General Instructions

1. Provide a cover page that includes the document name, product name, customer name, team name, team member names, and the current date.
2. Number the pages of the document. Include page numbers in the table of contents.
3. Number and label all figures. Each figure or table should have a number, title (brief descriptive name of the figure), and a caption (a longer description of what the reader is supposed to understand from the figure or table). Refer to the figures and tables by number in the text.
4. All sections should have an introductory sentence or two.
5. Do not use vague words and phrases such as may, might, could, possibly, assumed to be, some, a little, and a lot. Use strong, definite words and phrases such as shall, will, will not, can, and cannot. Words such as “should” can be used to show suggestions, but use it sparingly.
6. Watch your spelling, punctuation, and grammar. It is a reflection on your professionalism.

Why did you leave this page in the report?

COS 397: Computer Science Capstone

Blue Marble Geographics
Geospatial Data Portal
System Design Document

Team Undershrub
Anthony Jackson
David Sincyr III
Devin Carter
Grant Shotwell
Steven Kaplan

Nov. 10th 2021

Table of Contents

Document History

Name	Date	Reason for Changes	Version
David Sincyr	10/25/2021	Initial Creation	1.0
All Team Members	11/10/2021	Added in all relevant information	1.1

Very good

1 Introduction

Blue Marble Geographics® is a software development company that specializes in geographic information systems and geodetic desktop software. Their software is aimed towards all types of users who need a better tool to manipulate and visualize their geospatial data. In order to better serve their customers, a cloud based structure is needed. The dynamic web portal will allow its users to upload their geospatial data to an online database. Once in the database, they will have the ability to share, manipulate and visualize their data with others. Additionally, users will be able to automate the visualization process by using python scripts through a built-in python script editor. This is a capstone project for Blue Marble Geographics®, in partial fulfillment of the Computer Science BS degree for the University of Maine.

1.1 Purpose of This Document

The purpose of this System Design Document is to describe how the web portal will be designed. Specifically, it will detail the system at the architecture level, all subsystems and their services, data management systems, the overall software control structure, all hardware mappings, define access controls, and boundary conditions. The intended readership of this document is the project manager, the quality assurance manager, system developers, and system owner. Furthermore, this document is meant for all stakeholders to analyze the web portal prior to deployment to further their understanding of the system's functionalities.

1.2 References

1. System Requirements Specification: SRS_BlueMarbleGeographics_F21.docx
2. User Interface Design Document: Not yet implemented
3. https://fordfiringorder.com/
4. https://armacad.info/international-center-for-agribusiness-research-and-education--2019-09-05--certificate-program-geographic-information-systems-gis-and-remote-sensing-rs-icare
5. https://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/what-is-raster-data.htm
6. System Design Document Template Adapted from Susan Mitchell

2 System Architecture

This section will include a high level overview of the web portal that will detail the overall structure and design as it relates to the hardware and software requirements. It will use unified modeling language in order to better represent the design and provide the documents intended readership an easy to read format.

2.1 Architectural Design

System Architecture

Description

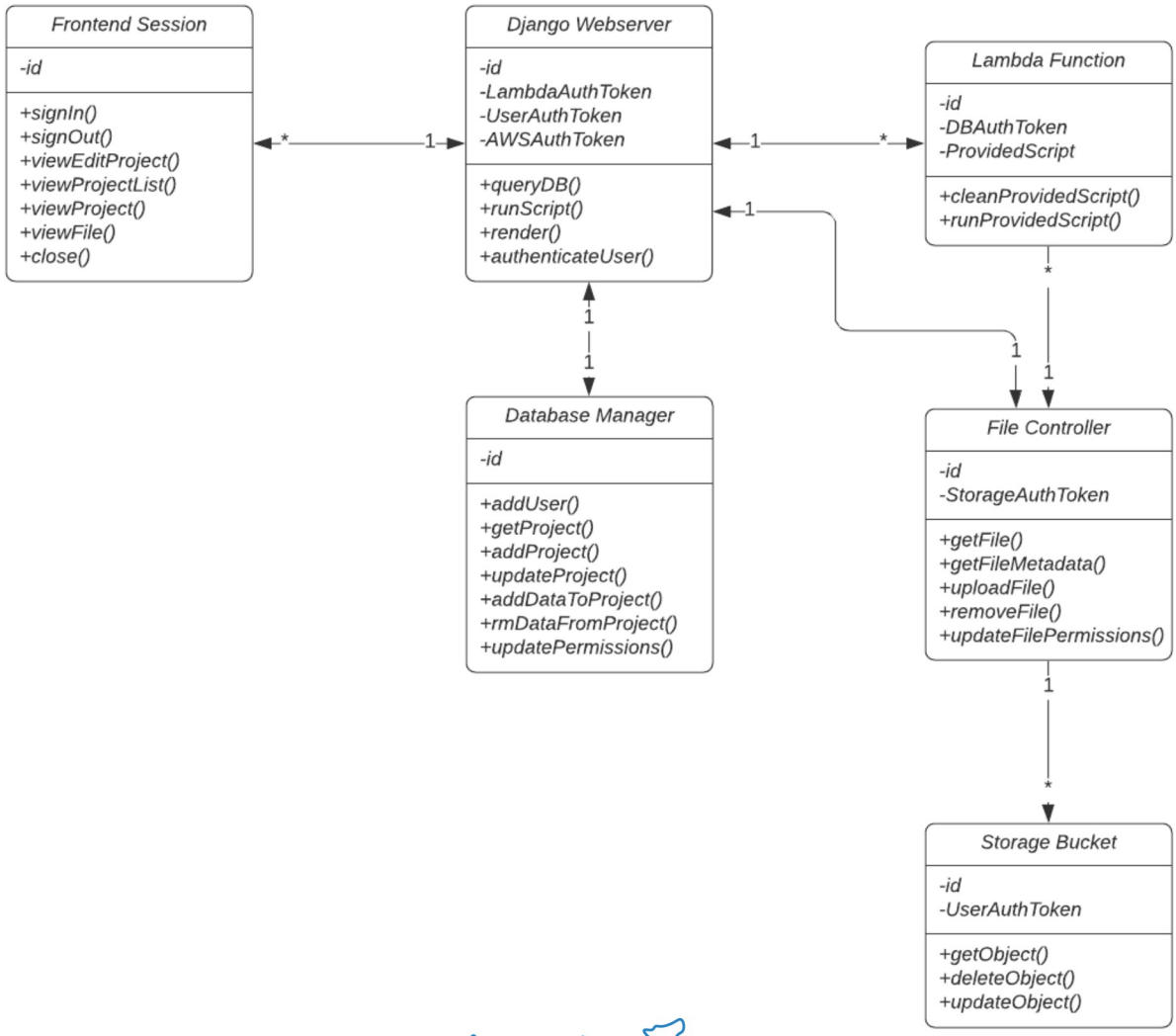
At a high level, the Geospatial Data Portal system consists of a frontend website, a database, file storage, lambda functions, and a Django backend server managing communication between each of these services. From the frontend site, users will be able to create projects, upload data, manage access to their projects, among other tasks. Queries from the frontend session to the backend Django server allow the user's actions to carry out their desired tasks. Django communicates with the Database Manager and the File Controller via authenticated API calls. The Django backend server also invokes lambda functions to carry out tasks specified in user-provided Python code. Lambda functions use API calls to query the File Controller and then return (to the Django server) some response (such as the access tokens for a newly updated dataset). In this process, the File Controller communicates with a file storage system such as Amazon S3 or AWS EFS to execute commands against user-uploaded data.

A UML diagram of the overall system architecture is provided in Figure 1. This high-level diagram highlights the relationship and distinct responsibilities of the frontend, Django backend, and cloud-based backend of the system. Section 2.2 identifies the system's subsystems and their components.

System Architecture UML Diagram

Figure 1: Overall System Architecture

Team Undershrub | November 8, 2021



List of Components

The system architecture will make use of various outside software components to function without a massive drain on the clients own resources, starting with the user's point of view the website will utilize Javascript's React in order to development interactive frontend interfaces which will render the website and allow the user to send modification requests for their data. There will be four of these interactive user interfaces which begins with a "homepage" to view overall information such as login and settings, a "project list view" which would allow users to access levels of their repositories, the "project view" that has access to one of these particular databases and then the "data visualizer" which would allow the user to see in depth image information for a particular part of the project. Each of these four menus will also have various other

functions such as requests to edit the project, change permissions and download/upload data depending on what the user does.

The backend for the system will be handled through the utilization of the Django as the web application framework, allow the major parts of the model view to be handled easily as the AWS database stores and allows the augmentation and upload/download of data without relying on heavy storage costs for the client or user. Then once the user wishes to execute code onto the database information we can provide them secure lambda buckets which can hold the information, allowing augmentation in a safe fashion without the risk of cross contamination of systems. Lambda will also function very well if the user wishes to upload or view images from other sources such as tablets or phones, as it has built in capabilities to resize images to fit the users current screen size. All of this information will of course be packaged into the amazon web hosting services, which will be the hosting format for the website and will provide high powered cloud data storage so the client can access their information from almost anywhere.

The hardware for this web hosted service is extremely minimal as AWS will handle the server management and hosting, the user will only need to already have some sort of computer and internet connection to access our utilities.

2.2 Decomposition Description

This section of the system design document will break down the various subsystems, functions, objects, files, scripts and all major components of the system. The purpose and function of each component will also be detailed.

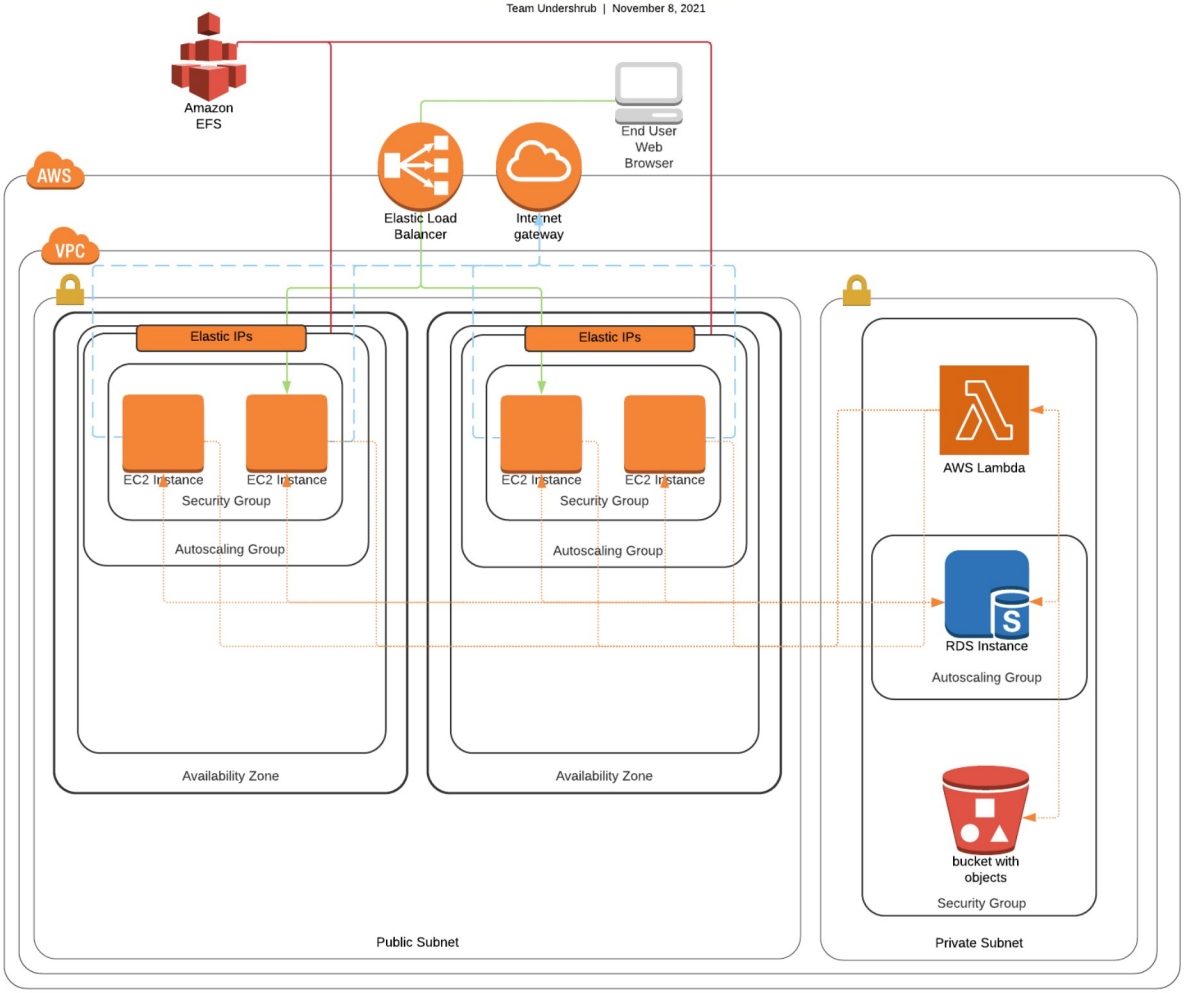
Cloud Architecture

Description

The cloud architecture will incorporate several services provided by Amazon Web Services. The core cloud architecture consists of one or more EC2 instances, a MySQL database hosted on RDS, data stored in S3 buckets and/or EFS, and dynamically generated AWS Lambda instances. The primary auxiliary cloud architecture consists of services such as load balancing which improve the resiliency, reliability, and availability of the system. The secondary auxiliary cloud architecture consists of services necessary for development and continued maintenance of the system. In this document, “cloud architecture” refers to the combination of the core and both auxiliary architectures.

Figure 2: Core Cloud Architecture

Team Undershrub | November 8, 2021



The Primary Auxiliary Cloud Architecture utilizes recommended cloud configurations to improve the performance of the system. An elastic load balancer directs traffic to webservers on different EC2 instances across multiple availability zones such that the load on any single instance is easily manageable. An autoscaling group ensures that the system is responsive to sudden increases in traffic. A VPC with a public and private subnet ensures that stored data (either in S3 or in RDS) is networked separately from the EC2 instances, decreasing the ability for malicious actors to gain unauthorized access. A separate autoscaling group for the RDS instance ensures resilience against high-traffic or large-scale data transfers. This subarchitecture is depicted in Figure 5.

The Secondary Auxiliary Cloud Architecture provides a secure pathway for current developers and future maintainers of the system to connect and interact with an isolated replica of the system. The system utilizes a MySQL database hosted on a private EC2 instance, with access restricted behind a separate bastion instance that enforces strict authentication protocols. All lambda functions and storage containers are secured in a private subnet.

Figure 3: Secondary Auxiliary Cloud Architecture

Team Undershrub | November 8, 2021

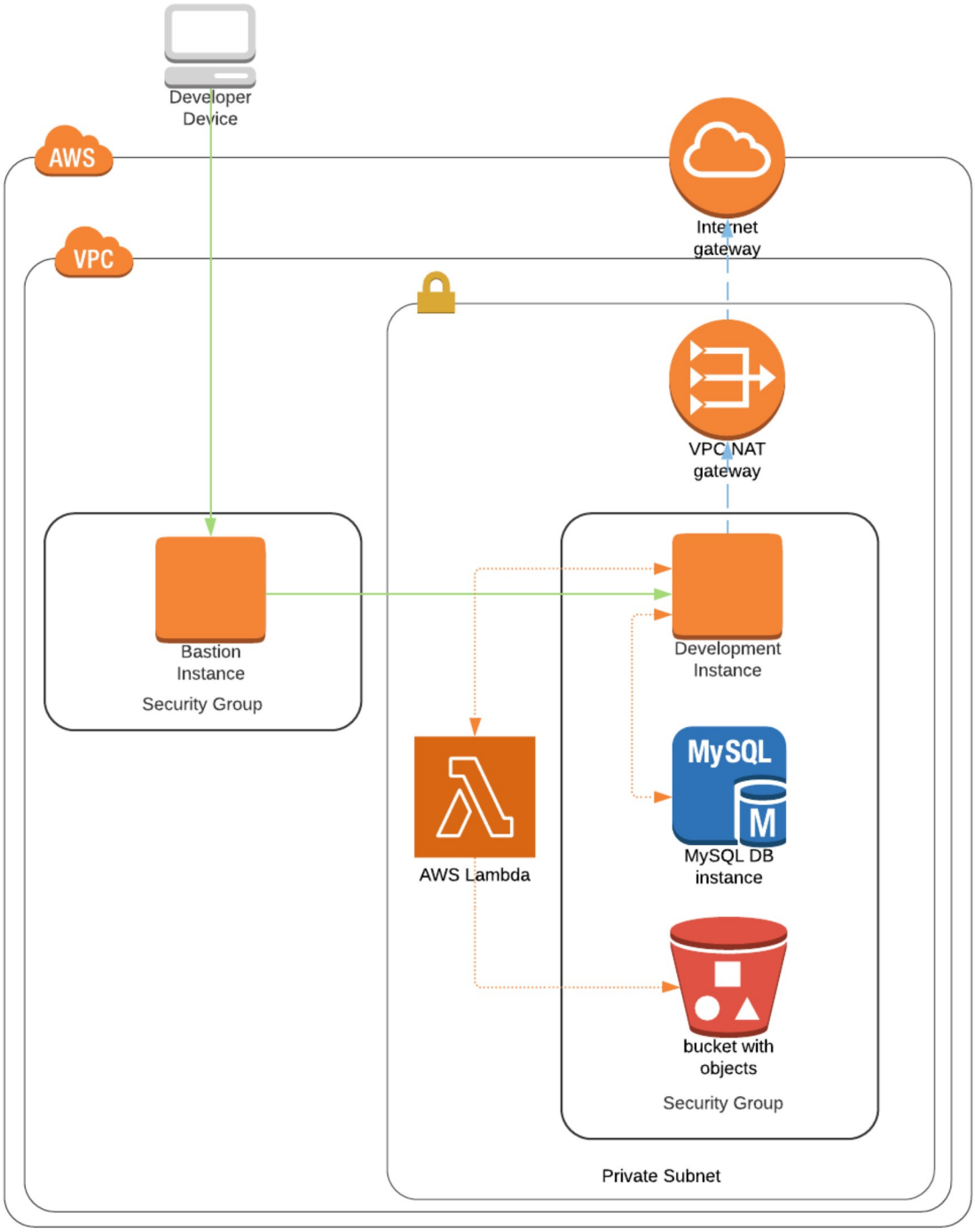
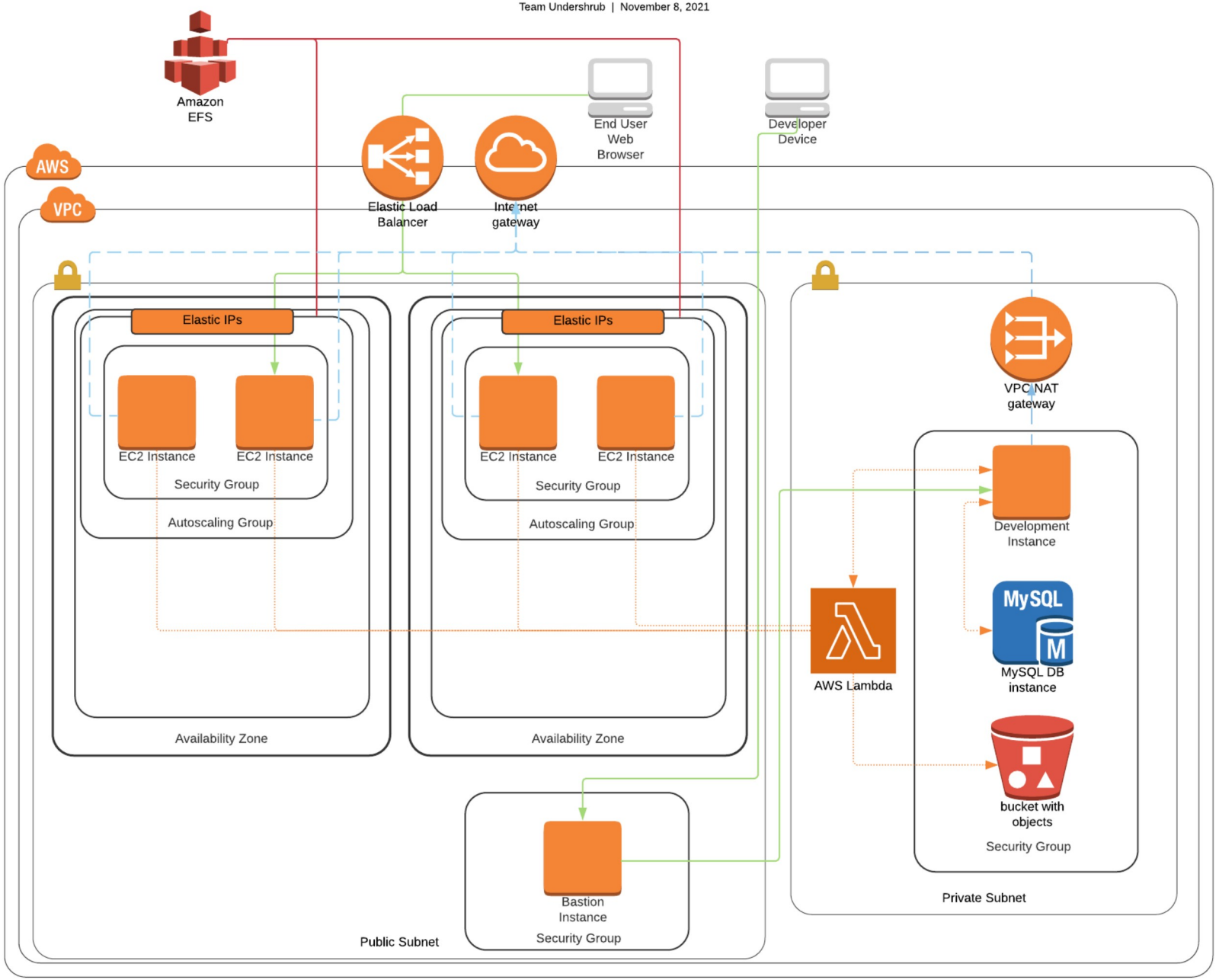


Figure 4: Geospatial Data Portal Cloud Architecture

Team Undershrub | November 8, 2021



List of Components

- Elastic Cloud Computer (EC2)
  - EC2 Autoscaling
  - EC2 Network Load Balancer
- Relational Database Service (RDS)
- Route 53
- AWS Certification Manager (ACM)
- Virtual Private Cloud (VPC)
- Amazon Simple Storage Service (S3)
- Elastic File System (EFS)
- Amazon CloudWatch

Backend Architecture

Django

Description

The backend architecture will use Django and be composed of various controllers.

The Authentication Controller will handle login requests by redirecting the user to their specified login choice, such as Google. Once redirected, the user will be prompted to log into their account. Upon successful login, they will be redirected back to the data portal, which will process the response, grab the URL for the user's homepage and send that view to the user.

Probably should interleaved figures and text.

The File Controller will be responsible for handling the users' file requests which will query the database, set up a view, and send it to the user. Furthermore, it will also handle save requests that will either save a file if not present or update one if the file already exists. Another feature that the file controller will handle will be users' scripts. When a user creates a script, the file controller will query the database, package it up in an Amazon S3 bucket and pass it along to Amazon's Lambda serverless computing service. This will provide users with resources to run their scripts using high-powered machines.

The project controller will handle all requests pertaining to a user's protected system files, such as projects. It will take a user's request, process it, query the database for the selected project, and then provide resources to the user.

List of Components

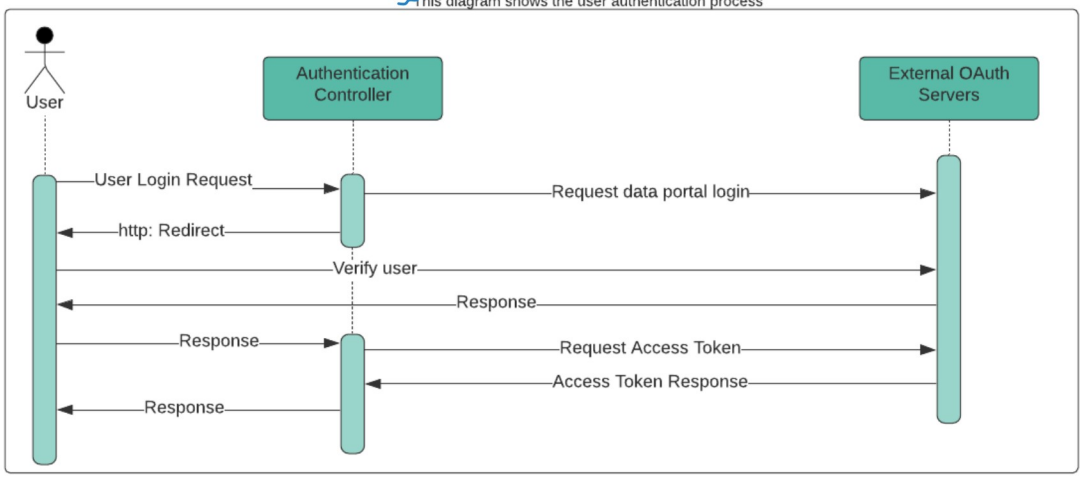
- File systems
- File controller
- Database
- Project Controller
- Web Server
- Authentication Controller

High-level Sequence Diagrams

Authentication Controller → External OAuth → Verify User

Figure 2: Authentication Sequence Diagram

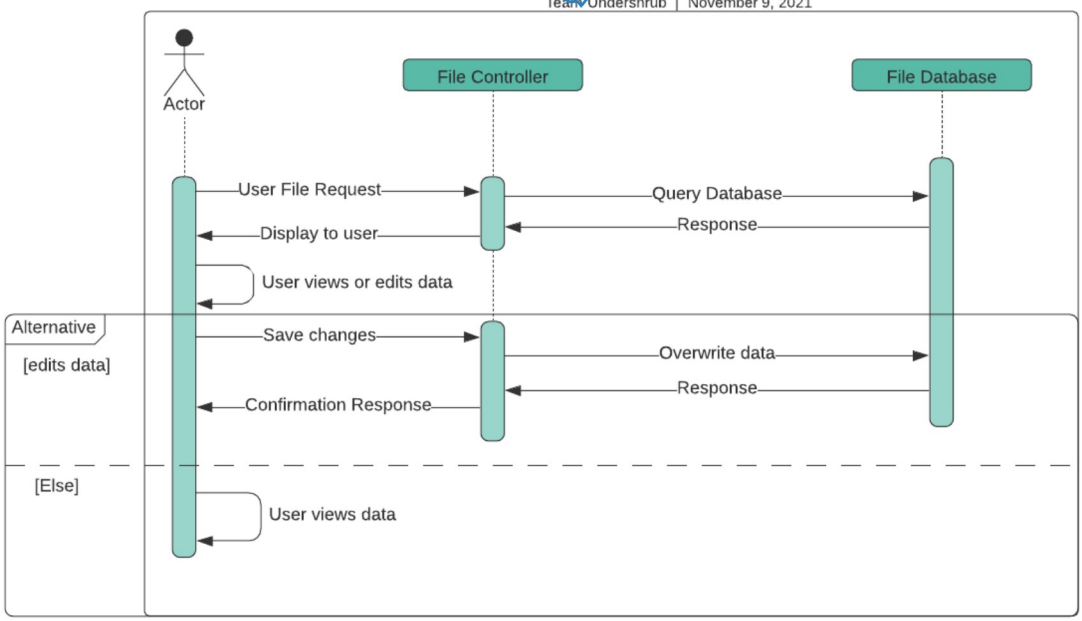
This diagram shows the user authentication process



File Controller → Query Database → Provide file to user →save data to database

Figure 3: File Sequence Diagram

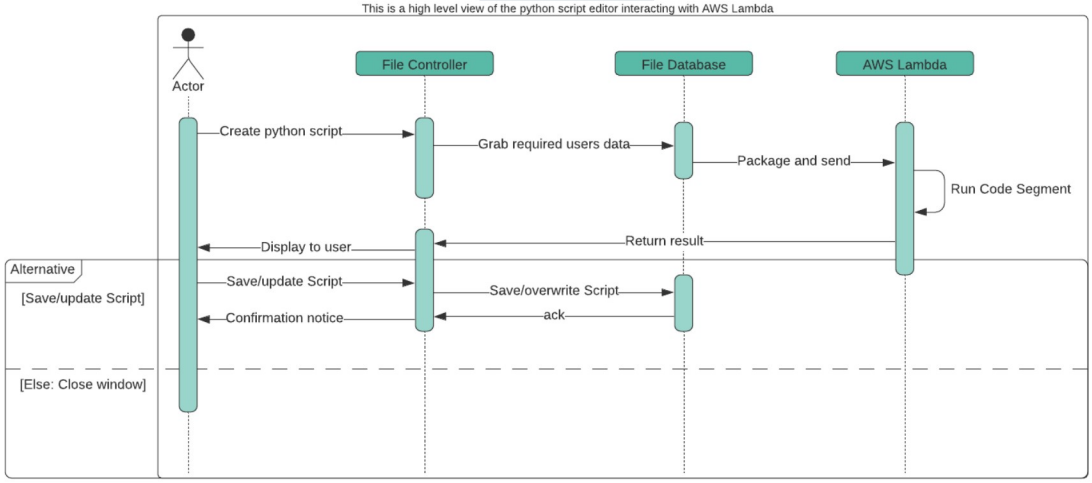
Team Undershrub | November 9, 2021



File Controller → Query Database → Query AWS Lambda → Provide resources to user

Figure 4: Python Script Sequence Diagram

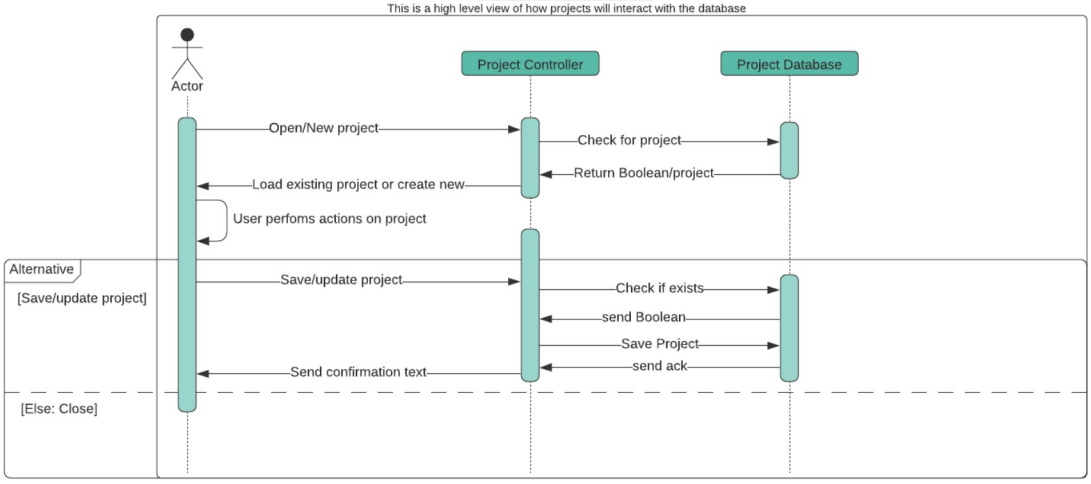
This is a high level view of the python script editor interacting with AWS Lambda



Project Controller → Query Database → provide resources to user

Figure 5: Project Sequence Diagram

This is a high level view of how projects will interact with the database



Frontend Architecture

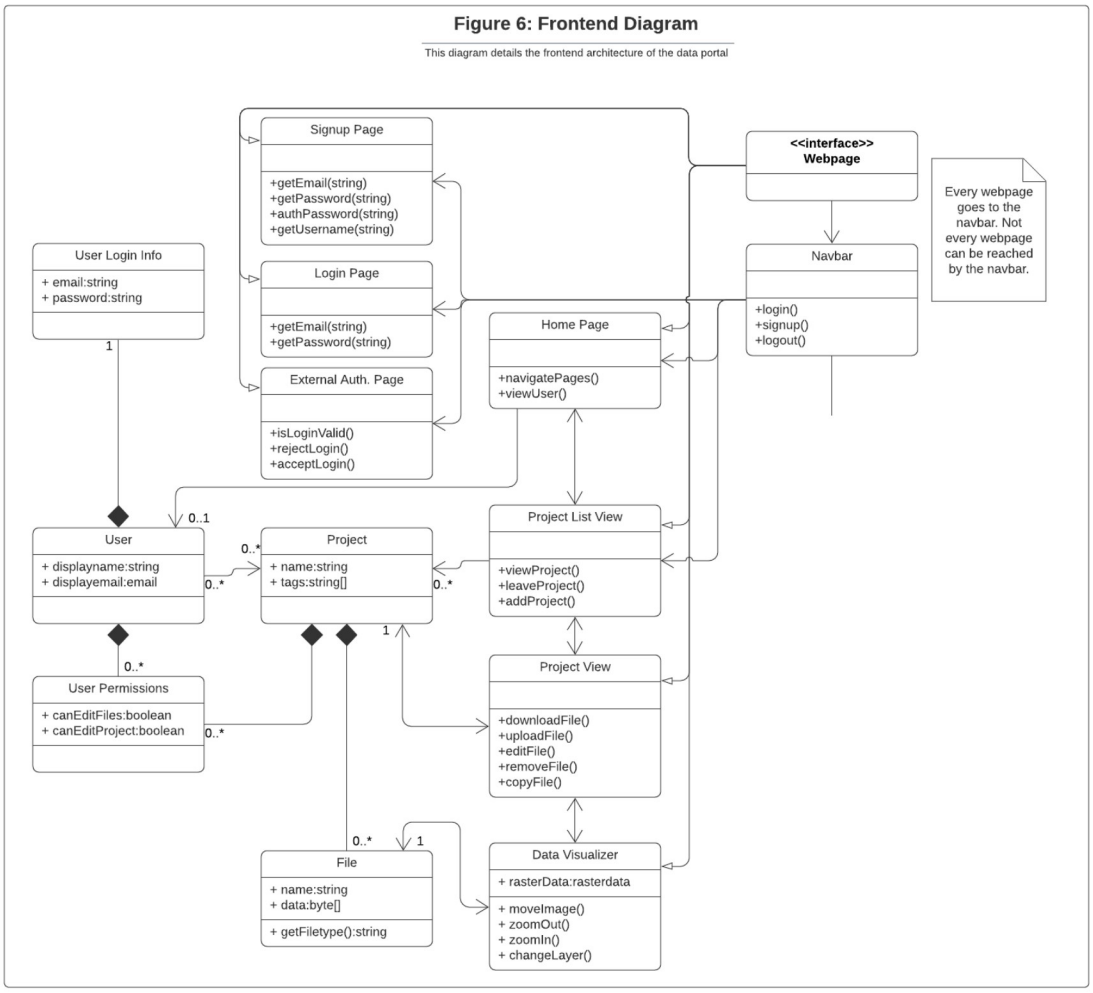
Description

The frontend architecture diagram gives a description of the system's web interface and design, detailing how users interact with the system's key components. The first component of the web interface is the homepage which

contains software information, account accessibility, introductions, links to the navbar and the project list. Once you navigate off the homepage and either have a guest or user account you can access the project view which shows all related project media the user has access to, alongside allowing the user to edit or select a project if they have the right credentials. The project is viewed within the project view and allows the display of any file the user wishes to access and the directory of the structure. Within the project view is the data visualizer which can be opened to view particular information on the project such as the layering, images, or even specific documents uploaded to the project.

The frontend will also work to create “user permission” objects to be associated with specific actions in a “user project” pair structure which can associate certain permissions which any user with that object gains access to. Within the diagram the “user login info” is only designed to exist for the signup/login process pages and won’t be used outside of them, while the “username” and “user” objects will exist along projects and the homepage to identify the user. There will be a simple (1 - 0) boolean which can represent a signal if the user is logged in, once the project notices a user is logged in it can then retrieve the specific information associated with the user. Notably; this system will allow multiple projects to exist alongside a single user and multiple users to exist for a single project, being viewable in the “project list view” to see which clients have access to what information.

UML Diagram



List of Components.

- ➢ <<interface>> Webpage
- ➢ Home Pages: Webpage
- ➢ Project List View: Webpage
- ➢ Project View: Webpage
- ➢ Data Visualizer: Webpage
- ➢ Navbar: Webpage
- ➢ Signup Page: Webpage
- ➢ Login Page: Webpage
- ➢ External Authentication Page: Webpage
- ➢ Project
- ➢ File
- ➢ User
- ➢ User Permissions
- ➢ User Login Info

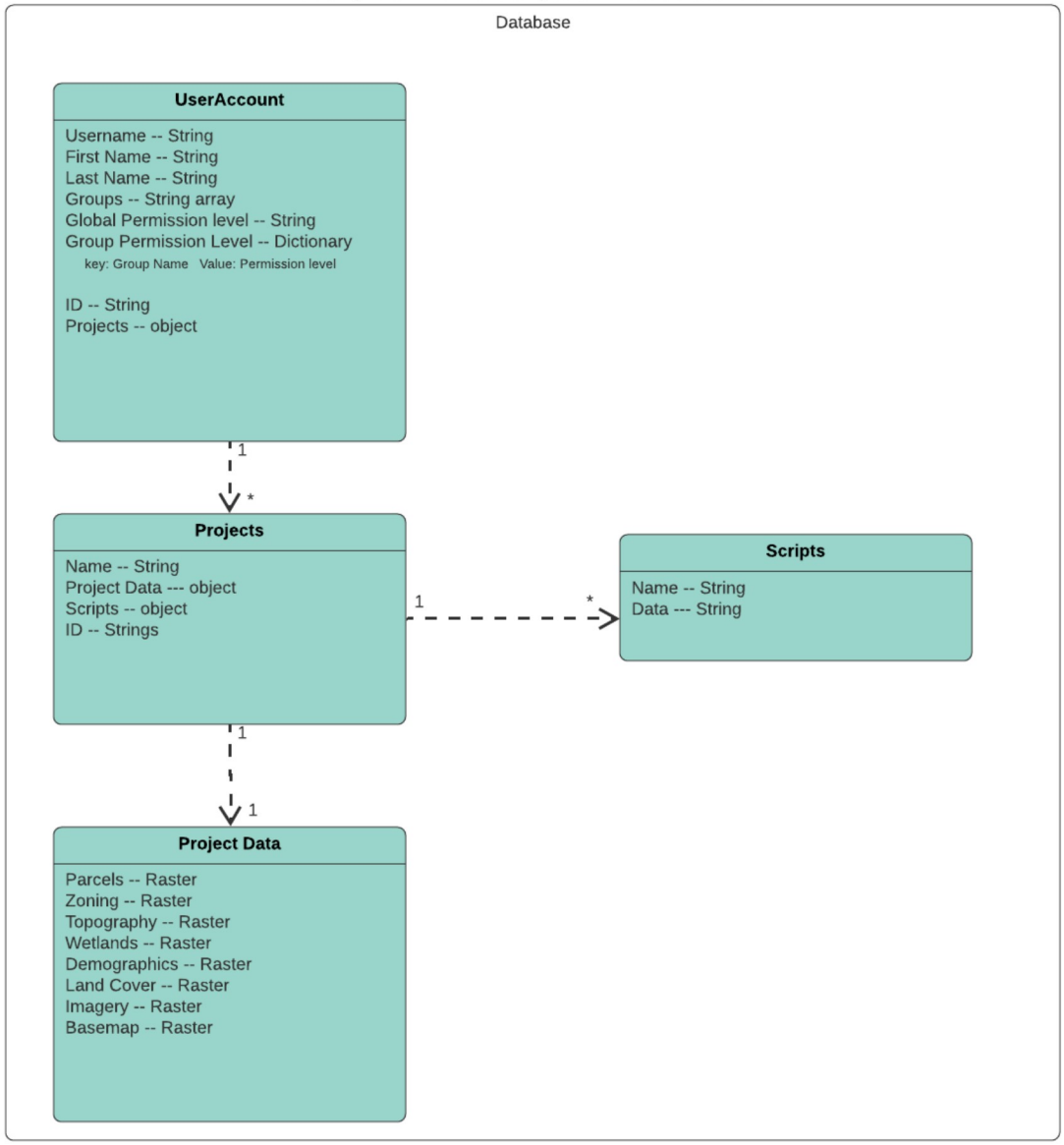
3 Persistent Data Design

This section will detail a high level view of the database's data architecture. A MySQL instance of AWS RDS is the database that will be primarily used to store information about the customer and their data. The database description section (3.1) will be an entity relationship diagram that will display what is being stored, the data type and what it represents. The following section (3.2) will be the File description section that will detail the file structure and its contents to include the name, data types, and representation descriptions.

3.1 Database Descriptions

Figure 7: Database Diagram

This is a high level view of what is being stored on the database as metadata



UserAccount will store a user's information which includes private and public data, the description of this data is detailed below.

- ➢ **Username:** This will be the user's display name.
- ➢ **First Name:** This is the user's first name.
- ➢ **Last Name:** This is the user's last name.
- ➢ **Group Name:** This is the current group(s) the user is a part of.
- ➢ **Global Permission Level:** The current permission level of the user that defines access controls
- ➢ **Group Permission Levels:** The current permission level of the user within each group the user is a member of.
- ➢ **Projects:** A current list of projects the user owns or is a member of.

Projects will store a user's current projects to include the name, its data (explained below) and any scripts attached to them.

Project Data will contain various layers of raster data that belongs to a user. The layers are as follows:

- **Parcels:** The data stored here will be property and tax map data.
- **Zoning:** This will allow a user to store zoning data on an area.
- **Topography:** This will allow a user to store data representing the area's topography.
- **Wetlands:** This allows the user to store wetland data.
- **Demographics:** This will allow a user to store maps pertaining to an area's demographics.
- **Land Cover:** This allows the user to store an area's vegetation, soil or urban environments.
- **Imagery:** This layer will allow the user to store an overhead view image of an area.
- **Basemap:** The base map layer will allow users to combine all other layers the user selects to create a custom map.a

Scripts will be stored by their name as a string and the data will be a string that will be the source code.

3.2 File Descriptions

This section will break down the file structure in terms of fields. Each field will be assigned a name, it’s data type, size, and a small description of what it will represent.

UserAccount

- An object for storing information about a user within a database.
- Username: string
- FirstName: string
- LastName: string
- Groups: string[]

- GlobalPermissionLevel: string
- GroupPermissionLevels: Dictionary<Key: string, Value: string>
- ID: string
- Projects: string[]

Projects
- An object for storing information about a project within a database.
- Name: string
- Data: ProjectData
- Scripts: Script[]

Script
- An object for storing information about a script within a database.
- Name: string
- Data: string //Parsed as Python code.

ProjectData
- An object for storing information about project data within a database.
- Parcells: byte[] //RasterData
- Zoning: byte[] //RasterData
- Topography: byte[] //RasterData
- Wetlands: byte[] //RasterData
- Demographics: byte[] //RasterData
- LandCover: byte[] //RasterData
- Imagery: byte[] //RasterData
- Basemap: byte[] //RasterData

Authentication Controller.
- Controller for authentication of users.
- UserLoginRequest(UserLoginInformation): Nullable<httpRedirect>

File Controller.
- Controller for accessing/modifying files by users.
- UserGetFileRequest(File): databaseResponse
- UserSaveFileRequest(File): databaseResponse

File Database.
- Object representing a database for storing files.
- DatabaseQuery(File): databaseResponse
- DatabaseWrite(File): databaseResponse

Project Controller.
- Controller for managing projects by users.
- OpenProject(Project): databaseResponse
- SaveProject(Project): boolean

Project Database
- Object representing a database for storing projects.
- DoesProjectExist(Project): boolean
- SaveProject(Project): boolean

<<interface>> Webpage.
- Interface to represent any webpage.

Navbar.
- A section on every webpage to help navigate to other pages.

Home Page: Webpage.
- The default page for the website.
- NavigatePages()
- ViewUser()
- Login()
- Signup()
- Logout()

Signup Page: Webpage.
- A page to create a new account.
- GetEmail(string)
- GetPassword(string)
- AuthPassword(string)
- GetUsername(string)

Login Page: Webpage.
- A page to login to an existing account.
- GetEmail(string)
- GetPassword(string)

External Authentication Page: Webpage.
- A web page to login or signup via an external authentication service.
- IsLoginValid(UserLoginInfo): boolean
- RejectLogin()
- AcceptLogin()

Project List View: Webpage.
- A web page to view all projects assigned to the user.
- ViewProject(Project)
- LeaveProject(Project)
- AddProject(Project)

Project View: Webpage.
- A web page to view a single project and its files.
- DownloadFile(File)

- UploadFile(File)
- EditFile(File, byte[])
- RemoveFile(File)
- CopyFile(File)

Data Visualizer: Webpage.
- A web page to visualize a single file's data.
- RasterData: raster_data

User Login Information
- An object to hold the information required to log in a user.
- Email: string
- Password: string

User.
- An object to represent a user.
- DisplayName: string
- DisplayEmail: string

User Permissions.
- An object to represent allowed permissions for a user.
- CanEditFiles: boolean
- CanEditProject: boolean

Project.
- An object to represent a project.
- Name: string
- Tags: string[]

File.
- An object to represent a file.
- Name: string
- data: byte[]
- getFiletype(): string

4 Requirements Matrix

See the external requirements matrix. Why is this a hyperlink?

Appendix A – Agreement Between Customer and Contractor

What is being agreed to when this document is signed.

Upon signing this document, the signer agrees to the terms and conditions presented in Appendix A and Appendix B, with any future amendments requiring an updated signature. The signer also agrees to provide assistance with the creation and any tasks related to the Geospatial Data Portal as agreed by the signer and either Team Undershrub or Blue Marble Geographic®. Additionally, the signer agrees to do everything needed to ensure that this agreement is upheld in good faith, alongside agreeing to treat any designed or developed material as “work made for hire” for Blue Marble Geographic®. Except as otherwise stated within this agreement, the signer will have full control over working time, methods, and decisions related to the project until either the Blue Marble Geographic® or Team Undershrub deem to end the arrangement. However, the signer will be responsive to the reasonable needs and concerns from either aforementioned group throughout the duration of the arrangement.

Procedures to be used for future changes to this document.

When a member of Team Undershrub or an employee of Blue Marble Geographic® wishes to commit any changes to this document, they must first address them in a comment underneath their signature below, then must alert either the SDD document manager or the Geospatial Data Portal project lead. The alerted individual will notify all signing parties. Subsequently, all signing parties will discuss via email or a meeting the proposed changes to the document. Implementation of proposed changes requires support from the Geospatial Data Portal project lead along with a majority of the development team. These procedures shall be followed for any future additions, deletions or modifications of material within this document.

Name:	**Date:**
Signature:	
Comments:	

Appendix B – Team Review Sign-off

Within this section of the SDD document you will find the signatures, names and date of the documental approvals for each acting member of “Team Undershrub”. These signatures acknowledge and authenticate the approval and review of this document. This authentication includes the overall content, formatting, identification of contributed material and development directions presented within. These signatures must be updated alongside the date and comments when there is a subsational update to the document, which in this case includes any formatting, editing, grammar and change of materials within.

Name: Anthony Jackson	**Date:** 11 - 7 - 21
Signature: 	
Comments:	

Name: David Sincyr III	**Date:** 11 - 8 - 21
Signature: 	
Comments: Signed Digitally	

Name: Devin Carter	**Date:** 11 - 8 - 21
Signature: 	
Comments:	

Name: Stephen Kaplan	**Date:** 11 - 8 - 21
Signature: 	
Comments:	

Name: Grant Shotwell	**Date:** 11 - 8 - 21
Signature: 	
Comments:	

Appendix C – Document Contributions

Identify how each member contributed to the creation of this document. Include what sections each member worked on and an estimate of the percentage of work they contributed. Remember that each team member must contribute to the writing (includes diagrams) for each document produced.

Percentages please.

Grant Shotwell	- Formatting - Frontend Architecture - File Descriptions
Stephen Kaplan	- Assisted in the designing of all systems - Created the Trello board
Devin Carter	
David Sincyr III	- Migrated over portions of the SRS - Wrote the introduction section - Setup the title page - Designed the sequence diagrams - Wrote the persistent data introduction section - Created the Database entity relationship diagram and wrote their descriptions - Wrote the backend description - formatted 3.1 & 3.2
Anthony Jackson	- Wrote appendix B's original formatting and SDD authentication. - Overviewed sequence diagrams. - wrote the architectural design description. - assisted with the front-end diagram by filling in functions. - rewrote the frontend architecture description.