metric

Tree coverage percentage was gathered through the *Google Earth Engine* API. For each grid the amount of tree coverage in that area was calculated and assigned to the location key. This metric represents the tree coverage that would be lost or damaged if a fire occurred in that location, therefore a higher percentage results in a higher tree coverage score.

(3) Vicinity Metric

The vicinity score is calculated using the *Wolfram Alpha* API. Given a longitude and latitude *Wolfram Alpha* returns the closest city's population, distance to the city, and supporting information. The vicinity score represents the danger a fire poses to the public, and highlights the possible need for evacuation. A large population at a small distance will result in a high vicinity score.

Calculations

Step 1: Scaling

Each metric had unique numerical ranges, in order to combine these three values in a meaningful way they needed to be scaled to a value between zero and one. The min-max method was chosen as each value had a distinct range.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

The above scaling was sufficient for tree coverage and protected areas; however, vicinity required an extra calculation. The vicinity score did min-max scaling on both distance and population, then performed one minus the scaled distance value to reflect small distances as large risk, finally two scaled values were combined using a weighted sum.

$$Vicinity\ Score = [0.5 \times (1 - scaledDistance)] + [0.5 \times scaledPopulation]$$

Step 2: Normalization

The vicinity score and tree coverage score had suitable distributions; however the protected areas score was left heavy (*Figure 4.4.1)*. To resolve this a logarithmic normalization was applied to the scaled protected areas values; the resulting distribution is shown in *Figure 4.4.2*. The normalized data maps the linear protected area metric values to a non-linear output. This means that going from zero protected areas to one protected area will cause a larger output score to change than going from six protected areas to seven. For example, in *Figure 4.4.2*, zero to one protected area has a score change of 0.4 but four to five protected areas only results in a score change of 0.2.

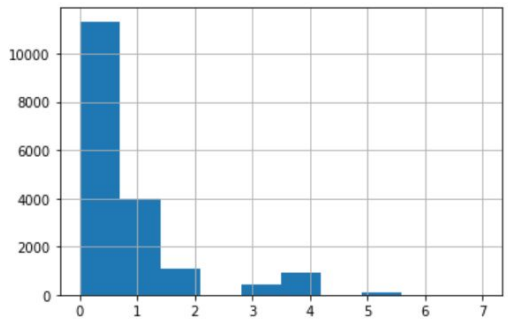


Figure 4.4.1: Protected Area Distribution Before Scaling and Normalization

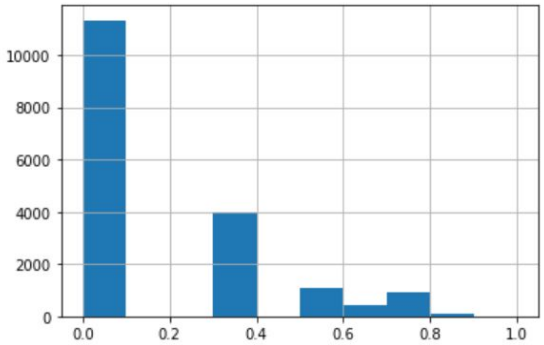


Figure 4.4.2: Protected Area Distribution After Scaling and Normalization

Step 3: Weighted Sum

With each metric scaled, and normalized to a value between zero and one, they can be combined to calculate the final damage score. The scores have default weightings of 0.33 and are combined using a weighted sum:

$$DamageScore = \sum_{x} weight(x) \times scored(x)$$

The weights are configurable through the UI to provide forest fire managers with a damage score that accurately represents their priorities.

Testing Document

Table of Contents

Acceptance Testing Plan

#	Page	Action	Expected Result
1.	**Login Page**	Navigate to the website in any popular browser (Google Chrome, Mozilla Firefox, Opera, Microsoft Edge, Internet Explorer)	System loads the website for the user with the "Login" web page opened.
2.	**Login Page → Location Search**	User enters credentials and clicks on the "Login" button.	System authenticates the user and loads the "Location Search" webpage with the maps API.
3.	**Location Search**	User drops pin on an invalid location.	System displays a warning guiding the user to select a valid location according to the "Supported Areas" map.
4.	**Location Search**	User clicks the "Supported Areas" button.	Pop-up appears with a map of Canada and supported locations.
5.	**Location Search**	User drops pin on a valid location.	Message appears validating the user has selected a valid location. Address selected appears below the map. "Next" button becomes available.
6.	**Location Search**	User searches a valid location.	Pin appears on the corresponding location on the map. Address selected appears below the map. "Next" button becomes available.

7.	**Location Search**	User clicks the “Next” button before selecting a valid location.	Button is unclickable.
8.	**Location Search**	User clicks the “Next” button after selecting a valid location.	Button is clickable. Date selection feature appears.
9.	**Location Search**	User selects a valid date range.	“Analyze” button becomes available.
10.	**Location Search → Results Page**	User clicks on the “Analyze” button.	System redirects the user to the “Results” page, the request is fed into the ML models, and an analysis is presented on the “Results” page.
11.	**Results Page**	User clicks on the “Generate PDF” button.	PDF of the ML results are downloaded to the user’s computer.
12.	**Results Page → Login Page**	User clicks on the “Logout” button.	System loads the “Login Page”.

System Testing Plan

Each element of the entire system must meet the functional requirements as outlined in the Functional Requirements section of the Software Requirements Specification (SRS) document.

Overall Strategy and Approach

1. Identify functions that the system is expected to perform.
2. Create test data based on the function's required input(s).
3. Compute the expected outcome with the selected test input(s).
4. Execute test cases.
5. Compare actual and computed test results.

System Testing Scope

The Forestcasting system is comprised of 3 main subsystems:

1) User Interface

- Account login, and logout
- Area selection via pin drop
- Area selection via search
- Date selection via calendar
- Results display
    - Risk measure per date selected
    - Weather per date selected
    - Ability to flip through selected dates
    - Possible damages of selected area
    - Adjustment of damage factors
    - Ecozone information

2) Machine Learning Model and Algorithm

- The risk model should predict the likelihood of a fire occurring in a given region on a given day.
- The damage algorithm should calculate the possible damages of a fire in a location.

3) APIs

- Communicate with the *Weather* API to get current weather data.
- Communicate with the *Google Maps* API for region selection.
- Communicate with the *Wolfram Alpha* API to get vicinity data.

The main subsystems will be tested for functionality along with:

- Usability: whether a user can easily navigate the web page without any difficulties.
- Error conditions: whether suitable error messages are displayed.

System Test Cases

ID	**Test Cases**
TC1	*Login: Invalid Credentials* User logs in with the following credentials: Username: abc Password: abc Expected Results: Error appears for inputting wrong credentials.
TC2	*Login: Valid Credentials* User logs in with the following credentials: Username: abcdef Password: 23456 Expected Results: User successfully logs into the system.
TC3	*Select Location: Pin Drop* User selects a valid location by dropping a pin and clicking on the map. Expected Results: The address of that location is obtained including the latitude and longitude coordinates. A message is displayed to the user to let them know they have selected a valid location, along with the address. Then, the user is rerouted to the dashboard to view the analysis report.
TC4	*Select Location: Invalid Selection* User drops a pin in a location in Southeast Europe and presses the Analyze button. Expected Results: Invalid location is selected, an error message is displayed to the user and they are

	prompted to view the supported areas map.
TC5	*Select Location: Search* User searches for an address by typing the address in the location search box. Expected Results: If a valid location is selected, the address of that location is obtained including the latitude and longitude coordinates. A message is displayed to the user to let them know they have selected a valid location, along with the address. Then, the user is rerouted to the dashboard to view the analysis report.
TC6	*Select Date: Calendar* Users are prompted to select prediction dates after selecting a location. The calendar tool can be opened, and the user can click on the start date and end dates to highlight a range. Expected Results: The user can see the number of selected days and the start and end dates are sent to the server for prediction.
TC7	*Select Date: Range* If the user selects a range greater than 16 days, the system automatically highlights the first 16 days of the selected range and a message is displayed to the user to remind them of the recommended range. This limitation keeps the efficiency of the application intact. Expected Results: The user can see the adjusted number of selected days and the warning message reminding them of the recommended range.
TC8	*Results Display: Appropriate Sections Present* Users are rerouted to view the analysis in the dashboard. Expected Results: The dashboard should contain the following sections: “Fire RiskPrediction by Date”, “Possible Damages”, and “Ecozone Information”.
TC9	*Results Display: Risk Measure and Weather per Date Selected* Users can click on the left and right arrows of the prediction range to view the fire and weather data per selected date.

	Expected Results: The risk meter and weather data will update accordingly to display information for each day.
TC10	*Results Display: Possible Damages of Selected Area* Users can view possible damages score on the right-hand side of the results page. Expected Results: The possible damages must include the overall score and weightings of each factor, information about each factor, and the ability to adjust the factors' weightings.
TC11	*Results Display: Adjustment of Damage Factors* The pie chart shows the weighting of the factors that went into calculating the Possible Damages. The default weighting is initially equal for all factors. Users can adjust the weighting by increasing or decreasing any of the factor's weighting or entering the desired weightins in the input boxes. Expected Results: The overall score and the pie chart will update to reflect the changes, the other factors' weightings will automatically adjust accordingly.
TC12	*Logout* User clicks the logout button. Expected Results: User is rerouted to the main page where it is required to enter credentials to log into the system again.
TC13	*Damage Algorithm: Input Data Quality Test* The data being fed to the damage algorithm is tested to see if there are any outliers that may result in unrealistic outcomes. The data for each metric going into the damage algorithm is graphed to ensure a normal distribution. Expected Results: If the data does not have a normal distribution, then a logarithmic scale is applied, and the graph is double checked to ensure a normal distribution. If there are any outliers, they are removed.
TC14	*Machine Learning Model: Fire Risk - Validation*

	Once the model is created using the training dataset (the sample of data used to fit the model), the model is tested using a validation dataset to evaluate the model. The predictions for the validation dataset are known beforehand and the model is not trained on this dataset. Expected Results: The model outputs predictions for the likelihood of a fire occurring in the given region on the given day. The predictions are tested for accuracy and an overfitting model by measuring how distant the predictions are from the actual results of the validation dataset.
TC15	*Machine Learning Model: Risk - Testing* The model is fed a testing dataset with labels relevant to the risk model for which the predictions are known beforehand. The risk model should have not been previously trained or tested on this dataset. Expected Results: The model outputs predictions for the likelihood of a fire occurring in the given region on the given day, within the known accuracy range.
TC16	*Machine Learning Model: Risk - AUC* The model is evaluated using the AUC ROC (Area Under Curve) metric. The probability that a day-grid with fire will receive a higher risk score than a day-grid with no fire presents a classification threshold. Therefore, the prediction probabilities are measured regardless of the threshold set. Expected Results: Forest fire managers can select a decision threshold that is most suitable for them using the ROC curve.
TC17	*Weather API Communication: Invalid Input* The API is called with invalid latitude and longitude points. Expected Results: The API returns an error and the error handling function is called to display the appropriate message to the user.
TC18	*Weather API Communication: Valid Input* The API is called with valid latitude and longitude points.

	Expected Results: The API returns the weather data for the respective latitude and longitude points and the data processing function is called to pass the results to the respective Machine Learning model.
TC19	*Google Maps API Communication: Invalid Input* The API is called with invalid latitude and longitude points. Expected Results: The API returns an error and the error handling function is called to display the appropriate message to the user.
TC20	*Google Maps API Communication: Valid Input* The API is called with valid latitude and longitude points. Expected Results: The API returns the region data for the respective latitude and longitude points and the data processing function is called to pass the results to the respective Machine Learning model.
TC21	*Wolfram Alpha API Communication: Valid Input* The API is called with valid latitude and longitude points. The coordinates must be converted to DMS format first. Expected Results: The API returns the respective data for the latitude and longitude points which includes information about nearby cities and population. This information is passed to the damage algorithm to calculate the vicinity score.
TC22	*Wolfram Alpha API Communication: Invalid Input* The API is called with invalid latitude and longitude points. Expected Results: The API returns an error and the error handling function is called to display the appropriate message to the user.

Conclusion and Recommendations

Recommendations

Improved Technologies

Forestcasting was limited to using open source technologies and the project had a 7-month lifecycle. These two limitations forced the team to make decisions that expedited development and sacrificed robustness of the application. *Forestcasting's* data pipeline consists of processing gathered data and training the ML model through a set of *Python* notebooks on *Google Collaboratory*. *Figure 6.1.1.a* depicts the current data pipeline, where each node is a *Python* notebook that inputs and outputs a CSV. With the current data pipeline, any updates must be completed by running the notebooks manually. *Forestcasting* would be greatly improved by automating this process through the use of existing data processing technologies.

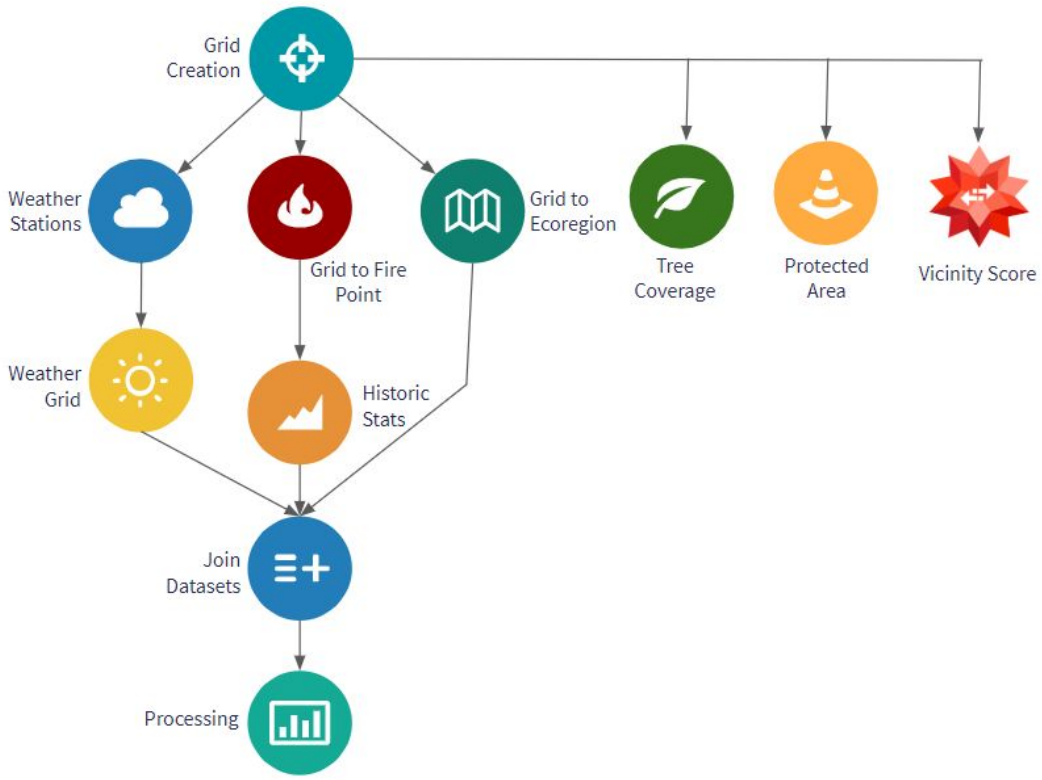


Figure 6.1.1.a: Data Pipeline

Data ingest could be optimized with streaming technologies such as *Spark Streaming* or *Apache Kafka.* The streaming technology would load the base datasets into a non-relational database suited for big data, such as *HDFS* or *Vertica.* Streaming would allow the base datasets to be updated regularly such that updates would trigger all following data updates. Once the base datasets were ingested, data manipulation and cleaning would be done through the *Apache Spark* API using *Python* or *Scala.* All intermediate CSVs would be eliminated, and data stored for the application, currently in *MongoDB Atlas,* would be updated automatically. These technologies would streamline the data pipeline; however, the team was limited to their individual PCs and the proposed implementation would require either (1) a cloud budget or (2) local devices with high computing power. *Figure 6.1.1.a* shows the new implementation that minimizes manual processes and simplifies CSV organization.

Forestcasting currently uses *MongoDB Atlas* as the cloud database system for the application, however as the application expands the current daily weather average cloud storage will not be sufficient. The dataset contains a daily weather average for each day of the year per grid; the size of this collection is already beyond *Forestcasting's* budget. As this collection grows it will become the slowest query and will bottleneck performance. *ElasticSearch* provides extremely quick querying using the lucene language and takes advantage of indexing to improve query times. Although the storage must remain on the cloud, *AWS* supports *ElasticSearch* and moving all cloud data to this storage implementation would easily integrate with the *Apache Spark* API and prevent future querying performance issues. *Figure 6.1.1.b* shows the new architecture of *Forestcasting* with the proposed technologies, for current architecture see *Figure 6.1.1.c: Current Architecture*.

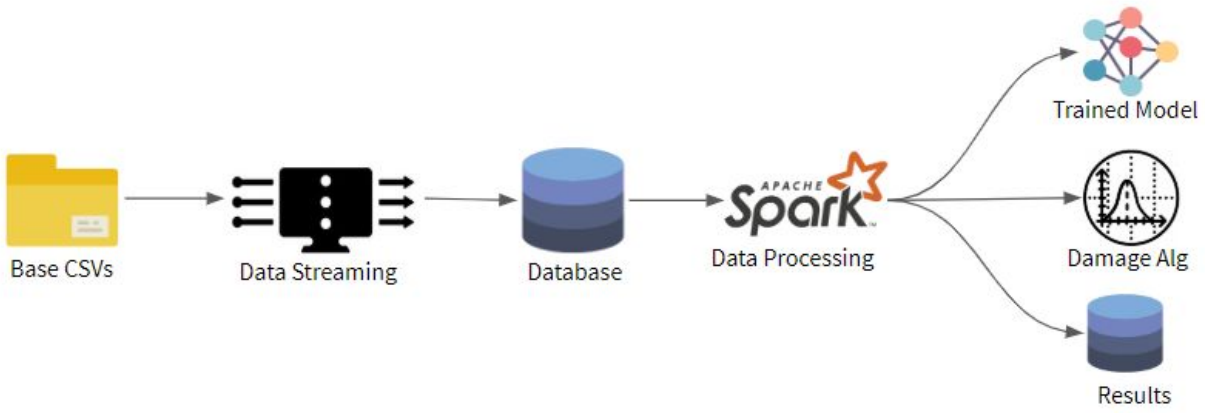


Figure 6.1.1.a: Proposed Data Pipeline

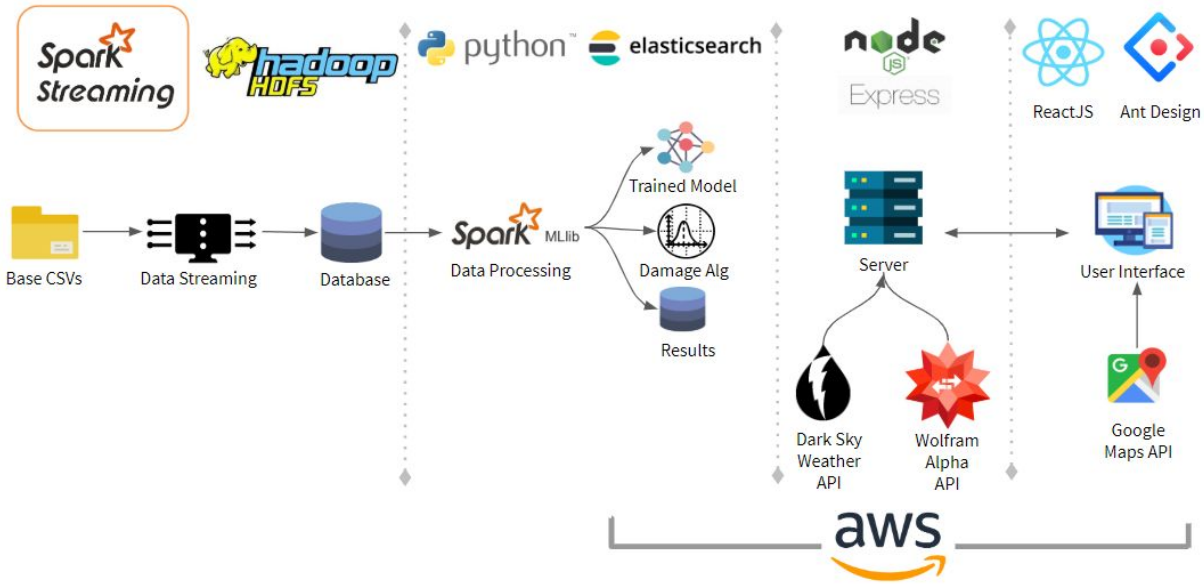


Figure 6.1.1.b: Proposed Architecture

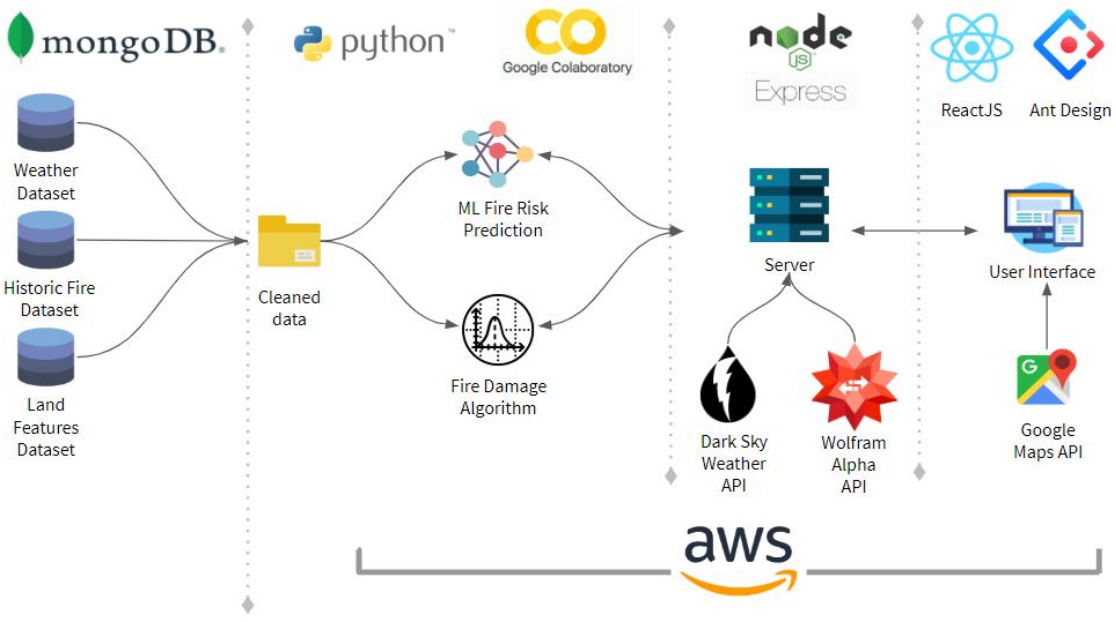


Figure 6.1.1.c: Current Architecture

Increased Resources

Google Collaboratory provides great processing power, however after continuous usage the TPU and GPU access becomes restricted and thus makes data processing and model training difficult. With a cloud budget technologies like *AWS SageMaker* can be used to have longer

uptime and allow for more comprehensive processing and training. *AWS SageMaker* is a fully managed service for building, training, and deploying machine learning models on AWS.

Considering model deployment, using an *AWS Lambda* function has size limitations. Complex models such as the *CatboostClassifier*, which achieved slightly lower loss compared to the production model, require over 100 megabytes of space. Therefore, with other dependencies required to make the model into an API, many complex models cannot fit into an *AWS Lambda* function. Again, with additional budget, alternatives such as *AWS SageMaker* can be used to deploy larger sized more complex models.

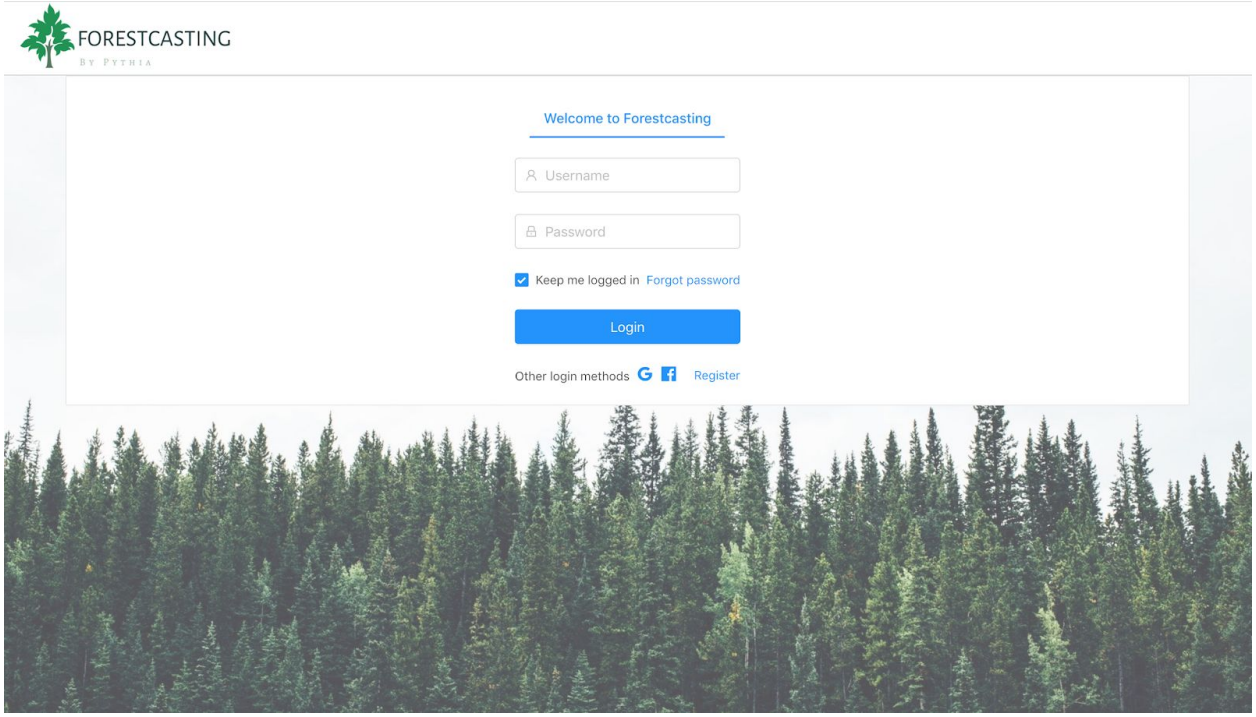
Conclusion

Several enhancements can be made to improve the production readiness of Forestcasting such as streamlining the data pipeline through the application of data ingest technology and increasing resources for model training and deployment. These improvements would allow for rapid updates and retraining when new data is collected. In general, similar data to what was collected for Canada can be collected for other countries with risk of forest fire. With the right adjustments the entire application can be made to work on other parts of the globe. By providing forest fire managers with a comprehensive tool to evaluate the risk and damage of a given location up to a year ahead, Forestcasting enhances the decision making of those managers. Overall, by taking a more proactive approach to forest fire management, Forestcasting stands a strong chance of helping the public sector in reducing forest fire damages to communities and the environment.

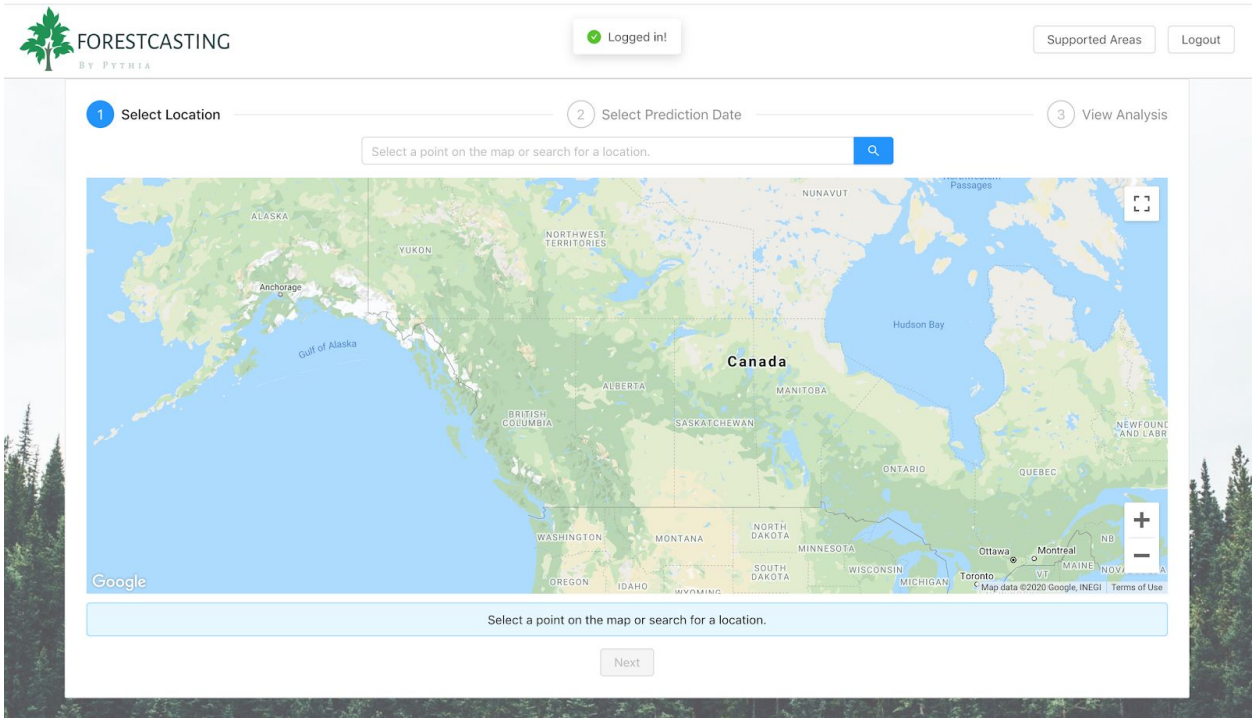
User Manual

Authentication

Log in to the app by providing user credentials (username and password)



Successful login - user is prompted to select a location for prediction.

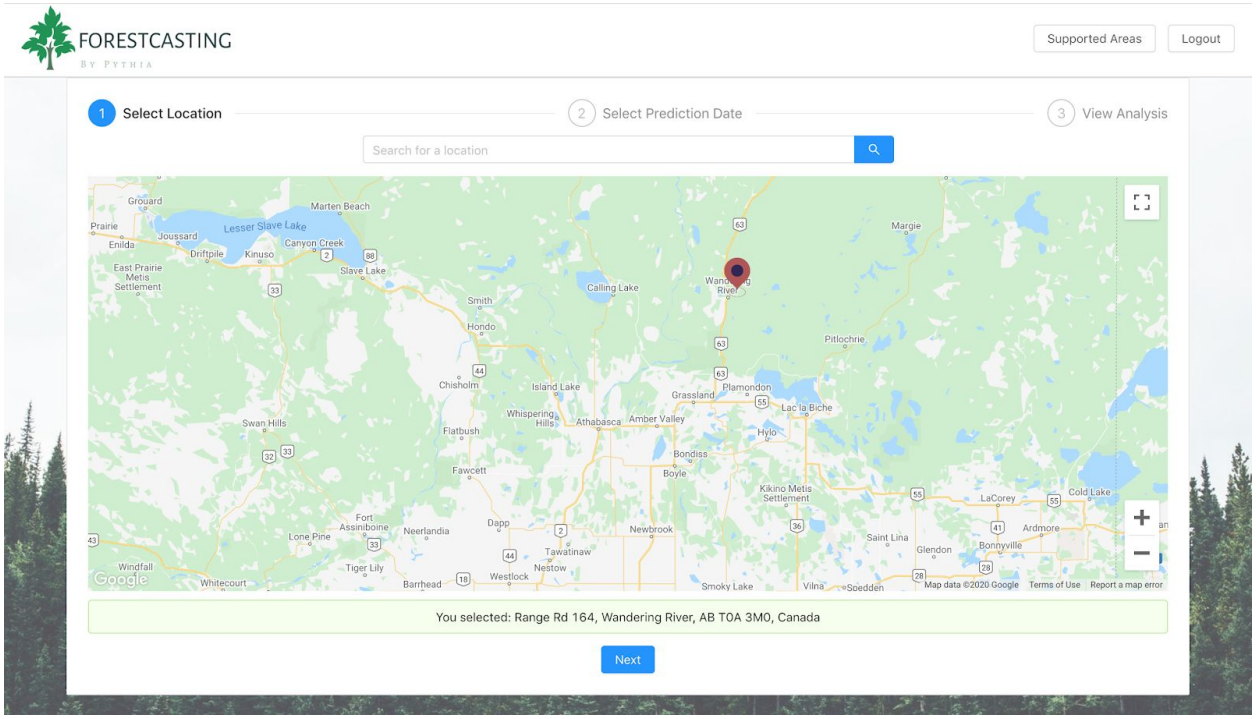


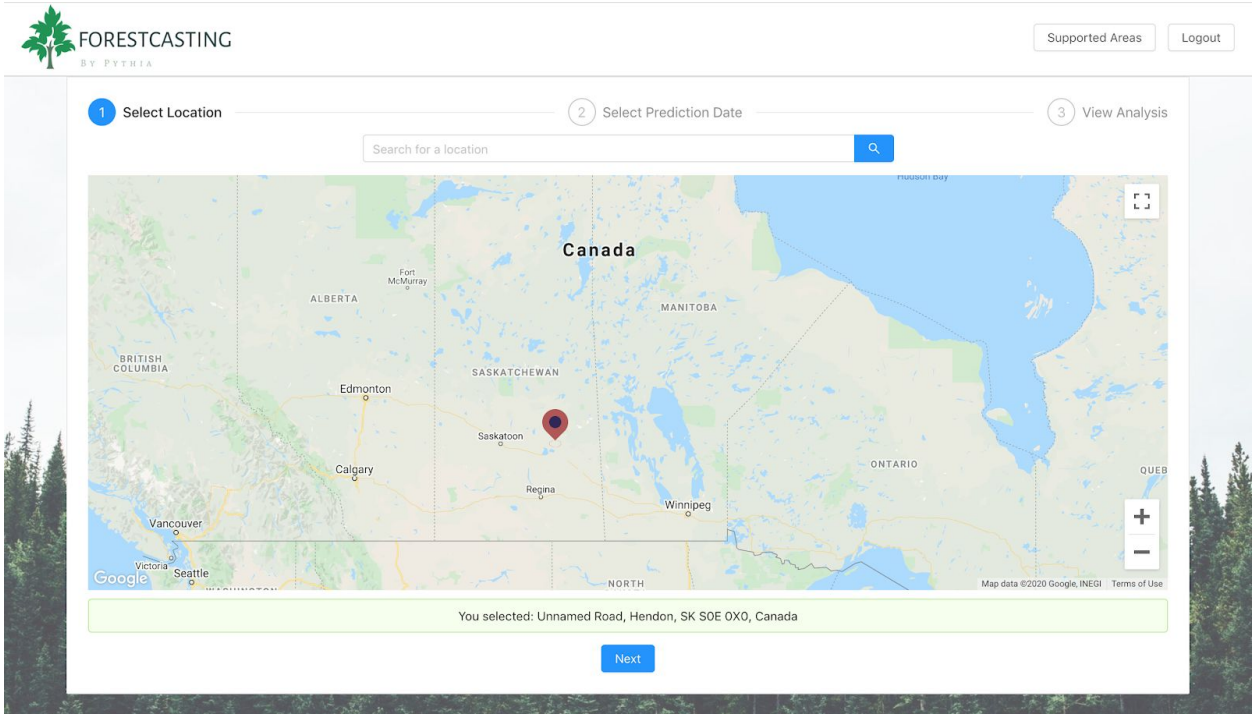
Selecting a Location from Map

Use the map tool to select the desired location:

a. Zoom in to clearly see the details of the map.

b. Click on the map to drop a pin and select that location for analysis, followed by Next.

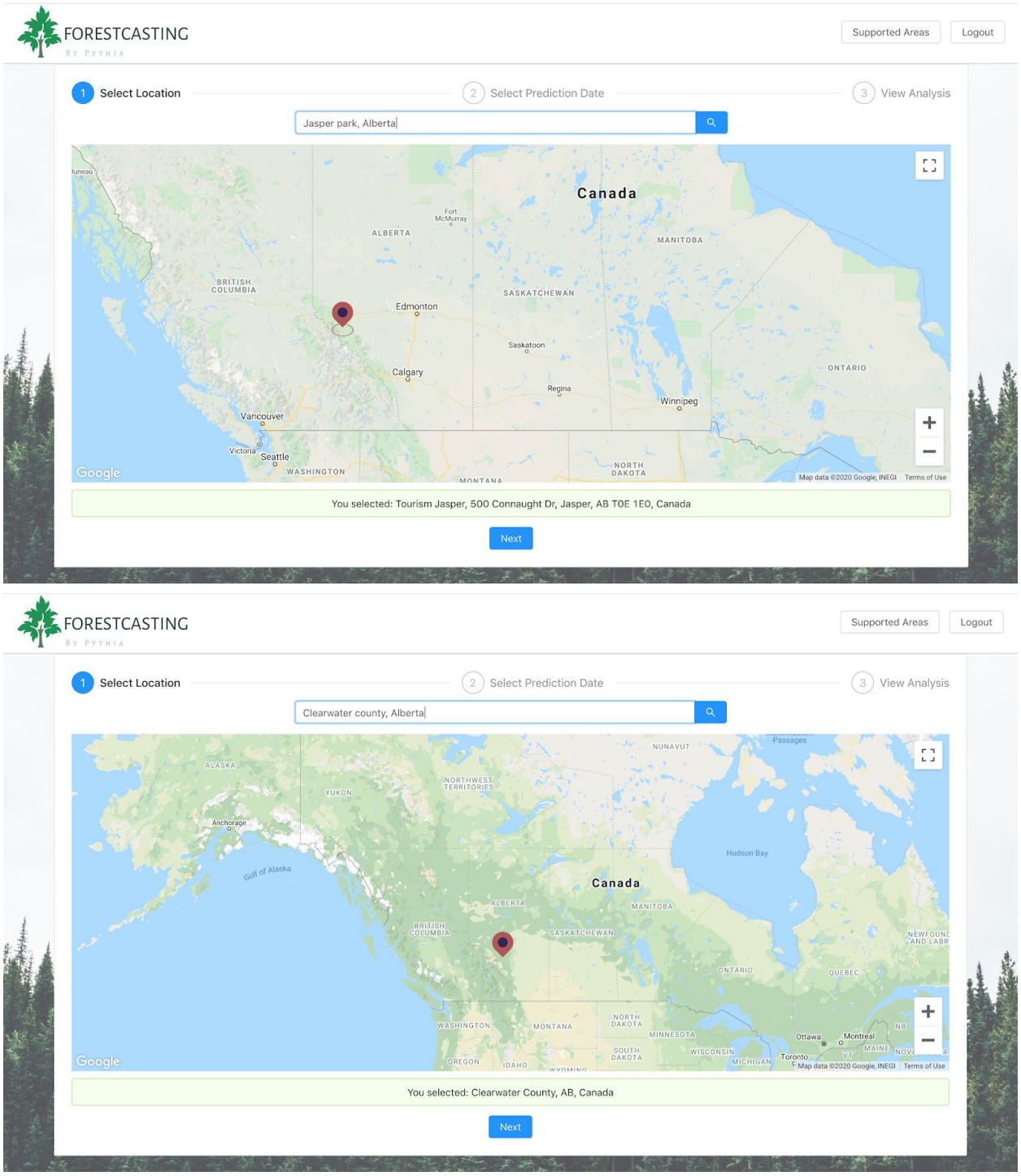




Selecting a Location From Search

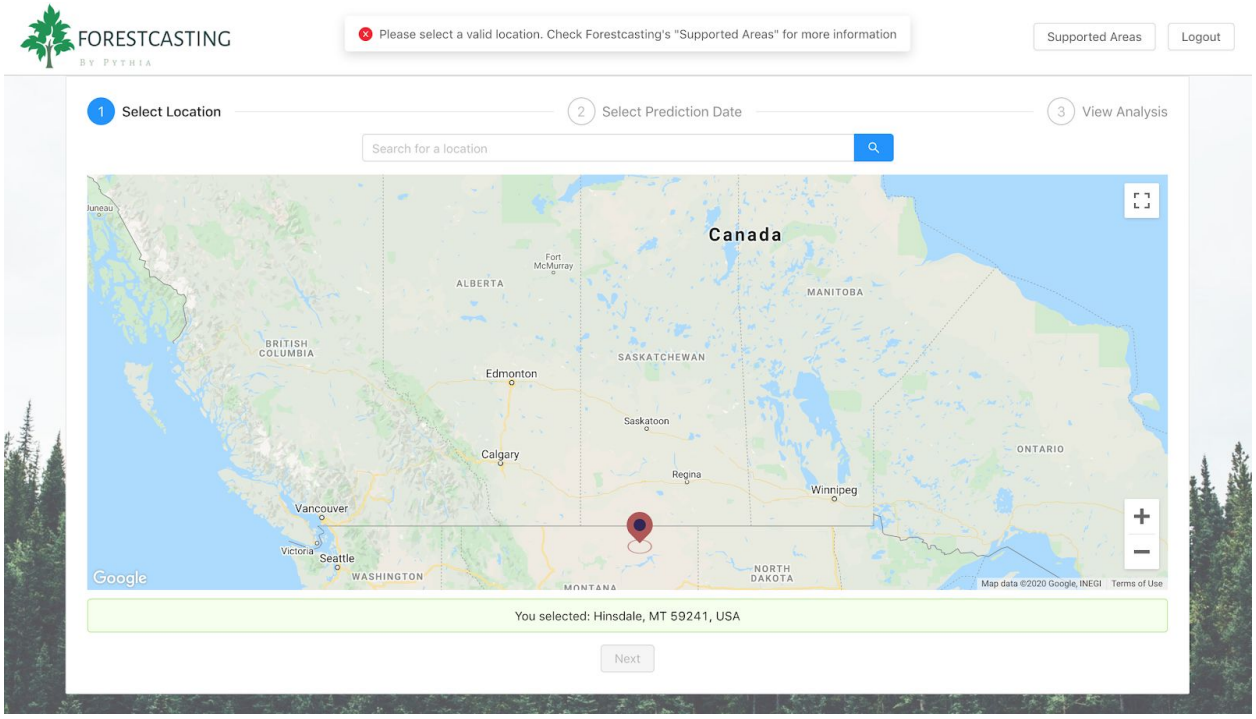
Use the search box to select the desired location:

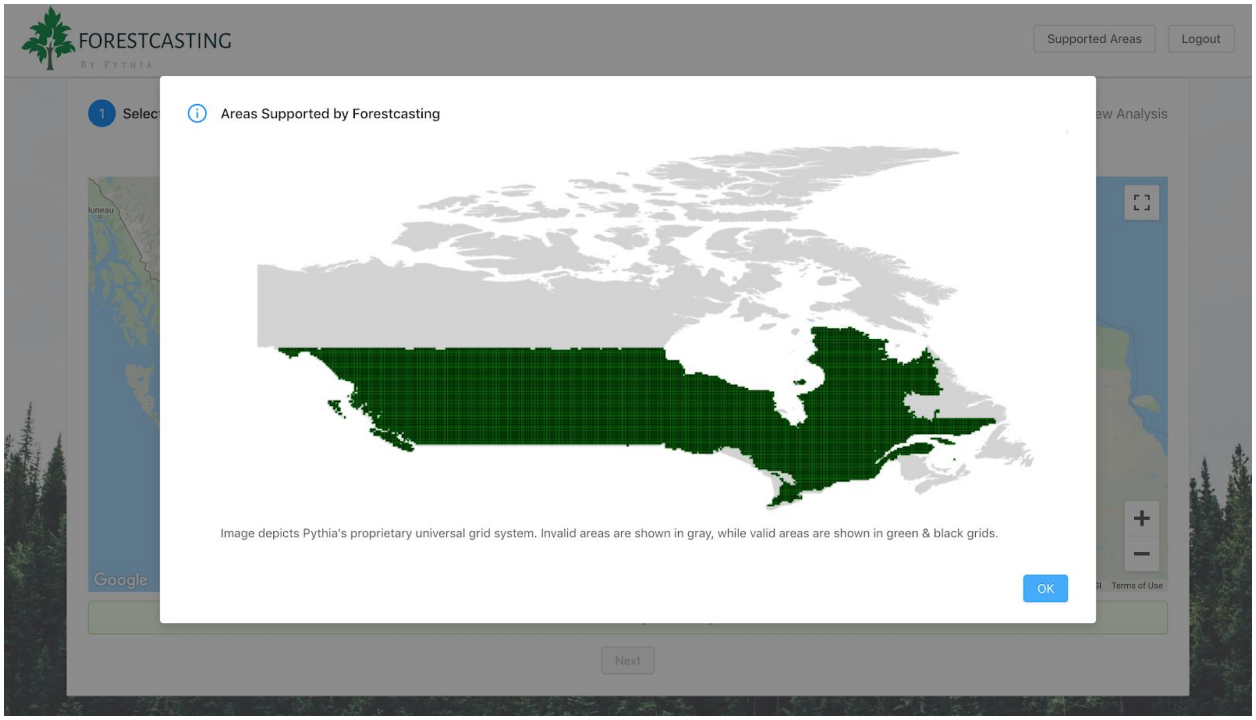
a. Type in the name of a location or an address.
b. Click search to drop a pin and select that location for analysis, followed by Next.



Selecting an Unsupported Location

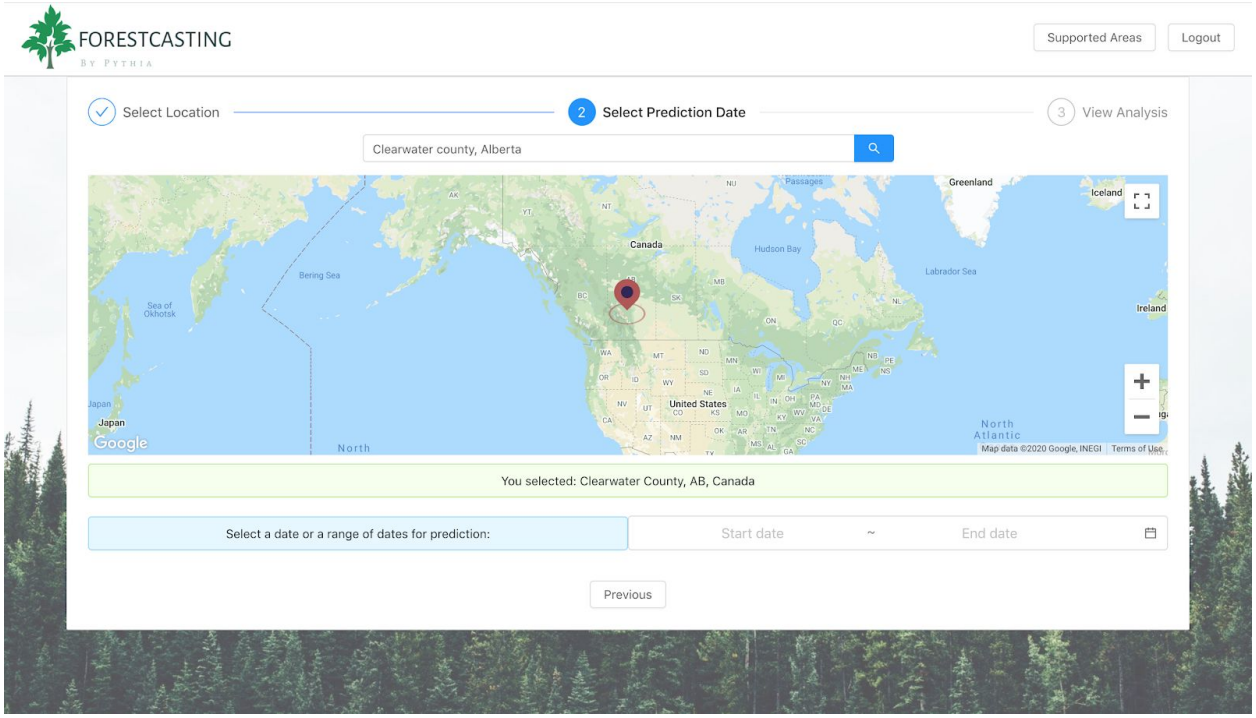
If a selected location is not supported by the *Forestcasting* model, the user will be prompted to check the “Supported Areas” map.



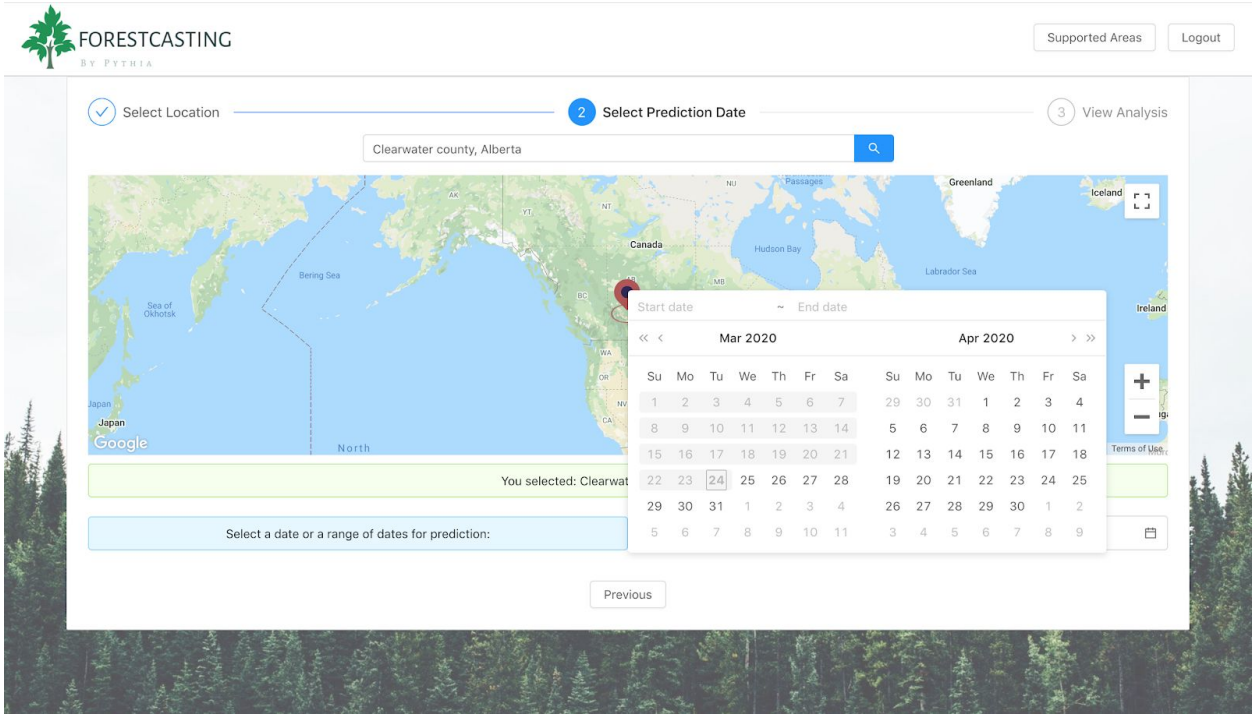


Selecting Prediction Date(s)

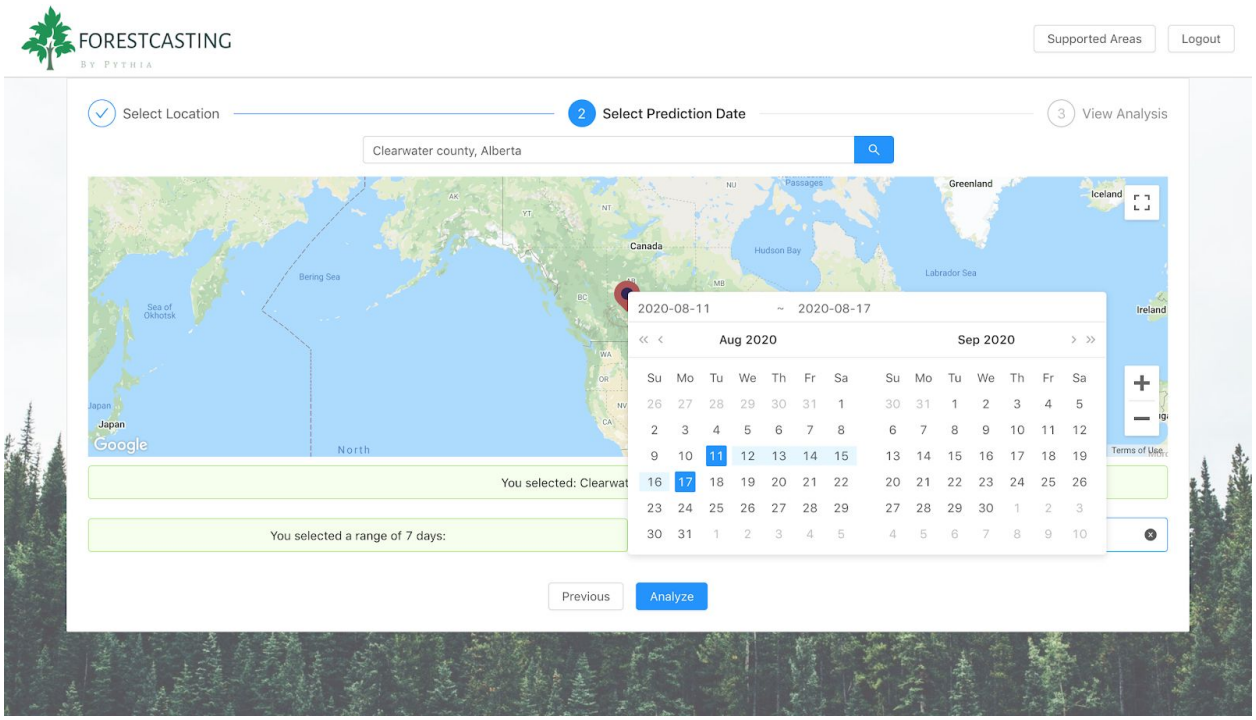
Pressing Next after selecting the location will prompt the selection of prediction date(s).



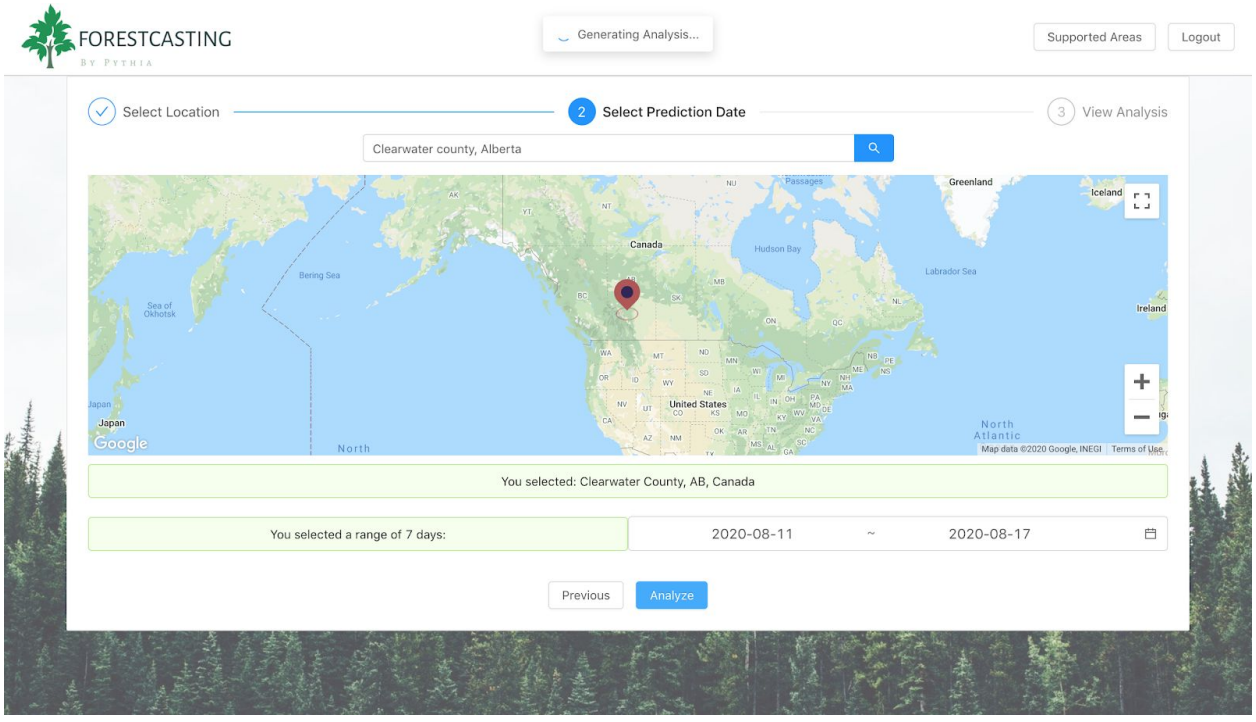
Open the calendar tool to select date(s).



Click on the start date and end dates to highlight a range.

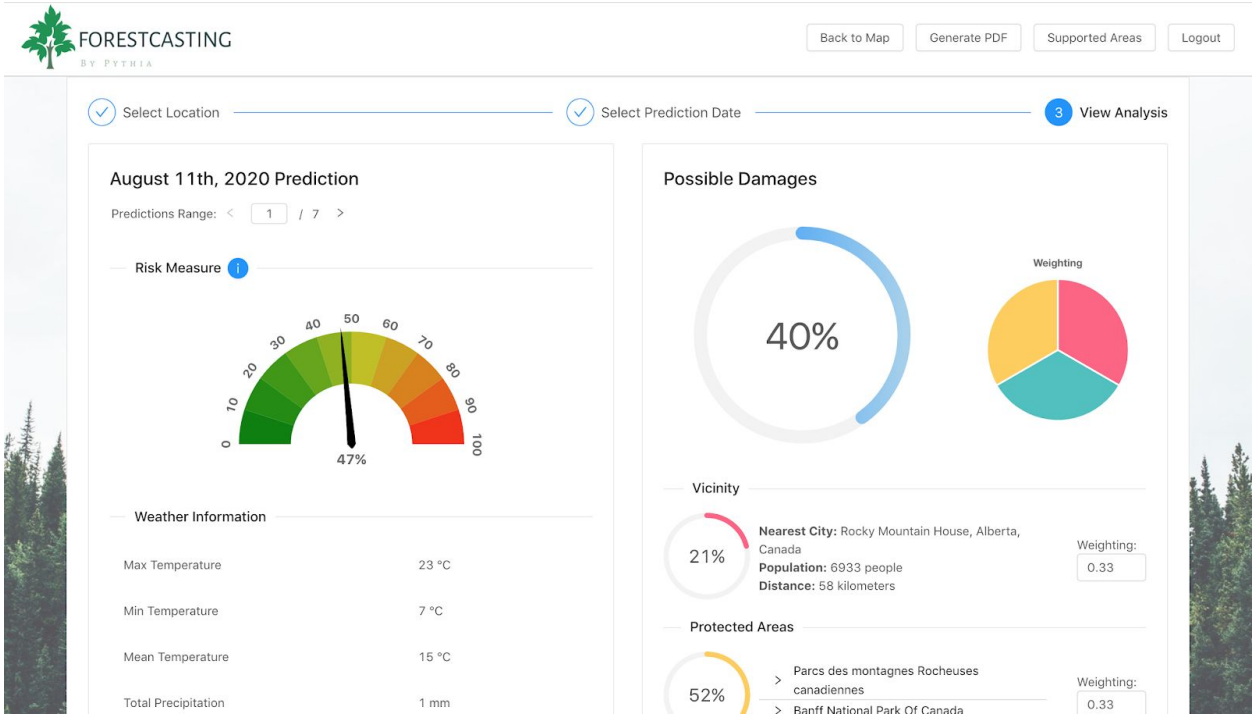


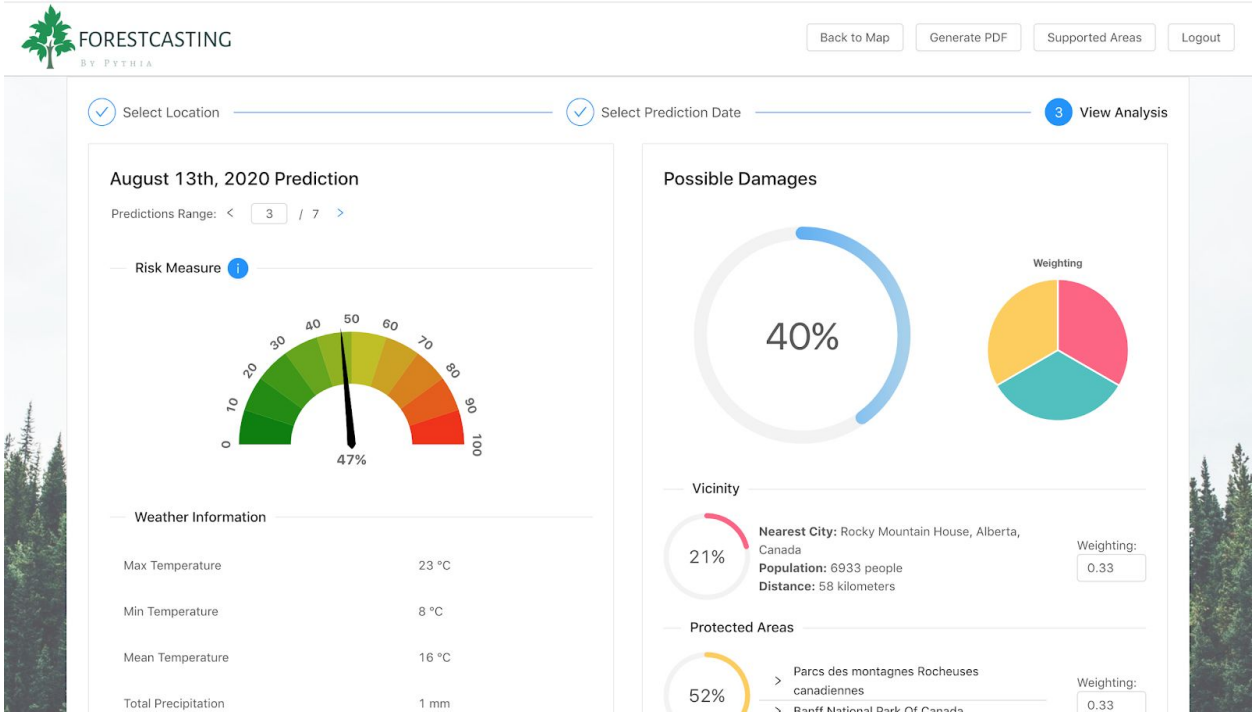
Press Analyze to view the results page.



View Fire Risk Prediction by Date

On the left-hand side, click on the left or right arrows to flip through the selected date range and view the fire risk prediction and weather information for each specific day.





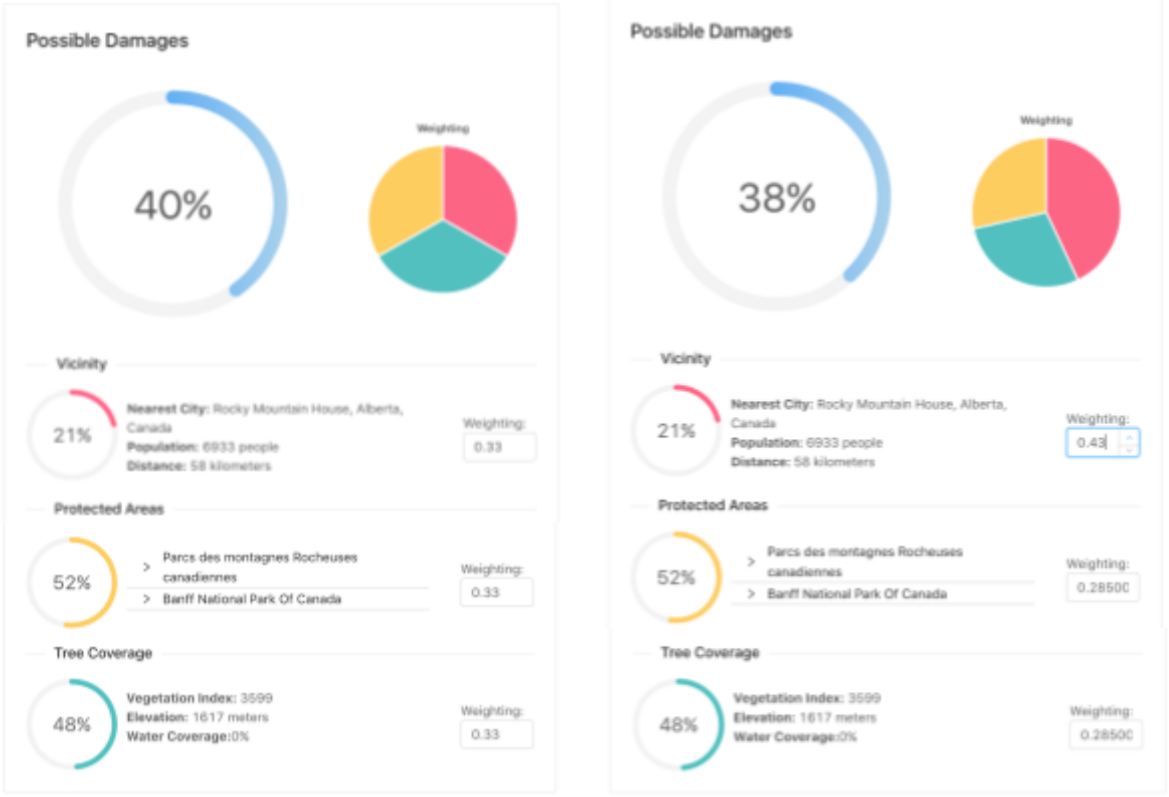
Adjust Possible Damage

On the right-hand side, the pie chart shows the weighting of the factors that went into calculating the Possible Damages.

The default weighting is initially equal for all factors.

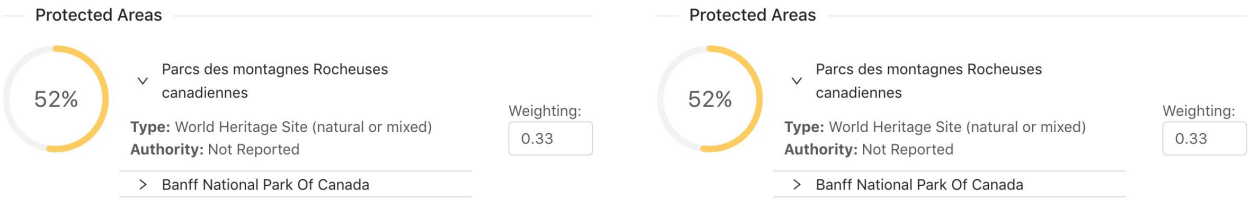
The weighting can be adjusted by increasing or decreasing any of the factor's weighting. The user can also enter the desired weightings in the input boxes.

The other factors' weightings will automatically adjust accordingly.



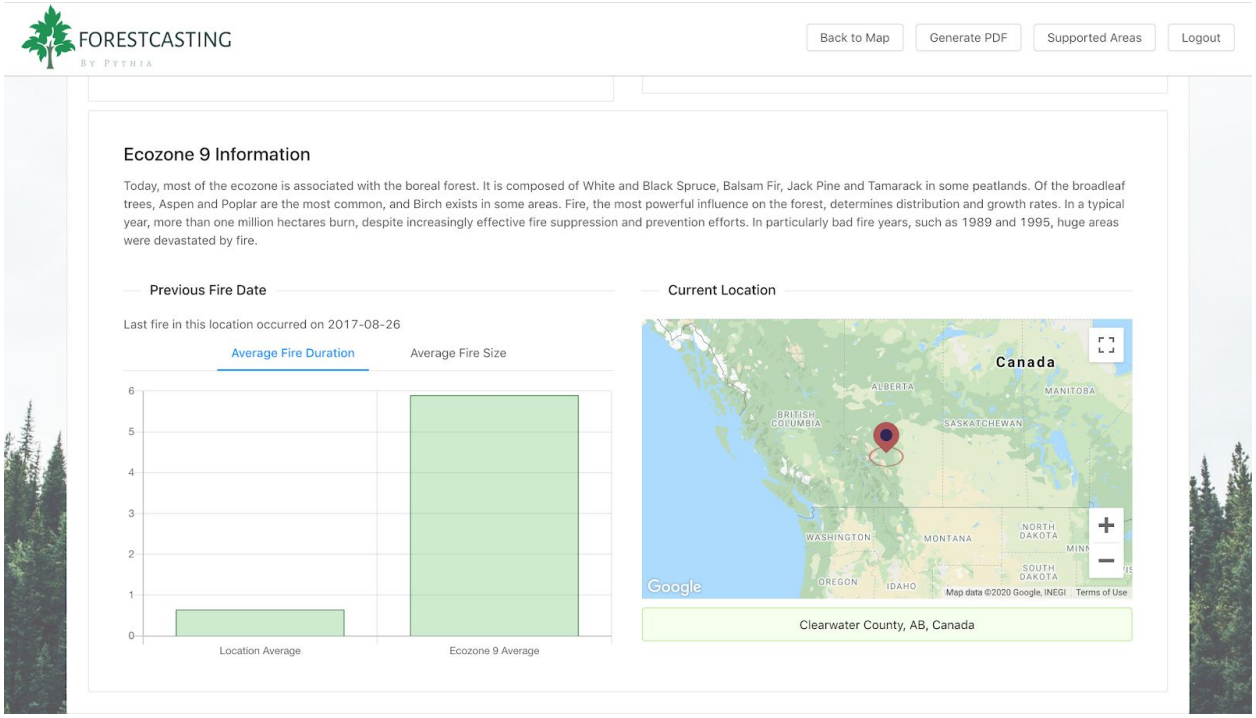
View Protected Area Details

The details of protected areas can be revealed by clicking on each one.

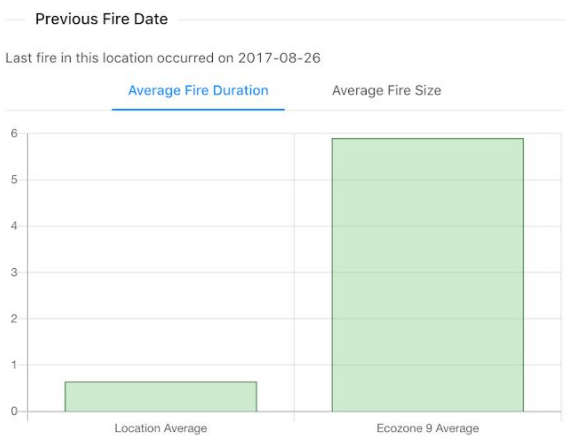


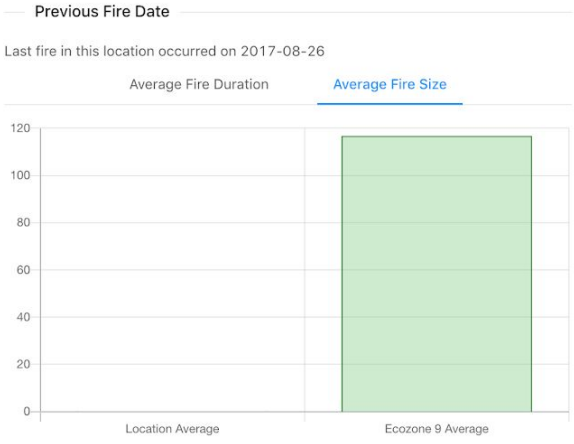
View Ecozone Information

The bottom of the results page includes information about the ecozone of the selected location.



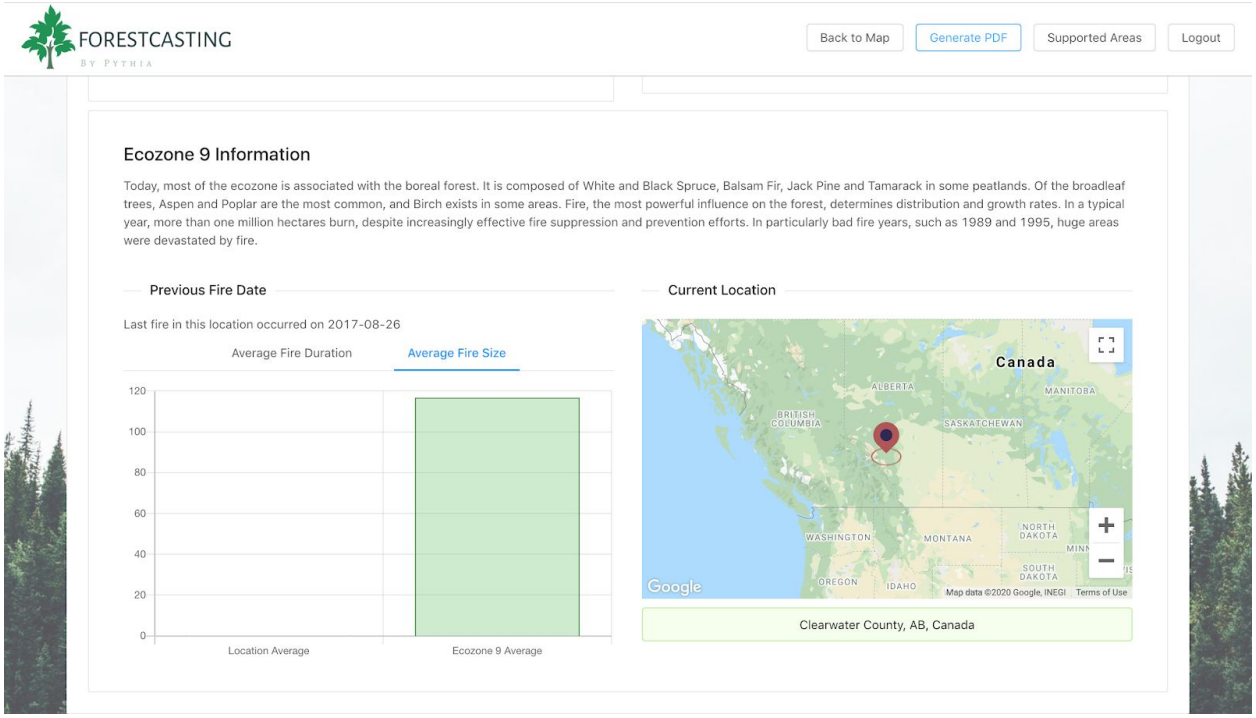
The user can view historical fire data by clicking on the tabs on the left-hand side.





Generate PDF

The “Generate PDF” button on the header will automatically generate a PDF of the results page and download it to the user’s computer.



Appendix

GitHub Repository

Link: https://github.com/ivanzvonkov/forestcasting

References

1. Canadian Wildland Fire Strategy (CWFS). “Background, Synthesis, Analysis, and Perspectives”. *Canadian Council of Forest Ministers,* https://cfs.nrcan.gc.ca/pubwarehouse/pdfs/26529.pdf.

2. Canadian Interagency Forest Fire Centre (CIFFC). “Evaluating Past, Current and Future Forest Fire Load Trends in Canada”. *B.J. Stocks Wildfire Investigations Ltd,* CIFFC, 2019, https://www.ciffc.ca/sites/default/files/2019-03/FireLoadTrends.pdf.

3. “Fire Management.” *Natural Resources Canada*, 28 June 2019, https://www.nrcan.gc.ca/our-natural-resources/forests-forestry/wildland-fires-insects-disturban/forest-fires/fire-management/13157.