Proctor Creek Watershed Web App Design

JIA Team 1108

Prepared By:

Ryan Bauer	rbauer32@gatech.edu
Grayson Raulston	craulston3@gatech.edu
Dylan Small	dsmall6@gatech.edu
Joanna Tolentino	jtolentino6@gatech.edu
Chiche Tsai	ctsai84@gatech.edu

Prepared for: Ms. Donna Stephens

Version 2.1

Last Edit: 11/282021

https://github.com/dylan-small/Proctor-Creek-Capstone-Project

Table of Contents

Introduction

Background

Illegal dumping, poor drainage systems, and large amounts of impervious cover in the area surrounding Proctor Creek has produced high levels of toxicity in the waters, damaging both the natural ecosystem and the communities situated in the watershed. The purpose of this project is to create a web application with three key goals:

1. Provide residents a place to report environmental issues and stay updated with the status of the watershed.
2. Provide a central site to learn about the environmental impacts of illegal dumping, soil erosion, and sewage leakages in the water.
3. Provide the Proctor Creek Stewardship Council a system to forward reports to federal organizations with which they have partnered.

This web application is designed to be easily accessible on both mobile devices and desktops to fit the needs of our intended users, the residents of the Proctor Creek Watershed. The application will consist of a database to store user reports, an interface for users to submit reports, and administrators access to manage the reports.

Document Summary

This report documents the Proctor Creek Stewardship Council Website in four main parts.

The System Architecture section describes the static and dynamic elements of our systems (React, Django, Django Rest, and Axios elements) and how they interact with each other.

The Data Storage Design section describes the database structure of our system, which consists of Django's built in ORM. It stores users' report data and administrator log in data. In this section, we additionally provide an entity relationship diagram for our database in addition to how our database communicates with the rest of our backend.

The Component Detail Design section provides details of the components of our front end and back end, as well as how the two components interact with each other. This includes the relationships between our models, views, and API.

The UI Design section presents the current state of the major User Interface screens that Proctor Creek residents will use to file reports through screenshots. This section also explains the reason behind our design decisions.

Terminology

The following dictionary is a collection of the terminology and corresponding definition that this document uses. This dictionary includes both technical and domain terminology. Technical terminology pertains to the inner workings of our application, such as the languages we use. Domain terminology relates to our problem space such as the definition of a watershed.

A	**API**	Application Programming Interface. In contrast to a user interface, which connects a computer to a person, API is a type of software interface that offers a service to other pieces of software.
	Axios	Axios is an HTTP client library that allows you to make requests to a given endpoint.
B	**Backend**	Backend is the server-side of the website. Its tasks include storing and querying our database, writing APIs, creating libraries, and working with system components without user interfaces.
	Bootstrap	Bootstrap is a free and open-source potent front-end framework for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.
C	**Class Diagram**	A class diagram is a static UML communication diagram that details the system’s classes, interfaces, and operations between objects.
	CSS	Cascading Style Sheets. It describes the presentation of a document written in a markup language such as HTML.
	CORS	Cross-Origin Resource Sharing. An HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading of resources.
D	**Django**	Python-based free and open-source web framework that follows the model–template–views architectural pattern.
	DOM	Document Object Model. A cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. DOM methods allow programmatic access to the tree; with this tree, one can change the structure, style, or content of a document.
	Domain Name	An identification string that defines a realm of administrative autonomy, authority, or control within the Internet.
	DNS	Domain Name System. DNS translates domain names to IP addresses so browsers can load Internet resources.
E	**ERD**	Entity Relationship Diagram. An ERD is a diagram representing the relationships between database tables, such as many to many, one to one, and one to many.
F	**Favicon**	It is a file containing one or more small icons, associated with a particular website or web page. It is also known as shortcut icon, website icon, tab icon, URL icon, or bookmark icon.
	Frontend	The frontend is the graphical user interface of a website, including HTML, CSS, and JavaScript, with which users can view and interact.
G	**Git**	Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

	GitHub	GitHub is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git.
H	**HTML**	Hypertext Markup Language. It is the standard markup language for documents designed to be displayed in a web browser.
	HTTP	Hypertext Transfer Protocol. HTTP is a protocol for transmitting media, like HMTL, between web browsers and servers.
	HTTPS	Hypertext Transfer Protocol Secure. HTTPS is an extension of HTTP with increased security and added encryption.
J	**JavaScript**	JavaScript is a high-level programming language which is compiled just-in-time, and multi-paradigm that conforms to the ECMAScript specification.
	JWT	JSON Web Token. JWT is a proposed Internet standard for creating data with an optional signature and/or encryption whose payload holds JSON that asserts claims. The tokens are signed either using a private secret or a public/private key.
L	**Load Balancing**	Load balancing acts like a "traffic cop," sitting in front of your backend servers and distributing client requests across a group of servers in a manner that maximizes speed and capacity utilization while ensuring no server is overloaded, which would degrade performance.
M	**Material-UI**	Material-UI, which is designed and developed by Google, is a library that allows us to import and use different components to create a user interface in our React applications.
N	**Node.js**	Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes.
	NPM	Node Package Manager. NPM is the default package manager for the JavaScript runtime environment Node.js.
O	**ORM**	Object Relational Mapping. ORM maps backend code into SQL to create, read, update, and delete objects through backend code.
R	**React.js**	React.js is an open-source front-end framework maintained by Facebook. It is a declarative, efficient, flexible JavaScript and component-based library for building user interfaces.
	Reverse Proxy	A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server.
	Robustness Diagram	A robustness diagram is a UML collaboration/communication diagram used to represent the inner workings of a system through high-level symbols (actor, entity, process, control).
P	**PII**	Personal Identifiable Information. According to the U.S. Department of Labor, PII is "Any representation of information that permits the identity of an individual to whom the information applies to be reasonably inferred by either direct or indirect means."
	Postman	Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated.
	Property Drilling	Property drilling, aka prop drilling, is the act of sending properties from parent components to child components through several hierarchy levels. Prop drilling

		is usually frowned upon as it makes React apps difficult to maintain and error prone.
R	**RWD**	Responsive Web Design. RWD is an approach to web design that makes web pages render well on a variety of devices and screen sizes from minimum to maximum display size.
S	**Serializer**	A serializer is a program that converts database objects into JSON format.
	SQL	Structured Query Language. SQL is a querying language that allows for the creation, reading, updating, deletion, and general manipulation of database objects.
W	**Watershed**	A watershed is an area of land that drains into a particular creek.
	Web Application	A web application is a client-side and server-side software application in which the client runs or sends requests in a web browser.

Table 2. 1. Terminology

System Architecture

This section discusses the structure of our system. The first subsection details the static attributes of our system, like what our backend and frontend portions are comprised of. The second subsection is about the dynamic structure of our system such as potential messages that are sent between the user, frontend, and backend.

Static Architecture Diagram & Description

This section specifically discusses what frameworks our project uses within our backend and frontend. Furthermore, this section explains how our front and backend components communicate with each other and within each other.

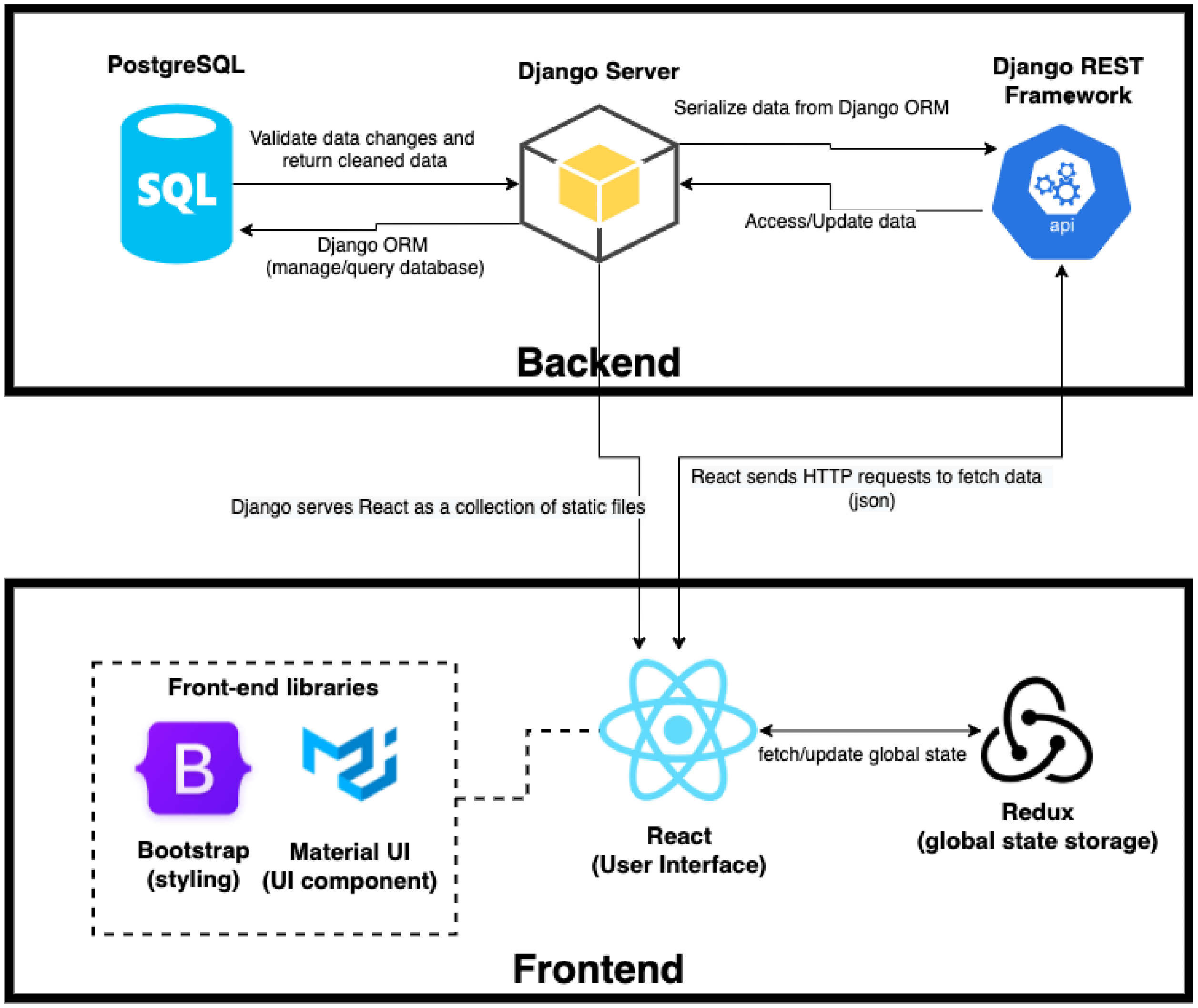

PostgreSQL
SQL
Django Server
Django REST Framework
api
Validate data changes and return cleaned data
Django ORM (manage/query database)
Serialize data from Django ORM
Access/Update data
Backend
Django serves React as a collection of static files
React sends HTTP requests to fetch data (json)
Front-end libraries
Bootstrap (styling)
Material UI (UI component)
React (User Interface)
fetch/update global state
Redux (global state storage)
Frontend

Figure 3.1: Static System Architecture Design for Proctor Creek Web App

Frontend

The frontend uses React Hook framework to build a single page application so all components are based on functions. We use Material UI to build UI components including sidebar, text field and buttons. We also use the grid system in Bootstrap and Material UI to make the web page responsive and mobile friendly. Redux helps handle global states of the frontend. Since React can only pass data down to its child components, we use Redux to avoid prop drilling. Redux in our application is used to save the administrator's information whenever they login which helps each component access the admin's login status.

Backend

The backend contains three main components: a PostgreSQL database, Django Server, and Django REST Framework. We use Axios to allow our frontend and backend to communicate and to support sending data back and forth. The backend will return data in JSON format, and the frontend will render the data on the webpage.

PostgreSQL: PostgreSQL is a relational SQL server. This will store all our user data as well as all report data that we will collect from user input. It communicates with the Django server. The database will validate all data being written or updated in the server.

Django Server: Django server is a centralized backend server that communicates with both the database and the REST API. This server handles all incoming requests from the frontend, and then decides what needs to happen from there. Django can read/write to the database using its ORM. If the frontend needs static files from the backend, Django can serve it directly. But if the frontend is trying to access or update any data within the PostgreSQL database, the frontend will have to communicate with the Django REST Framework using the API endpoint.

Django Rest Framework: This component is a RESTful API which can use GET, POST, PUT, or DELETE HTTP requests in order to access or update data. This component handles all API fetch requests and returns any response to the front end in JSON format. The REST API gets data by having the Django Server serialize database information into JSON to return to the front end. The REST API also can access/update data. It will pass this data into the Django Server which will use the Django ORM in order to correctly access/update the PostgreSQL database.

Dynamic Architecture Diagram & Description

This subsection explains the system's dynamic response to user actions. Figure 1.2 shows the messages passed between the user, the frontend, backend, and database according to a snapshot of potential user action. Our dynamic diagram is broken down into general user actions and administrator actions.

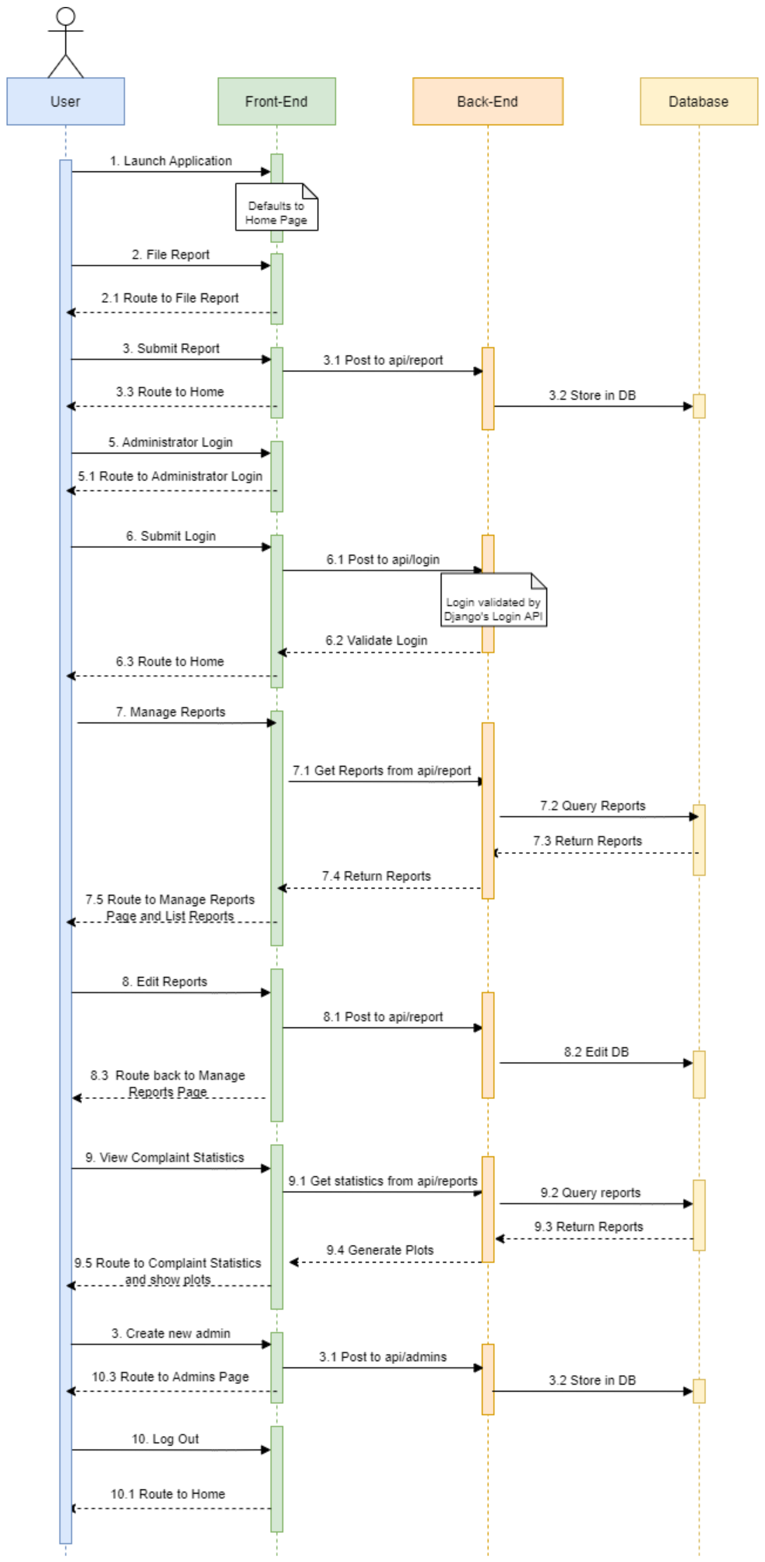


Figure 3.2: System Sequence Diagram for Proctor Creek Web App

https://drive.google.com/file/d/1r3n1DbA21sKCcJv64ZbTr9SxA4A6mXHN/view?usp=sharing

General User Features

To use our application, the user first must launch the application. Upon launching, the application directs to the Home Page. From this point, they have the options to file a report, view maps, get more information or log in as an administrator. If they choose to enter the File Report Page (2), they can then fill out and submit their report (3). After submitting their report, they are redirected to the Home Page (3.3). Most functions of our application redirect to the Home Page. If the user is an administrator, they can submit their login information (6), which will be validated by Django's native Login API. Afterward, they will be redirected to the Home Page and will be offered three additional pages: Manage Reports, Manage Organizations, and Administrator Settings.

Administrator Features

These three administrator pages are limited to only users with administrative credentials. Our application checks for credentials upon sending a get request to each respective endpoint so that regular users cannot simply type the endpoint into their browser and gain access. Upon successful validation (7.4), the application queries the database for all reports and specific fields. The response is propagated up to the user interface (7.3, 7.4) where the user is promptly redirected to the respective page.

Data Storage Design

The data storage design section has four subsections: database use, file use, data exchange, and security. The first subsection, database use, describes the database structure of the application in detail. Relationships between each table as well as their relationship classification are included. Moreover, the diagram is normalized to remove redundancies and adhere to diagram standards. The next subsection, file use, discusses what types of files the application uses. Following file use is a section about data exchange. It is about how the application transfers data. The data exchange subsection is modeled over the application as it would be in production, although this document was written during development. This subsection will explain what protocols the application uses. The last subsection is about security. Cybersecurity is a large concern of this project as the application will be storing PII. In this subsection, concerns regarding security are posed and addressed.

Database Use

The figure below features an ERD that describes our data entities, their fields, and relationships to other tables. We are using PostgreSQL for database storage in production and SQLite in the development environment.

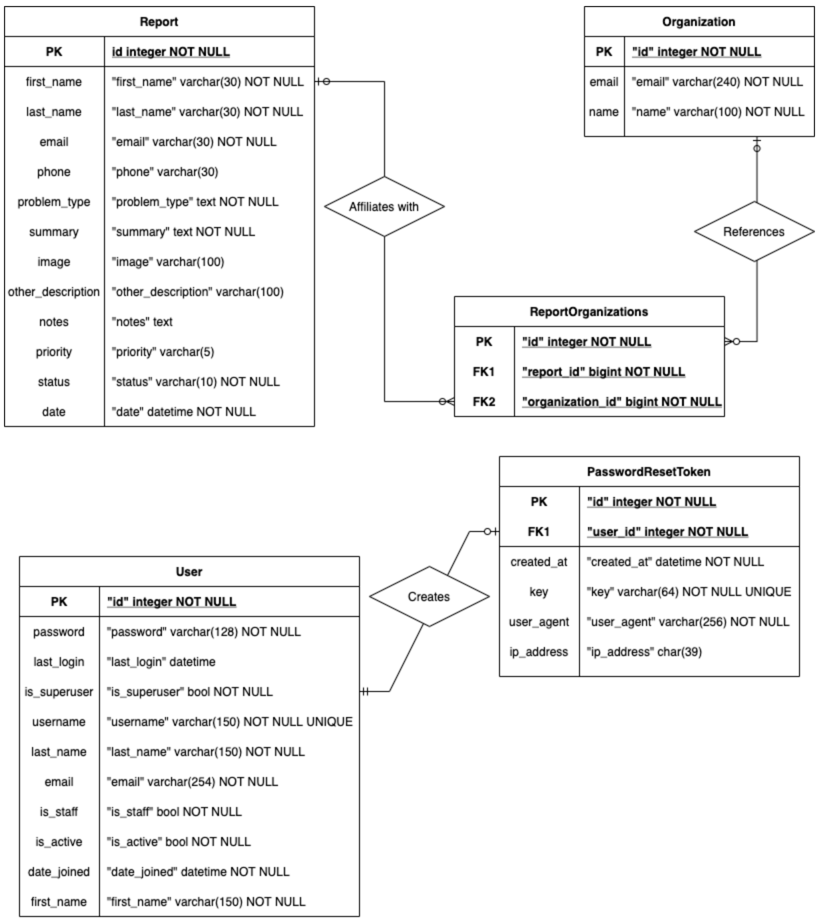

Report
PK | id integer NOT NULL
first_name | "first_name" varchar(30) NOT NULL
last_name | "last_name" varchar(30) NOT NULL
email | "email" varchar(30) NOT NULL
phone | "phone" varchar(30)
problem_type | "problem_type" text NOT NULL
summary | "summary" text NOT NULL
image | "image" varchar(100)
other_description | "other_description" varchar(100)
notes | "notes" text
priority | "priority" varchar(5)
status | "status" varchar(10) NOT NULL
date | "date" datetime NOT NULL
Organization
PK | "id" integer NOT NULL
email | "email" varchar(240) NOT NULL
name | "name" varchar(100) NOT NULL
Affiliates with
References
ReportOrganizations
PK | "id" integer NOT NULL
FK1 | "report_id" bigint NOT NULL
FK2 | "organization_id" bigint NOT NULL
PasswordResetToken
PK | "id" integer NOT NULL
FK1 | "user_id" integer NOT NULL
created_at | "created_at" datetime NOT NULL
key | "key" varchar(64) NOT NULL UNIQUE
user_agent | "user_agent" varchar(256) NOT NULL
ip_address | "ip_address" char(39)
Creates
User
PK | "id" integer NOT NULL
password | "password" varchar(128) NOT NULL
last_login | "last_login" datetime
is_superuser | "is_superuser" bool NOT NULL
username | "username" varchar(150) NOT NULL UNIQUE
last_name | "last_name" varchar(150) NOT NULL
email | "email" varchar(254) NOT NULL
is_staff | "is_staff" bool NOT NULL
is_active | "is_active" bool NOT NULL
date_joined | "date_joined" datetime NOT NULL
first_name | "first_name" varchar(150) NOT NULL

Figure 4.1. Entity Relationship Diagram

https://drive.google.com/file/d/1YOvP0x3hd-BmOJiB--o6iot3ChQLXlTU/view?usp=sharing

FK	Foreign Key
PK	Primary Key
UNIQUE	Attribute must be unique
NOT NULL	Attribute cannot be null (must have a value)
varchar(n)	Character array of size n (limited size)

Table 4.1. Entity Relationship Diagram Legend

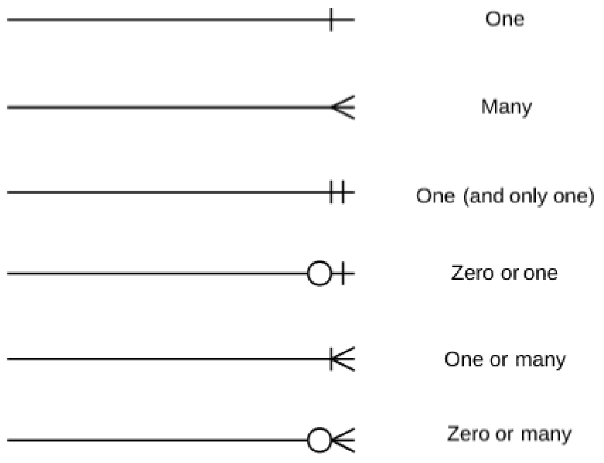


Figure 4.2. Entity Relationship Diagram Arrow Legend

Here is a list of all major entities:

Report:

Contains information about a report filed by a Proctor Creek resident (first_name, last_name, email, phone, summary, problem_type, and image).

The date field is automatically set to when the report is filed. Authenticated users can update this table information by adding other_description, notes, priority, and status. This table affiliates with the ReportOrganizations table with an (bi-directional optional) one to many relationship. This gives the authenticated user the ability to tag a filed report with 0 or more organizations.

Organization:

Contains information about an organization that is created by an authenticated user (email, name).

This table references the ReportOrganizations table with an (bi-directional optional) one to many relationship. This gives the authenticated user the ability to tag an organization with 0 or more reports.

ReportOrganizations:

Contains mappings between reports and organizations with a report_id and organization_id.

Both fields are foreign keys that reference the Report and Organization tables, respectively. This table is the product of normalizing the otherwise (bi-directional optional) many to many relationship between reports and organizations.

User:

Contains information of any authenticated users (password, last_login, is_superuser, username, last_name, email, is_staff, is_active, date_joined, and first_name).

Passwords are "salted" to encrypt the password data in this table, which will make any password data breach useless. This table has a one to (optional) one relationship with the PasswordResetToken table. This allows a user to optionally reset their password if they have forgotten it.

PasswordResetToken:

Contains information about password reset tokens (user_id, created_at, key, user_agent, ip_address).

The foreign key that references the User table is "user_id." The reset token itself is the "key" attribute. "user_agent" and "ip_address" refers to the application that submitted the request and its IP. "created_at" refers to the time the token was created. This table has an (optional) one to one relationship with the User table. This allows a created password token to be affiliated with a single user.

File Use

Images are stored locally in development (and are served as urls to the back-end server) but will be stored in Amazon S3 buckets in production. We will support all image types (.jpg, .svg, .png) when uploading an image for a report. Because our chosen language for development is JavaScript, most of our files have the .js extension.

Data Exchange

Only our back-end server can create, read, update, or delete data in our database (PostgreSQL in production, SQLite in development). It is important to note that PostgreSQL and SQLite use the same protocols and data exchange. We are using SQLite in development to facilitate local data visualization. To send data to our front-end application, the backend hosts an API, which will send data through the HTTP protocol (HTTP in development, HTTPS in production). The HTTP requests sent to the back end return a JSON response to the front end, which it can read to utilize the database data.

Security

Security is our biggest concern when transferring our application into production. SQLite is used in development due to its easy interface and visualization. However, SQLite carries high security risks and can easily be prone to attacks such as SQL injection. We are also using HTTP, rather than HTTPS in development. HTTP is not safe and should not be used in production due to unencrypted data being sent across applications. Since we are running our application locally during the development phase, security is not a concern. However, when we bring our application into production, we need to switch to a safe database and HTTP protocol. We will use PostgreSQL rather than SQLite, due to its built-in security measures that can help keep our data safe and encrypted. We will also switch to HTTPS, which encrypts all data being sent between our back-end server and front-end application, making our data safely transfer between applications. For further security, we "salt" every user's password with random data to encrypt it within the data table. This adds an extra layer of security to guarantee that no password data is leaked or other PII.

Component Detailed Design

The component detailed design will show the static and dynamic components of our system in greater detail. A higher-level description of our system can be found in the System Architecture section. The static and dynamic diagrams in each section are comparable (frontend and backend, varying user functions). This section has two subsections: static and dynamic components. The static section shows the relationships and connections between our front-end and backend components, namely the functional components we use in the frontend and our database tables in the backend. The dynamic component section shows the different paths our system would take depending on specific user actions. This subsection will show paths for both general user and administrator actions.

Static Components

The static components in Figure 5.1 highlight the relationships between our frontend and backend components. The frontend components are functional components which correspond to specific pages, styles, and smaller components. The backend components represent database tables. Lastly, the connections between the frontend and backend components show where data is passed back and forth.

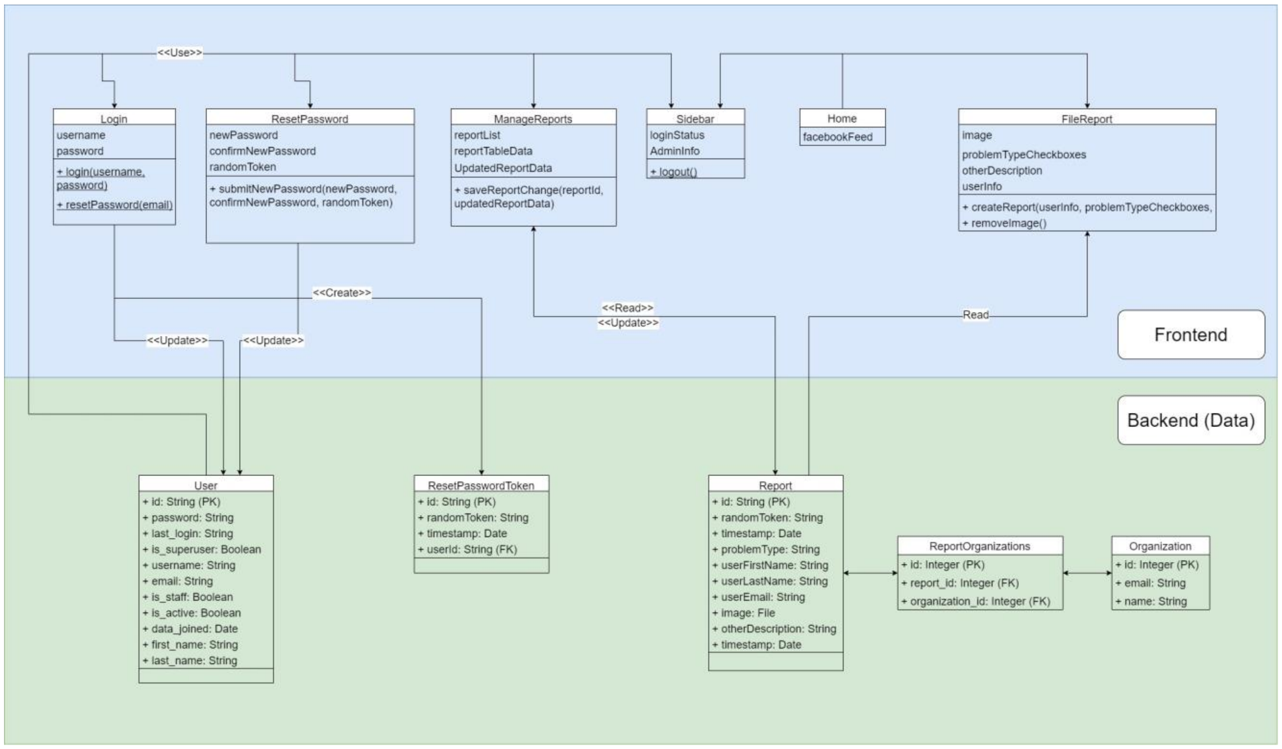


Figure 5.1. Static Component Diagram

https://drive.google.com/file/d/12FVDcc9ChR8VUOdM_cCDFA8xuOQ0kY6r/view?usp=sharing

Figure 5.1 is organized into two distinct components: frontend and backend. Furthermore, the frontend is divided into administrative and general user portions. The general user portions are located on the right of the Sidebar class and the administrative pages are to the left. The general user pages are for Proctor Creek residents who are not members of the Proctor Creek Stewardship Council. The administrative pages are specifically designed for members of the Council. Moreover, the administrative pages correspond to all data tables, and the general user is only linked to the Report table. Subscripts of how our frontend interacts with the backend are located on corresponding lines.

Dynamic Components

Our Dynamic Component Design is modeled by a robustness diagram which displays how the user interacts with the components and how the components interact with each other. A detailed explanation of the diagram is located below. Note: Figure 5.2 differs from Figure 5.1 in that a table for organizations exists, though we do not have functionality to create an organization yet.

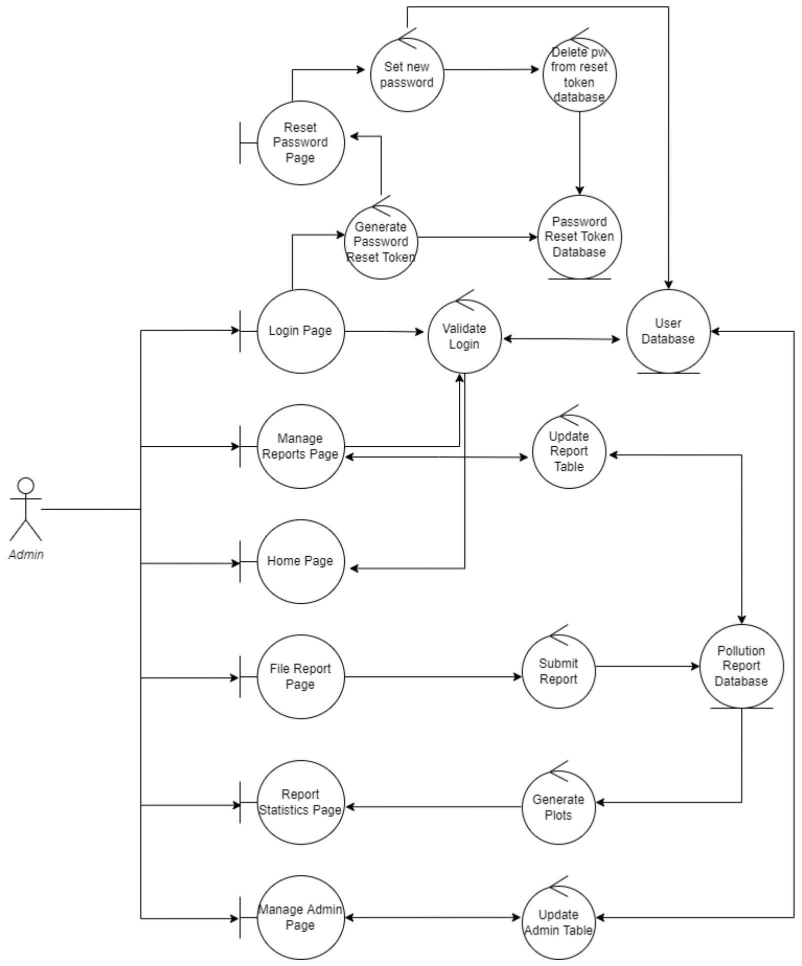


Figure 5.2. Robustness Diagram

https://drive.google.com/file/d/1B_kK8czlM3Pm-Mf4Z2StbpKyvh-q34nj/view?usp=sharing

Forgot Password:

When a user interacts with the login page, they can select that they forgot their password, and this will generate a reset password token and add that to the reset token database. Through an email sent to the user, they will then be taken to the reset password page where they can set a new password which will update the user database and remove the reset password token from the reset token database.

Login:

From the login page the user can also login by entering their email and password, which is verified by the user database and gives the user a JWT indicating that they are an administrator and then sends them to the Home page.

Manage Reports:

If the user has a valid JWT, the Home page will display an option to visit the manage reports page which displays contents of the pollution report database. The manage reports page only displays reports if the user is an administrator. On this page, the user can edit certain fields of reports which will update within the pollution report database.

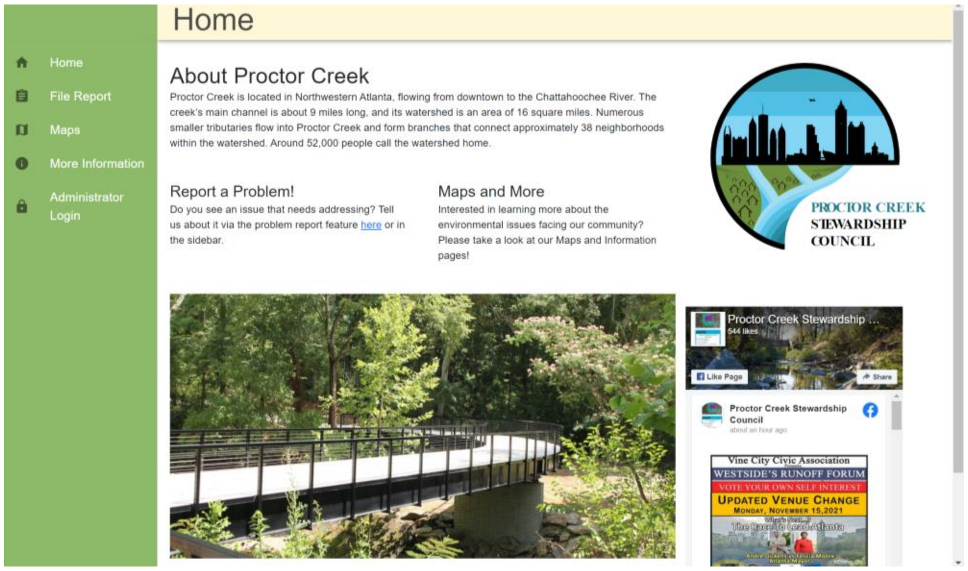
File Report:

General users can submit a complaint via the file a report page; once they have filled the required fields in that page and submitted, the pollution report database will update with the new report.

UI Design

The UI Design section will cover the primary screens the user will interact with when using our application as well as the UI design decisions behind them.

When the user opens the application, they are brought to the Home screen consisting of a sidebar with navigational links to key parts of the application, as well as a primary page with sections containing information about Proctor Creek (Figure 6.1). The Home screen includes a Facebook feed of the Proctor Creek Facebook page. Here, as well as all other screens of the application, the user is made aware of the system's status through the page header (clearly marked as "Home" in Figure 6.1). As a result of the side bar being always accessible, the user is given constant control when navigating the application. We also see that the side bar matches the user's language, having familiar icons associated with the page to which they link (House->Home, Report->File Report, …).



Home

About Proctor Creek

Proctor Creek is located in Northwestern Atlanta, flowing from downtown to the Chattahoochee River. The creek's main channel is about 9 miles long, and its watershed is an area of 16 square miles. Numerous smaller tributaries flow into Proctor Creek and form branches that connect approximately 38 neighborhoods within the watershed. Around 52,000 people call the watershed home.

Report a Problem!

Do you see an issue that needs addressing? Tell us about it via the problem report feature here or in the sidebar.

Maps and More

Interested in learning more about the environmental issues facing our community? Please take a look at our Maps and Information pages!

Figure 6.1. Home Page

From the Home page, the user can navigate to the file a report page (Figure 6.2). This page consists of a form that the user can fill in and then submit. This page follows the principal of recognition instead of recall because it clearly labels all the form entries so that a user can be informed on what is needed in each field without any prior knowledge. This page design also helps users recover from errors since it displays what page they are on, gives access to other pages, and reminds users of required fields denoted with an asterisk.

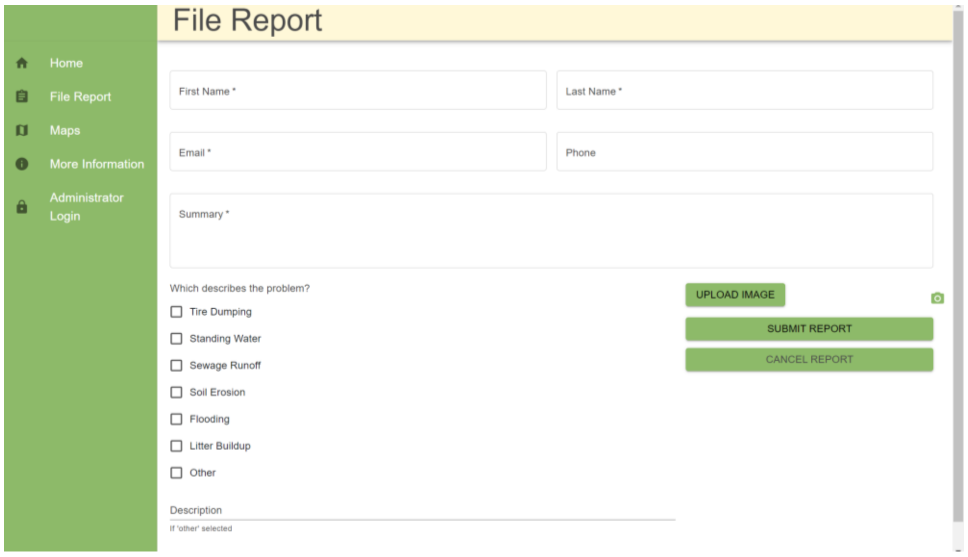


Figure 6.2. File Report Page

By clicking on the Administrator Login tab in the side bar, the user is taken to the Login page (Figure 6.3). This page presents users fields to input their username and password and a link to help users if they forgot their password. This page follows the principle of minimalist design by showing the user what is needed to login and keeping the page clean and simple. It avoids having unnecessary parts that might confuse the user and lower usability. When the user submits their credentials correctly, they will be taken to the home page with new administrator functionality available.

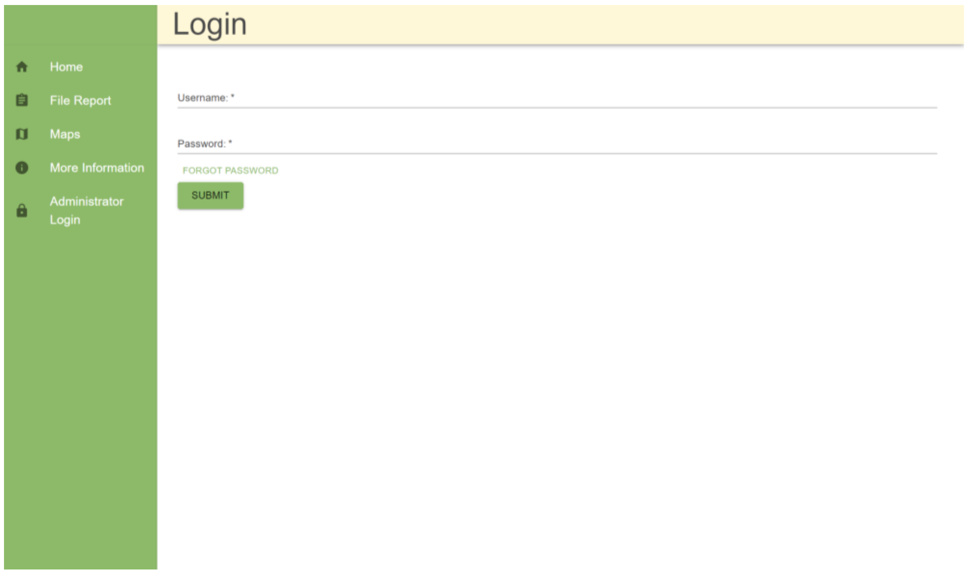
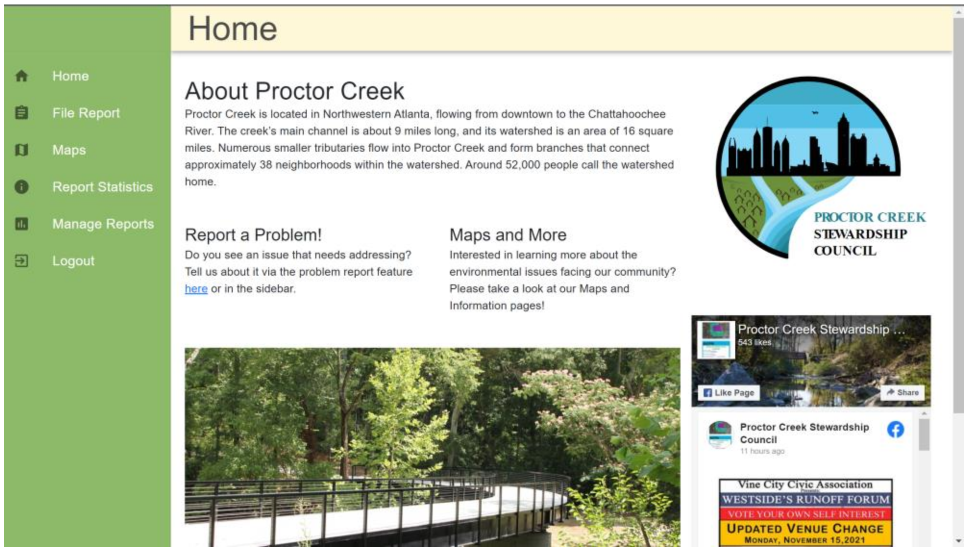


Figure 6.3. Login Page

Upon successfully logging in as an administrator, the administrator is redirected to the Home page and given specific administrator functionalities in the sidebar (Figure 6.4). The sidebar is essentially the same as the original sidebar, except with “Administrator Login” updated to “Logout” and the addition of two new sections in the sidebar, “Manage Reports,” and “Report Statistics.” The addition of these new sections, as well as the change from “Login” to “Logout” in the sidebar, each with an appropriate change in icon, ensure that the change in system status is visible to the user.



Home

About Proctor Creek

Proctor Creek is located in Northwestern Atlanta, flowing from downtown to the Chattahoochee River. The creek's main channel is about 9 miles long, and its watershed is an area of 16 square miles. Numerous smaller tributaries flow into Proctor Creek and form branches that connect approximately 38 neighborhoods within the watershed. Around 52,000 people call the watershed home.

Report a Problem!

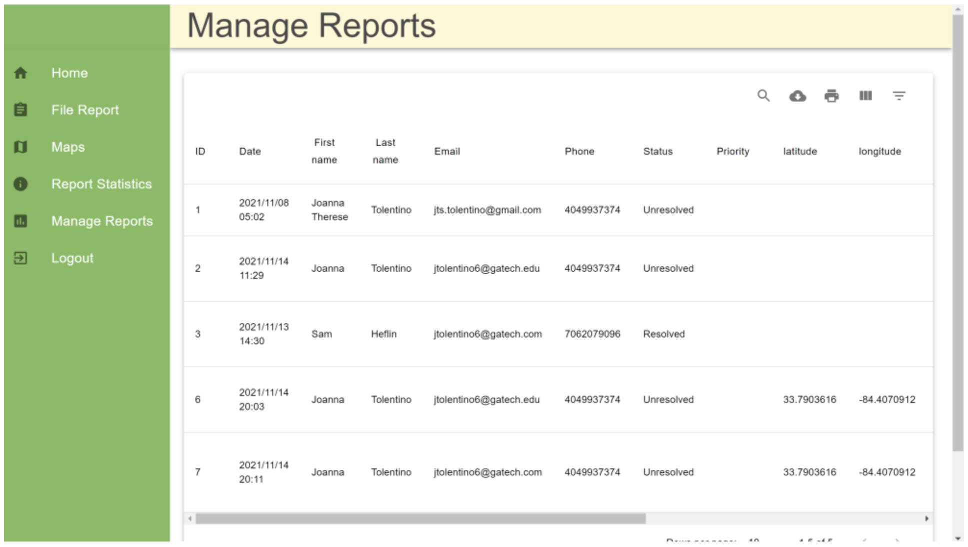
Do you see an issue that needs addressing? Tell us about it via the problem report feature here or in the sidebar.

Maps and More

Interested in learning more about the environmental issues facing our community? Please take a look at our Maps and Information pages!

Figure 6.4. Home Page for Logged in administrator

In the Manage Reports page, the administrator is presented with a table of all the pollution reports in the database (Figure 6.5). In this table, administrators can sort the table by the values in each of the columns, and they can choose to filter the table by certain values. The administrator can also edit the reports by pressing the edit button on the far-right side of each report. This button was designed to match the system and the real world by using a pencil icon to represent it. This icon corresponds to how people can edit and write things in a non-digital setting.

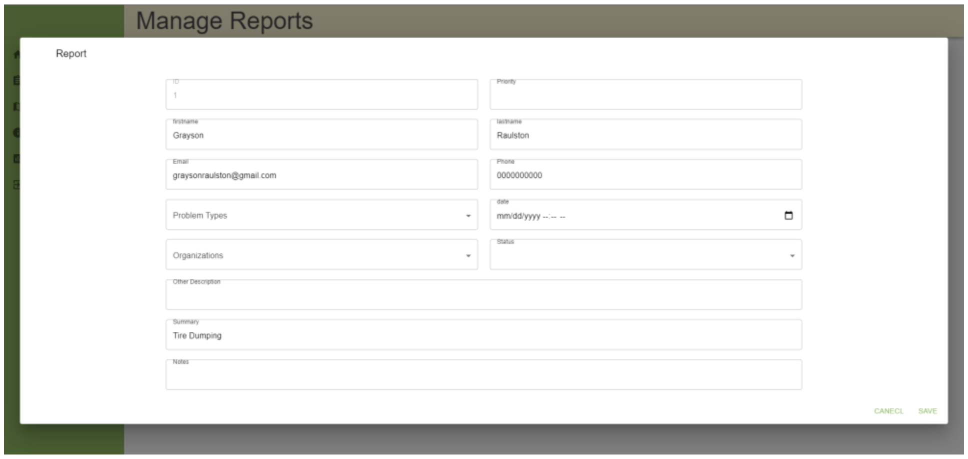


Manage Reports

ID	Date	First name	Last name	Email	Phone	Status	Priority	latitude	longitude
1	2021/11/08 05:02	Joanna Therese	Tolentino	jts.tolentino@gmail.com	4049937374	Unresolved			
2	2021/11/14 11:29	Joanna	Tolentino	jtolentino6@gatech.edu	4049937374	Unresolved			
3	2021/11/13 14:30	Sam	Heflin	jtolentino6@gatech.com	7062079096	Resolved			
6	2021/11/14 20:03	Joanna	Tolentino	jtolentino6@gatech.edu	4049937374	Unresolved		33.7903616	-84.4070912
7	2021/11/14 20:11	Joanna	Tolentino	jtolentino6@gatech.com	4049937374	Unresolved		33.7903616	-84.4070912

Figure 6.5. Manage Reports Page

By clicking on the edit button on a report in the Manage Reports page the user is displayed a dialog with fields that can be edited (Figure 6.6). In this dialog, the user can edit the values in the different fields of the report and when they are done changing entries, they can press save which will update the values of the report and return the administrator to the Manage Reports page. If the administrator decides that they do not want to make any changes, they can press cancel which will take them back to the Manage Reports page without changing any values.


Manage Reports

Report

ID: 1	Priority:
firstname: Grayson	lastname: Raulston
Email: graysonraulston@gmail.com	Phone: 0000000000
Problem Types	date: mm/dd/yyyy --:-- --
Organizations	Status:
Other Description:	
Summary: Tire Dumping	
Notes:	

Figure 6.6. Manage Reports Page Edit Dialog

Lastly, the administrator also has access to the Report Statistics page (Figure 6.7). This page compiles report statistics into five graphs representing report activity, report status, problem type distribution, organization distribution, and priority distribution. Administrators can interact with the charts by hovering over attributes and receiving specific values.

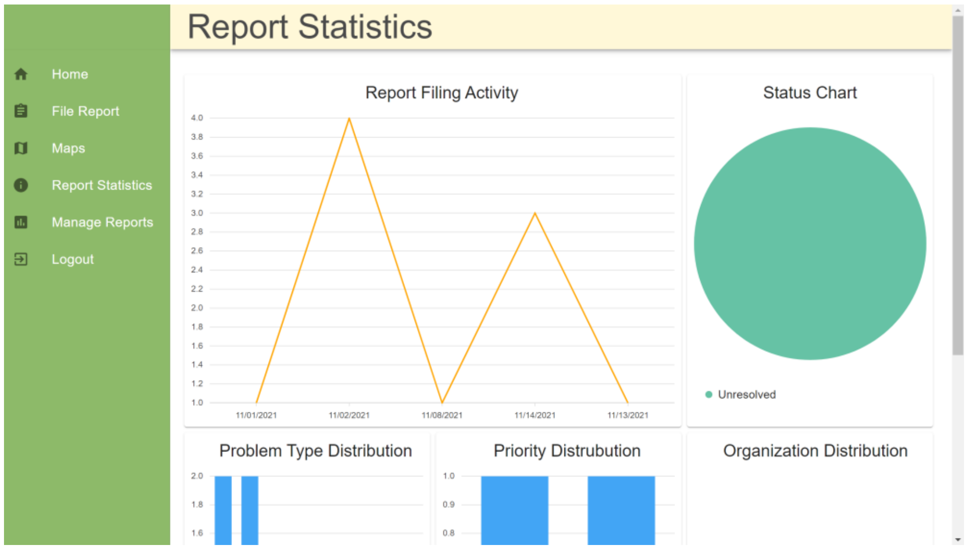


Figure 6.7. Report Statistics Page

Appendix A: REST API Documentation

Response Codes

Code	Text	Description
200	OK	Successful call to the API, usually for GET and PUT requests
201	Created	Successful creation of an object due to API call, usually for POST requests
400	Bad Request	API request cannot be served, comes with a more detailed error
401	Unauthenticated	User is not able to make a request due to lack of an authentication token
403	Forbidden	The request is understood but won't be fulfilled, as it is not allowed to be
404	Not Found	The URI requested is either invalid or does not exist
500	Internal Server Error	The server ran into an error when handling a request
503	Service Unavailable	The server is overloaded with requests, try again later

Table 7.1. Response Codes

POST api/token/

Returns a refresh and access JWT token given correct login credentials

URL: http://localhost:8000/api/token/

Request Format: JSON (params are in form body)

Response Format: JSON

Requires Authentication: False

Rate Limited: No

Parameters:

Parameter	Description
username	Username affiliated with a user account
password	Password affiliated with username

Response Fields:

Response Field	Description
refresh	Refresh JWT token, can be used for 14 days (about 2 weeks) to refresh access token
access	Access JWT token used to authenticate protected API calls. Lasts for 3 days, can be refreshed using refresh token

POST api/token/refresh/

Returns a new access JWT token given valid refresh token

URL: http://localhost:8000/api/token/refresh/

Request Format: JSON (params are in form body)

Response Format: JSON

Requires Authentication: False

Rate Limited: No

Parameters:

Parameter	**Description**
refresh	Refresh JWT token given when logged in

Response Fields:

Response Field	**Description**
access	Access JWT token used to authenticate protected API calls. Lasts for 3 days, can be refreshed using refresh token

POST api/decode/

Returns a new access JWT token given valid refresh token

URL: http://localhost:8000/api/decode/

Request Format: JSON (params are in query paramaters)

Response Format: JSON

Requires Authentication: False

Rate Limited: No

Parameters:

Parameter	Description
jwt	JWT to decode user information from

Response Fields:

Response Field	Description
access	Access JWT token used to authenticate protected API calls. Last for 3 days, can be refreshed using refresh token
exp	Timestamp that tells when the JWT will expire
jti	A unique identifier for the givemn JWT
user_id	Primary key for the user affiliated with the JWT
username	Username for the user affiliated with the JWT
email	Email for the user affiliated with the JWT

POST api/password_reset/

Allows the backend to send an email to the user containing a token that will allow the user to change their password

URL: http://localhost:8000/api/password_reset/

Request Format: JSON (params are in form body)

Response Format: JSON

Requires Authentication: False

Rate Limited: No

Parameters:

Parameter	Description
email	Email associated with an account that wishes to reset their password

Response Fields:

Response Field	Description
status	Status of the request

POST api/password_reset/confirm/

Updates user password with a new one given a token

URL: http://localhost:8000/api/password_reset/confirm/

Request Format: JSON (params are in form body)

Response Format: JSON

Requires Authentication: False

Rate Limited: No

Parameters:

Parameter	Description
token	Token from password reset email sent at the POST to api/password_reset endpoint
password	New password to update the user information with

Response Fields:

Response Field	Description
status	Status of the request

GET api/images/

Get all images or ones only for a certain report

URL: http://localhost:8000/api/images/

Request Format: JSON (params are in query parameters)

Response Format: JSON

Requires Authentication: True

Rate Limited: No

Parameters:

Parameter	Description
report	The primary key of the report you want to filter images through

Response Fields:

The response is an array of image objects. Below is each image object's fields and description

Field	Description
pk	The primary key of the image
report	The primary key of the report associated with the image
image	Image URL

POST api/images/

Updates the database with new image(s)

URL: http://localhost:8000/api/images/

Request Format: JSON (params are in form body)

Response Format: JSON

Requires Authentication: False

Rate Limited: No

Parameters:

Parameter	Description
report	The primary key of the report
image	New password to update the user information with

Response Fields:

The response is an array of image objects that have just been created. Below is each image object's fields and description

Field	Description
pk	The primary key of the image
report	The primary key of the report associated with the image
image	Image URL

GET api/report/

Get all filed reports

URL: http://localhost:8000/api/report/

Request Format: JSON

Response Format: JSON

Requires Authentication: True

Rate Limited: No

Parameters: None

Response Fields:

The response is an array of report objects that have just been created. Below is each report object's fields and description

Field	Description
pk	The primary key of the report
first_name	The first name given in a report
last_name	The last name given in a report
email	The email given in a report
phone	The phone number given in a report
problem_type	The problem type of a report
summary	A summary of the reported pollution
other_description	If problem type is other, this field will have a description of the problem type
date	Date of report upload

notes	Extra notes attached to the report by an administrator
priority	Priority of a report set by an administrator
status	Status of a report set by an administrator
organizations	Organizations of a report tagged by an administrator
latitude	Latitude of a report filing
longitude	Longitude of a report filing

POST api/report/

File a report

URL: http://localhost:8000/api/report/

Request Format: JSON (params are in form body)

Response Format: JSON

Requires Authentication: False

Rate Limited: No

Parameters:

Field	Description
first_name	The first name given in a report
last_name	The last name given in a report
email	The email given in a report
phone (optional)	The phone number given in a report
problem_type	The problem type of a report
summary	A summary of the reported pollution
other_description (optional)	If problem type is other, this field will have a description of the problem type
notes (optional)	Extra notes attached to the report by an administrator
priority (optional)	Priority of a report set by an administrator
status (optional)	Status of a report set by an administrator
organizations (optional)	Organizations of a report tagged by an administrator
latitude (optional)	Latitude of a report filing
longitude (optional)	Longitude of a report filing

Response Fields:

The response is the report object that has just been created. Below is a report object's fields and description

Field	Description
pk	The primary key of the report
first_name	The first name given in a report
last_name	The last name given in a report
email	The email given in a report
phone	The phone number given in a report
problem_type	The problem type of a report
summary	A summary of the reported pollution
other_description	If problem type is other, this field will have a description of the problem type
date	Date of report upload
notes	Extra notes attached to the report by an administrator
priority	Priority of a report set by an administrator
status	Status of a report set by an administrator
organizations	Organizations of a report tagged by an administrator
latitude	Latitude of a report filing
longitude	Longitude of a report filing

PUT api/report/id/

Update a filed report

URL: http://localhost:8000/api/report/

Request Format: JSON (params are in form body)

Response Format: JSON

Requires Authentication: True

Rate Limited: No

Parameters: the "id" in the URL is the primary key of the report that is being updated

Field	Description
first_name	The first name given in a report
last_name	The last name given in a report
email	The email given in a report
phone (optional)	The phone number given in a report
problem_type	The problem type of a report

summary	A summary of the reported pollution
other_description (optional)	If problem type is other, this field will have a description of the problem type
notes (optional)	Extra notes attached to the report by an administrator
priority (optional)	Priority of a report set by an administrator
status (optional)	Status of a report set by an administrator
organizations (optional)	Organizations of a report tagged by an administrator
latitude (optional)	Latitude of a report filing
longitude (optional)	Longitude of a report filing

Response Fields:

The response is the report object that has just been updated. Below is a report object's fields and description

Field	Description
pk	The primary key of the report
first_name	The first name given in a report
last_name	The last name given in a report
email	The email given in a report
phone	The phone number given in a report
problem_type	The problem type of a report
summary	A summary of the reported pollution
other_description	If problem type is other, this field will have a description of the problem type
date	Date of report upload
notes	Extra notes attached to the report by an administrator
priority	Priority of a report set by an administrator
status	Status of a report set by an administrator
organizations	Organizations of a report tagged by an administrator
latitude	Latitude of a report filing
longitude	Longitude of a report filing

GET api/organization/

Get all organizations

URL: http://localhost:8000/api/organization/

Request Format: JSON

Response Format: JSON

Requires Authentication: True

Rate Limited: No

Parameters: None

Response Fields:

The response is an array of organization objects. Below is each report object's fields and description

Field	Description
pk	The primary key of the organization
name	The name of the organization
email	The email affiliated with the organization

POST api/organization/

Add an organization

URL: http://localhost:8000/api/organization/

Request Format: JSON (parameters in form body)

Response Format: JSON

Requires Authentication: True

Rate Limited: No

Parameters:

Field	Description
name	The name of the organization
email	The email affiliated with the organization

Response Fields:

The response is the organization object that has just been created. Below is each organization object's fields and description

Field	Description
pk	The primary key of the organization
name	The name of the organization
email	The email affiliated with the organization

POST api/user/

Add a new admin

URL: http://localhost:8000/api/user/

Request Format: JSON (parameters in form body)

Response Format: JSON

Requires Authentication: True

Rate Limited: No

Parameters:

Field	Description
username	The name of the account
email	The email of the admin
password	The password of admin account
first_name	The first name of the admin
last_name	The last name of the admin
is_staff	Is the user a staff (defualt=true)
is_superuser	Is the user a superuser

Response Fields:

Field	Description
pk	Primary key of the account
username	The name of the account
email	The email of the admin
password	The password of admin account
first_name	The first name of the admin
last_name	The last name of the admin
is_staff	Is the user a staff (defualt=true)
is_superuser	Is the user a superuser
date_joined	Account created date

GET api/user/

Get admin list

URL: http://localhost:8000/api/user/

Request Format: JSON

Response Format: JSON

Requires Authentication: True

Rate Limited: No

Parameters: None

Response Fields:

Field	Description
pk	Primary key of the account
username	The name of the account
email	The email of the admin
password	The password of admin account
first_name	The first name of the admin
last_name	The last name of the admin
is_staff	Is the user a staff (defualt=true)
is_superuser	Is the user a superuser
date_joined	Account created date

Appendix B: Detailed Design Collaboration Form

Team member name	Role(s) performed for each draft (e.g., 1st, 2nd, final)
Dylan Small	Pre-writer/Brainstormer, Drafter
Christopher Raulston	Pre-writer/Brainstormer, Drafter
Ryan Bauer	Pre-writer/Brainstormer, Drafter
Chiche Tsai	Pre-Writer/Brainstormer, Drafter
Joanna Tolentino	Pre-Writer/Brainstormer, Drafter, Copyeditor/Proofreader, Delivery Coordinator