Developing a Web Application with SQL and NoSQL Database

Management Systems: An Exploratory Case Study

SID: 2182389

# Table of Contents

[Aims and Objectives:](#_Aims_and_Objectives:)

[Literature Review](#_Literature_Review)

[Methodology](#_Methodology)

[Design](#_Design)

[Implementation](#_Implementation)

[Testing](#_Testing)

[Discussion](#_Discussion)

[Conclusion](#_Conclusion)

[References](#_References)

[Appendix](#_Appendix)

# Aims and Objectives

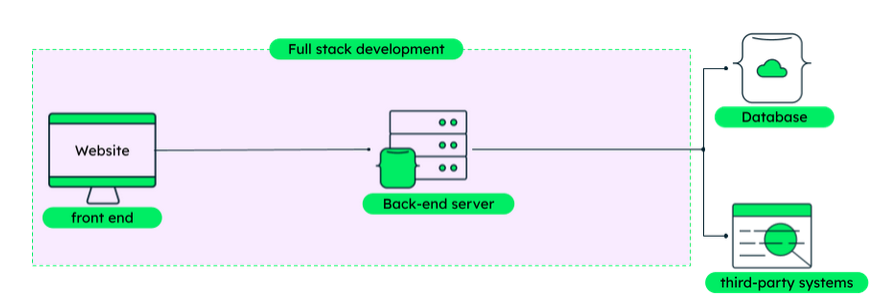
In common parlance, a full-stack application conjures an idea of a web application, a server, a runtime environment for the web application and a database management system. There is a wealth of technologies available for developing these applications, and selecting the appropriate choices are crucial parts of the software development lifecycle. Most modern web applications are developed using a framework system. Frameworks simplify the web development process by using APIs to perform DOM manipulation. Beyond providing standard styles to develop web applications, frameworks often include support for databases, session management and promote code reuse. Modern web frameworks use the Model-View-Controller paradigm, separating the web application into data, the presented view, and logic to modify that view based on changes to the data. Full-stack development makes use of tiered organisation, separating the data storage, application and client to keep business logic for each area separated and secure.

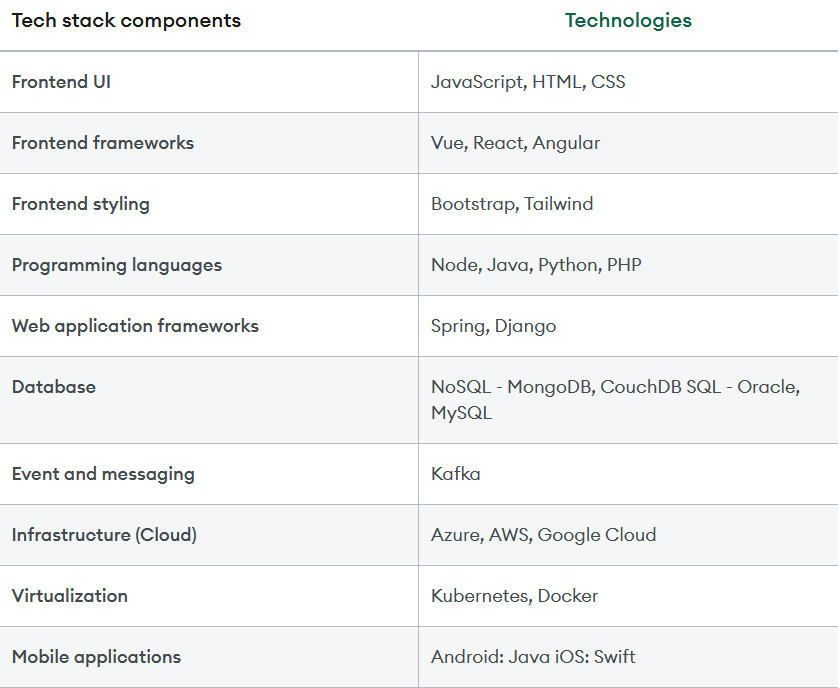
The aim of this project is to explore the development process of a full-stack application, including an interactive front-end, data storage, data collection pipeline and RESTful API server. This project has a timespan of approximately 17 weeks. As the data used in the databases does not exist in readily formatted data, it must be parsed and stored, which is done using Python scripts. The technology used in this project will be VueJS v3.2 and Bootstrap v5.3 to develop and deliver the front-end of the application, an Express server runs the API to manage interactions between the front-end and the back-end. The back-end will consist of a PostgreSQL and MongoDB database, as well as a collection of Python scripts to manage the PostgreSQL database. To keep the application user-friendly and secure, best practices will be implemented to the best of their ability in the given time frame. These best practices are compiled from official documentation, enthusiast driven blogs and discussion boards, and industry leaders.

# Literature Review

## Full Stack Development

A full stack application is an application that is built end-to-end, including front- and back-ends. (MongoDBb, 2023) The front-end of the application is the user-interface (UI) where the user interacts, for example where one can browse products on an e-commerce website. The back-end of the application is the part of an application performs most of the business logic and interacts with stored data, often through an API. (MongoDB (a), 2023) Front-end development typically uses HTML, CSS and JavaScript. (Eddie, 2022) Back-ends can be written in a variety of languages and emphasize speed and scalability. (Eddie, 2022). Full-stack development is defined by Landry et al as, “a methodology which addresses all stack layers and in doing so creates a complete, implementable solution to business requirements”.

End-to-end production workflow. (Mongo)

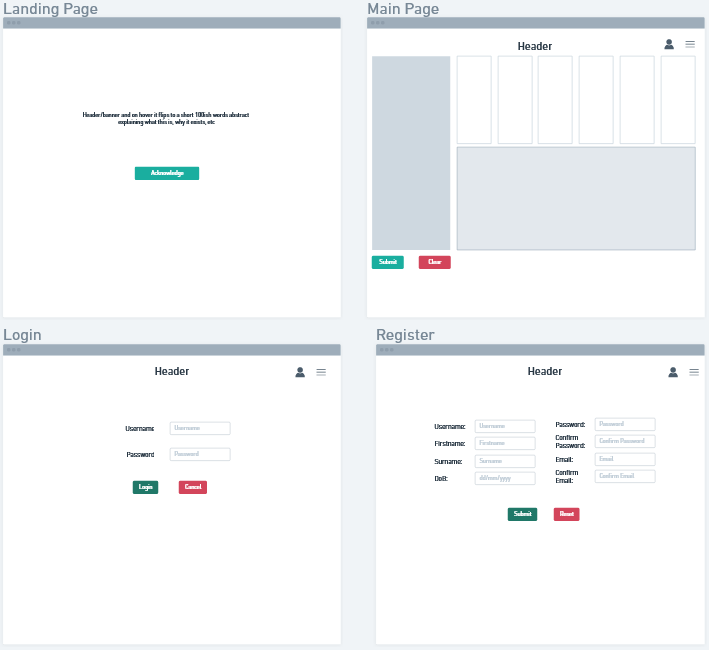
A “technology stack”, which may also be referred to as a software stack or simply a stack, is the set of technologies used to develop an application. (Heap). Tech stacks traditionally consist of a server operating system, web server, database server and programming language, however this is not an exhaustive list. (Landry) When choosing a tech stack for an application it is important to compare what different technologies offer and what they lack. (Heap) Choosing an incorrect technology without considering the costs slow down production time waste useful hours writing code for a system that will not be implemented. Planning when choosing a tech stack reduces horizontal and vertical scaling issues as the application’s userbase grows. (Mongo (a), 2023)

Examples of different technologies commonly used in various parts of a technology stack. (Mongo)

## Presentation

User experience (UX) for this project was designed to be simple and intuitive. Bilousova et al state that UX is aimed to achieve application purpose and user satisfaction. This is contrasted to UI focuses on the process of the application presentation and improving interactivity. (Bilousova, et al) The United States General Services Administration (USGSA) (2006) offer guidelines to keep user experience (UX) comfortable and accessible. The USGSA suggests a simple and direct interface. An interface should not have unnecessary elements and necessary elements should have clear language on labels and in messages. (USGSA, 2006) Elements should be consistent and purposefully laid out on the page, with colour and texture, such as drop shadows or input field highlighting, in mind. (USGSA, 2006)

Emily Stevens (2002), writing for UX Design institute, emphasizes context when designing an application. It is important to consider what devices a user may use for the application, what assumed knowledge related to the product and data of the application, and so forth. Understanding the context in which an application is used is useful to consider both layout and accommodation to create a better experience. (Stevens, 2022) Creating hierarchy helps guide users around a page, emphasizing the more important elements at the top of the hierarchy. (Stevens, 2022) Typography can be used to create hierarchy and make a clearer presentation. Using different font sizes and arrangements of text can make a page faster and more intuitive to navigate while simultaneously increasing readability. (USGSA, 2006) Visual hierarchy considers the layout of the individual elements. Placing elements in the centre or top of a page emphasizes their importance to the user. (Stevens, 2022) Styling the application with different colours can help smaller important elements stand out to the user. (Stevens, 2022)



Wireframe planning of the application user-interface.

## Data Storage

Databases are used to store data to be made accessible quickly and accurately by a user or application. (Harrington, 2016, p. 47) Anderson and Nicholson (2023), two senior database engineers at IBM, explain the differences and use-cases for a relational or non-relational database system. SQL is a language used for interacting with relational databases, invented in 1970 and has been the standard, in various flavours, since then. (Anderson and Nicholson, 2022) SQL databases are very efficient with structured data, that is data that has a relationship between its variables and entities. (Anderson and Nicholson, 2022) Mostly all relational databases use SQL to access and modify data. (Jatana, et al, 2012) PostgreSQL (2023) uses the SQL language, but also extends it to add features to the standard SQL language. RDBMSes emphasize reducing duplication in a database and store information in tables. (Microsoft, 2023) This project will use PostgreSQL for its RDBMS. In a non-RDBMS, often referred to as a “NoSQL” database, data may be unstructured and vary from entry to entry or can be made to adhere to a schema. (Anderson and Nicholson) This project will use MongoDB for its non-RDBMS.

### Relational Database

In SQL database design theory, an entity is anything about which a data is stored. (Harrington, 2016, p. 55) For example, in an e-commerce database, a customer, a widget for sale and an order are all entities. It is important to avoid “collections of entities” by not organizing data in a way where there are multiple values for an attribute in an entry. (Harrington, 2016, p. 60) According to Microsoft (2023) a good RDBMS design is one that satisfies four rules:

1. A database must divide information into subject-based tables to reduce redundancy
2. Provide the information required to join the information together into tables
3. Support and ensure the accuracy and integrity of the information
4. Accommodates data processing and reporting needs

When designing a database, it is important to ensure each row in each table has a unique identifier, called a primary key, and this primary key must always have a value. (Microsoft, 2023) In PostgreSQL, this can be set using a SERIAL type, which incrementally increases with each new entry created. (PostgreSQL, 2023)

Normalization rules in SQL database design can be used to ensure that tables are structured correctly. (Microsoft, 2023) There are three normalization rules that are assumed to be required required in a majority of database design. (Microsoft, 2023) The first normal form states every row and column intersection in the table is a single value and never a list. To satisfy the second normal form, each non-key column must be fully dependent on the entire primary key. If a table has two primary keys, non-key columns must be fully dependent on both primary keys. The third normal form requires every non-key column be independent of the other non-key columns.

According to both Jatana, et al (2012) and Anderson and Nicholson (2023), SQL databases must be what is referred to as “ACID compliant” which means they must exhibit the following four properties: atomicity, consistency, isolation, and durability. To have atomicity, a transaction in a database must either succeed or fail completely and cannot be left partially complete. (Anderson and Nicholson, 2022). Consistency means database must have rules to ensure it remains stable and valid, both before and after transactions. (Anderson and Nicholson, 2022).

An RDMBS is a good technology choice when data is structured and related in easy to store ways. (Jatana, et al 2012). When data sets are too large or very complex, the shortfalls of an RDBMS become apparent, although many RDBMS, including PostgreSQL, have many features for dealing with complex data, they are not often used, adding to the cost of the RDBMS itself. (Anderson and Nicholson, 2022) RDBMS can scale vertically, by increasing the hardware on a server to a point but can encounter issues when scaling horizontally. (Anderson and Nicholson, 2022) Additionally, SQL can be quite complex when dealing with large, complex sets of data, that may be partitioned across multiple servers, leading to slower, more complicated queries. (Jatana, et al, 2012)

### Non-relational Database

In contrast to an RDBMS, a non-RDBMS does not require a set schema to function and is designed to be flexible and optimized for data access depending on the business logic needs of the application. (Tejada, 2023) MongoDB is a document database that stores data as documents, which are like a JSON document or object. (MongoDB (c), 2023) Each document contains the entire data entity in key-value pairs which may be any data type by default. (MongoDB (c), 2023) A document store does not require all documents have the same structure which provides a great deal of flexibility. (Tejada, 2023) While no schema is required of a document database, it may be enforced to ensure accuracy and integrity. Each document is identified with a document key, this is analogous to the primary key in a SQL table. (Tejada, 2023)

## Application design

### Model-View-Controller Frameworks

The Model-View-Controller (MVC) design pattern is commonly used to develop user interfaces, data and business (controlling) logic. (MDN, 2023) In brief, MVC aims to separate user interfaces from the underlying data within an application. (Avraham & Rayfield, 2001). This design pattern has existed since 1979, when it was first described by Trygve Reenskaug developing on Smalltalk-79. (Reenskaug, 1979) An example of an MVC is a search function. The view is the actual search bar and button, the model is the programmed logic that queries for the users search input, and the controller is the database containing the data is returned to the view via the controller to the user. The model defines what data the app should contain, the view defines how the data is presented, and the controller defines how the model and/or view are updated in response to changes in data. (Avraham & Rayfield, 2001)

The three most widely used JavaScript frameworks are React, Angular and Vue. (Vyas, 2022) React, technically a library, uses a virtual DOM, making it lightweight and efficient, but lacks built-in capability for routing, state management and some other features. (Vyas, 2022) Angular and Vue on the other hand are full JavaScript frameworks. (Vyas, 2022) Angular interacts with the al DOM, which makes it slower than Vue or React as the entire tree must be re-rendered on state changes. (Vyas, 2022) Vue is a framework that lies somewhat between React and Angular. (Vyas, 2022) Vue uses a virtual DOM which is lighter than React, and is only 21KB, reducing start-up time. (Vyas, 2022)

### Vue

Vue applications use components to split the UI into independent and reusable pieces, in a way that is very similar to traditional HTML development. (Vue (b), 2023). Components are built from **.vue** files which are Single File Components, or SFCs. Child components must be imported to the parent component to use and are then rendered into the **<template>** area represented in the style of a self-closing, case sensitive HTML tag of the components imported name. (Vue (b), 2023) Vue recommends using PascalCase in tag names of child components, in order to differentiate from native HTML elements which are lowercase. (Vue (b), 2023) Vue components can be understood as three types: global which may be used by the entire application, local – those for use within a single component, and single-file, for instance an SVG file may be rendered as a .vue component, which gives it access to Vue functionality.

Data may be passed to components using props, custom attributes which may be registered to a component. (Vue (b), 2023) Props are declared within components via the *defineProps()* macro. (Vue (b), 2023) There is no limit to the number of props that may be passed to a component and any value can be passed to a prop by default. (Vue (c), 2023). Vue’s virtual DOM uses HTML-style syntax allows DOM elements to bind to underlying component data. (Vue (c), 2023). This binding can be achieved through two methods: text interpolation using “moustache syntax” into raw HTML elements or by binding attributes through the v-bind directive. (Vue (c), 2023). Within these data binding styles, JavaScript expressions may be used directly for conditional data rendering in elements. (Vue (c), 2023)

### Vite

Vite (2023) is a build tool that provides faster development for modern web browsers by using Hot Module Replacement (HMR) and a build command that bundles with Rollup, a modern and widely-used JavaScript module bundler. (Vite, 2023) HMR allows Vite to precisely update pages by only reloading modified components. This reduces data transfer and loading times and preserves application state. (Vyas, 2022) Vite and Vue are created by the same person, Evan You, and Vite was specifically created with Vue in mind, reinforcing the decision to use Vite in this Vue project. (You, 2023)

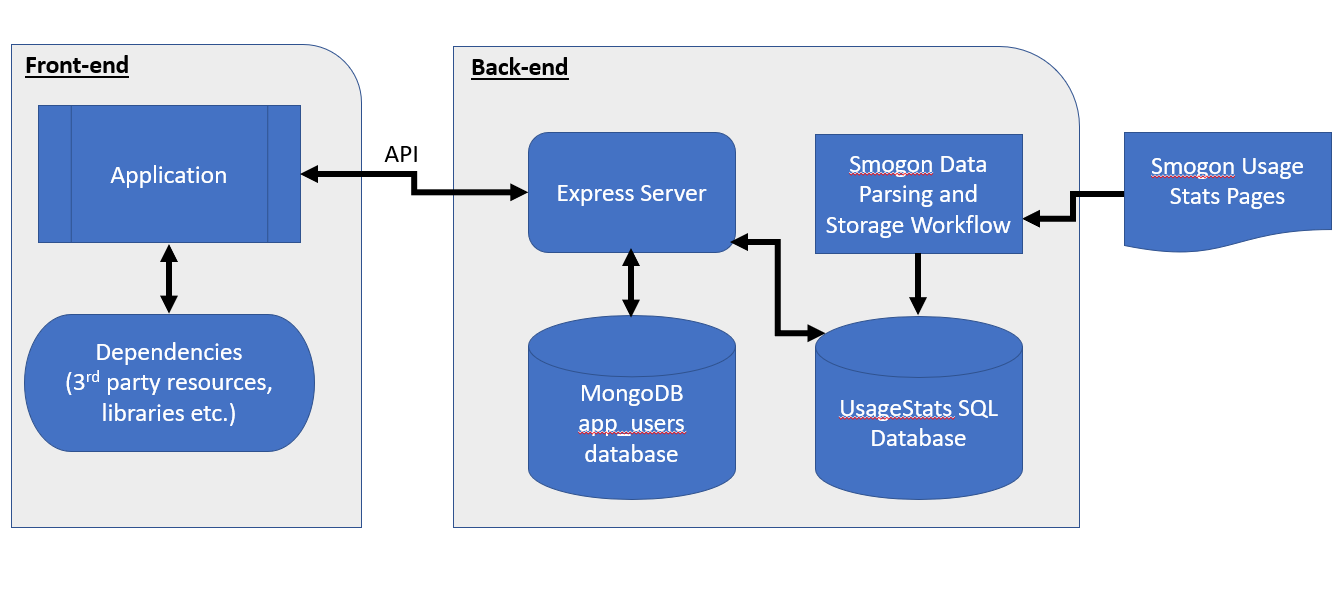
Vite can be compared to other common build tools, such as Webpack, Parcel or esbuild. (Wildermuth, 2022) According to NPM trends (2023), over the last year Vite has been consistently third in downloads, behind Webpack and Rollup. Vite (2023) makes building faster by taking advantage of advancements in the development ecosystem. Webpack’s key features are its customizability and flexibility for a variety of module types and languages. (CodeAcademy, 2023) Webpack can be configured to support a wide array of applications and non-JavaScript languages such as CoffeScript, TypeScript, Babel and Sass. (CodeAcademy, 2023) Webpack’s loaders transform non-JavaScript languages into JavaScript for use in applications, and even allows for writing of custom loaders. (CodeAcademy, 2023) Parcel is a very lightweight, “zero-configuration” bundler, designed to be used right away after installation. Parcel includes HMR, code-splitting and is entry-point agnostic. (CodeAcademy, 2023) As well Parcel has built-in bundlers for CSS, HTML and JavaScript files and is designed with support for a variety of non-JavaScript languages. (CodeAcademy, 2023) Esbuild is a JavaScript bundler designed to be very fast, able to bundle applications at 10 to 100 times the speed of other bundlers, without using caches. (CodeAcademy, 2023) While esbuild is very fast at bundling, it does not include features like HMR or support for other languages. (CodeAcademy, 2023) Vite makes use of esbuild in the background to prebundle dependencies and reduce dev server start times. (Nas, 2023) Vite works by looking only for changes in the virtual DOM the chain and updating those changes with HMR. (Nas, 2023)

### NodeJs

NodeJS is the JavaScript runtime for this project. NodeJS is an asynchronous, event-driven JavaScript runtime, designed to build scalable applications. (NodeJS, 2023). NodeJS was created in 2009 and has been a popular choice for backend developers since its inception and is currently used by industry leaders such as Amazon, LinkedIn and Netflix to power their websites. (Panchal, 2023) According to Panchal 95% of NodeJS developers use databases in their project and around 86% use a front-end framework. As this project uses both, NodeJS is a logical conclusion for a run-time environment.

# Methodology

This project is an exploratory case study the software development lifecycle of a full-stack web-based application from the point of harvesting and processing data to displaying a reactive application on user input. Additionally, it aims to explore the use of SQL and NoSQL databases together a server for more flexible and secure data storage. Data for use in this project was gathered by a web scraping script. User data is automatically generated using mockaroo.com. Data flows from the primary source, through the Python data workflow, and is then made readily available through the RESTful API which is consumed by the application.



Application workflow (arrows indicate flow of data)

Python scripts are used to collect and parse the raw data from the source primarily because of this author’s familiarity using Python and experience as a data scientist. Python is a high-level programming language, currently in version 3.11; in this project, Python 3.10 was used. The Python psycoPG2 library is used in the maintenance and management of the PostgreSQL database. The data scraping and formatting uses mostly builtin tools along with the pandas library. Pandas is a Python data analysis package that uses dataframes to analyse and manipulate relational data. (Pandas, 2023) Dataframes are two-dimensional arrays that organise data in a table-like structure. (Pandas, 2023). NodeJS is used to run the application, as it is one of the most widely used JavaScript runtime environments, powering over 6.3 million websites, at time of writing. (Panchal, 2022)

The use of software design patterns and principles were considered during the development of this project. DRY, or “Don’t Repeat Yourself” is a design principle aimed at reducing the amount of repetitive code within any software application. (Muldrow, 2020) In addition to DRY, this project aims to implement SOLID principles as often as possible. SOLID is a set of 5 principles to support scalability. These principles are: Single Responsibility Principle, Open Close Principle, the Liskov Subsitution Principle, Interface Segregation Principle and Dependency Inversion Principle. (Singh & Hassan, 2015) In brief the single responsibility principle stats there should never be more than one reason for a class to change, if there is more than one reason, it is assumed to have multiple responsibilities and should be restructured. (Singh & Hassan, 2015). When one change to a program causes a cascade of changes to dependent models, it is in violation of the open close principle. (Singh & Hassan, 2015). The Liskov substitution principle states a function that uses a pointer or reference to a base class, must be able to use derived objects of that class without knowing it. (Singh & Hassan, 2015). If a client depends on an interface it does not use, it is in violation of the interface segregation principle. (Singh & Hassan, 2015). The dependency inversion principle requires high-level and low-level modules be separated. (Singh & Hassan, 2015) While these principles might not have noticeable effects at this level of development, they will set a solid foundation for future expansion.

Finally, this project tries to adhere to industry-oriented best practices for code consistency. W3Schools (2023) provides examples of coding conventions which are commonly used by programmers. This includes verbose names for functions and variables to make code easy to read and understand, using camelCase naming for functions and variables. (W3Schools, 2023) Vue 3 does not, at the time of writing, have an official style guide, and the existing Vue 3 style guide is slightly outdated as there were significant changes between Vue 2 and Vue 3 regarding implementation. (Vue (a)., 2023) The Vue 3 documentation examples serve as a template for how this project is structured and formatted. (Vue (b), 2023)

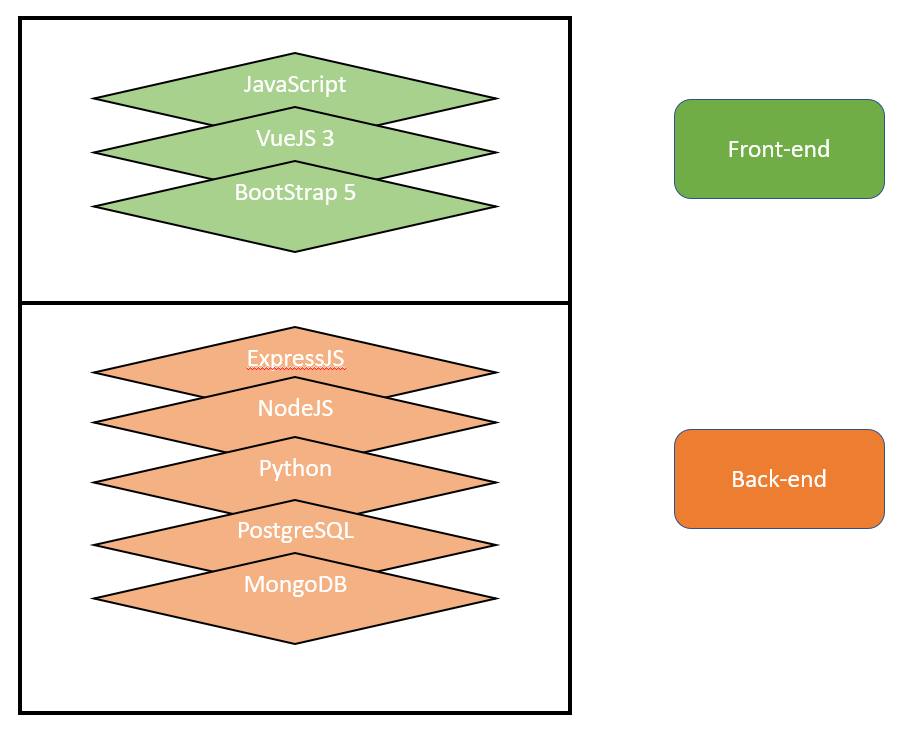
Using Vite as a build tool is suggested by Vue in their official documentation and is used with Vue when using **npm create vue@latest** command. Vite has almost everything necessary to run a Vue project and provides rapid debugging capabilities with HMR’s speed. Vitest is a testing tool that is made to run with and test Vite projects and was thusly selected to run unit tests for this project. Unit tests will not be carried out for this project, and instead testing will be done using ad-hoc testing with fabricated data for the user interface. Postman will be used to test the Express API. Postman is an API platform, a software system with integrated tools and processes to build, manage, publish and consume APIs. (Postman 2003)

# Design

## Determining the Technology Stack

When determining the technology stack for this project, considerations considered were familiarity with the technologies, ease of adoption, utility and performance. For the data collection and organisation pipeline, Python and SQL were chosen as technologies due to previous familiarity with them, as well SQL is ideal for storing table-like data, as the processed data structure stored in CSV format, which is easily converted to SQL tables. (Microsoft, 2023) Another reason for using Python is that has many high-quality data processing-oriented libraries for easily processing data, such as pandas. Express was selected as an API server because of its lightweight overhead and ease-of-use, as well the API does not have complex routing requirements so choosing a technology like express reduces the pitfalls encountered with overengineering solutions.

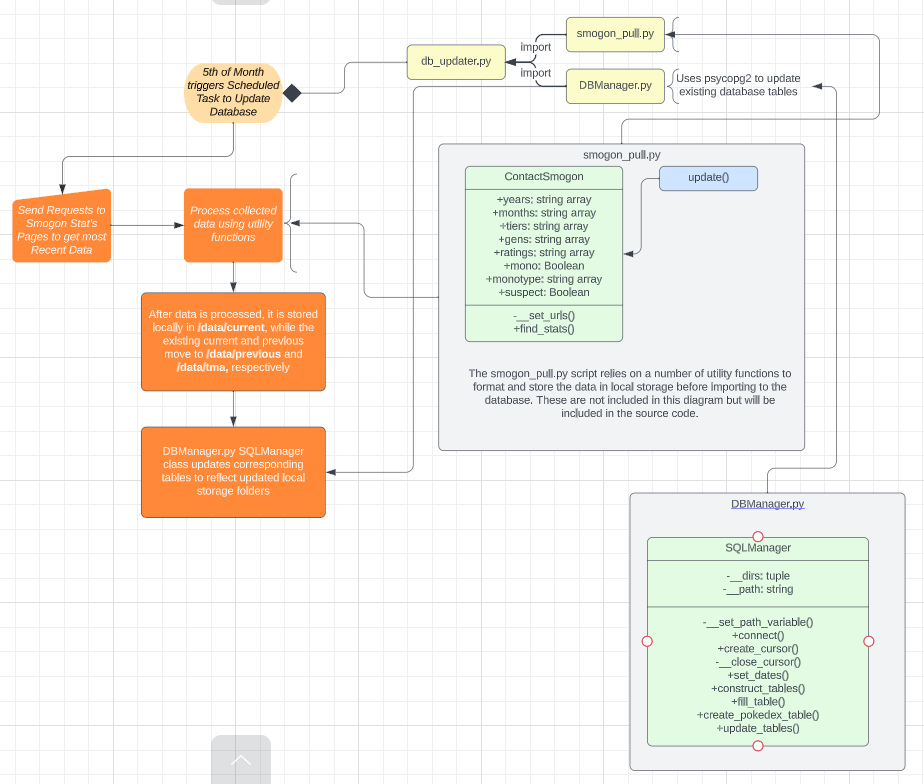
Front-end technologies used include Bootstrap for styling and design and VueJS for displaying content and UI changes. Bootstrap’s extensive API allows for very quick design and provides a modern look, reducing time spent fine-tuning layouts. As this application is expected to be used primarily on personal computers, media breaks for small, portable screens such as smart phones have not been considered but would be valuable additions if mobile use is desired. VueJS was chosen over similar frameworks like React or Angular, primarily due to familiarity and personal preference. VueJS also has a wide array of supporting libraries such as Pinia and Vite which make the development process much easier and allow for data stores, routing and other essential functions which are not included with React. Angular being a TypeScript-, rather than JavaScript-based framework, would require learning nuances of a different language which seemed difficult given the time constraints of this research. NodeJS serves as the run-time environment due to familiarity and its capability for handling front- and back-end management.



The technology stack used for this project, divided into front- and back-ends.

## Data Collection and Access

The data used in this project is usage statistics data generated from an online Pokémon battle simulator. (See Appendix C) In brief terms, it is data for how often a Pokémon was used during a one-month period by users. This data is stored in a directory of publicly available web pages and presented as stylized plaintext format. (See Appendix C) A set of scripts written in the Python programming language is used to remove the formatting and store the data into a large collection of comma-separated values (CSV) formatted documents, which are stored locally to be ingested into PostgreSQL tables. Three tables will be available for the application, which are **current**, **previous**, and **tma**, which is an abbreviation for “two months ago” and is accessible through the **older** branch in the server API (see below). The database needs be manually updated in the first week of every month, which will roll off the oldest of the three tables and update accordingly. Although in this project it is not, this task would benefit from implementation of task automation.



In order to access to this data from the database, it must be called from a server. This application uses an Express server, because it is lightweight and easy to setup and use, and a widely used JavaScript server library. There are three routes in the server, **current**, **previous**, and **older**. These allow access to their respective tables, and search in the **tier** column to match the data from the team which is input into the form.

## Content Display

The user-interface of the application is created using Vue3, a JavaScript framework that uses its own syntax to develop applications. Vue separates out the logic of the application into component templates. [Write more about Vue3]

The presentation and layout is done primarily through Bootstrap, a CSS framework geared towards fast layout development. Bootstrap was developed and made open-source in 2011 by an internal development team at Twitter and has been widely used since. Bootstrap uses a set of pre-made CSS class names to set layout and design for a web application. This project also uses some custom CSS properties for styles that are not available with Bootstrap, for instance Bootstrap’s height and width can be set at 25, 50, 75 and 100% marks, any other width must be set with custom CSS or Bootstrap’s own custom input API.

There are four views for the user to interact with: the landing page, the application page (**/main**), the login page (**/login**) and the profile page (**/profile**). The application page is found at the **/main** path and serves the core application. Within this page is a form page, where the user inputs their team and information related to the team, and two displays which display the team in a card-like format and three outputs of usage statistics below that.

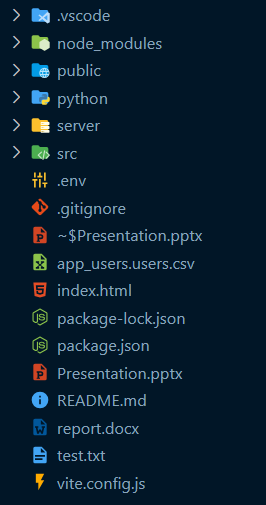
## SQL - NoSQL Interaction

This project makes use of a NoSQL database using MongoDB to supplement the user information stored in the SQL database. The **users** table in the SQL database contains id, username, password, mongo\_id and role columns, which are used for authentication at login, because SQL is faster than MongoDB for these types of functions. The mongo\_id column corresponds to the **\_id** field in the MongoDB **app\_users.users** collection. The objects in this collection contain data relating to the user’s profile for any additional information a user may wish to provide, such as date of birth, location, or any websites they wish to promote, in addition to any teams they have saved. MongoDB and NoSQL databases are ideally created for this type of data storage which has a variable data structure.

# Implementation

## Project Setup

Due to the nature of how NodeJS projects are setup to run, and to keep configuration as simple as possible, the root directory contains all the config files necessary to setup and run the application. The Python folder contains all the requisite files and folders to manage the data collection and storage process. The rest of the folders in the root directory are related to the Vue application. The design of a Vue application divides the abstract idea of an application into views, what is presented to the user, and components, elements of the page that make up the view. For an example of a view, consider e-commerce application, the store and checkout pages would be considered views while each the items for sale, the display of the items, and filtering options would all be components. The developer can choose how much to compartmentalise each view’s components. The Express server and database setup files can be found in the server folder.



Project directory layout.

## Data collection and storage

The Pokémon Showdown usage stats data is analysed, parsed and stored on smogon.com’s stats page, (See Appendix C) which has several branches. These branches are broken down by month, and within each month are several branches as well. These root branches are the area of interest for this application at the moment. Inner branches include moveset data in the /moveset/ branch, the various playable tiers, which does change with each new game release, and other data about the metagame in the /metagame/ branch, separate branches for monotype teams in the /monotype/ branch, and data about specific Pokémon under the /chaos/ branch. This project is focused only on the usage stats found in the root branch for each month.

The data is available at approximately the third day of each month, as it takes around 44 hours to process and analyse the data (Antar). To account for any errors in the data processing, the python updating script is run via the Windows Task Scheduler to run on the 5th of each month. The python script takes a couple of hours, due to the sheer amount of data scraping, and the implementation of a sleep timer of three seconds after each call to prevent throttling or being blacklisted from attempting to contact the raw data too fast.

The script smogon\_pull.py contains the Contact\_Smogon class, this class handles all the web scraping and data sanitization. There are five required inputs for the Contact\_Smogon, which must be lists of string values, for years, month, gens, tiers and ratings. Years and months are self-explanatory, they must be lists of the months or years that the script wants to query. November 2014 is the earliest available data, queries before that time will not yield results. The month must also be at least one month prior to the current month, for example if the current month is June, data will only be available for May and before, due to the nature of how the data is collected. The gens parameter refers to the generation, which is a stringified integer from 1 to 9, as of the time of writing. Each new release of a game in the main game series is considered a new generation, for instance, when Pokémon Scarlet and Violet were released in 2022, that marked the addition of generation 9. Tiers parameter contains the tiers of interest, such as OU (overused), which is considered the “standard” tier. (Antar) Ratings are a binary choice of either 1695 for OU tiers and 1630 for non-OU tiers. This is due to OU being the de-facto standard and most-played tier and is easier to get a rating of 1630 vs 1695 in OU. (Antar)

After processing and cleaning the data, each dataset is stored into a CSV file which is stored locally in the /data/csv directory. The CSV files are named according to their target from the web scraping, i.e. the June 2022 gen 8 OU stats are stored as “2022-06\_gen8ou.csv”. The three most recent set of stats collected are stored in the current, previous and tma folders, which are subsequently stored in their respective database tables. Database updating is handled with the DBManager.py script, which performs the rolling storage of said data. Due to the use of PostgreSQL and Python, this script uses the psycopg2 library to handle interactions with the database. Psycopg2 was chosen over other SQL libraries because of the author’s previous experience with it. To perform updates of the database and store the newest datasets, the db\_updater.py script runs the **update** method from the Contact\_Smogon class, as well as the table management functions from the SQLManager class in DB\_Manager.py.

The python scripts use a venv, or virtual environment, which keeps installed libraries required for this suite of scripts to run separate from globally installed variables and prevents conflicts in different library versions, which may have breaking changes in functionality and cause errors during execution. Additionally, using a .env file keeps login details for the database for additional security.

## Server Architecture

The server to access the data is setup using ExpressJS a lightweight, JavaScript server library. The server provides three GET routes to the three tables of the latest three months of data and has two POST routes for login and registration validation. The table routes are used to find the usage stats for the tier of the team the user selects during team input and form submission. Tiers are passed as parameters, which are selected from the submission form.

### Routing

There are three GET routes for the current API. These routes all function the same and will be explained together. The only differences are the data being retrieved. The get routes go to current, previous and older branches and have a parameter of **tier**. The tier is the generation and metagame the user’s team is currently competing in. If a team is meant for the “generation 9 overused” tier, the user should select those from the dropdown menu provided in the team submission form. These tiers are used to check in the corresponding SQL tables in the UsageStats database by using the **node-postgres** (pg) library as a driver. Queries are handled with parameterised queries, which are more secure and considered best-practice by the pg documentation. This provides protection against SQL-injection attacks, by passing the query text unaltered as well as the parameters which are substituted with rigorous parameter substitution code in the server. This method also safely converts *undefined* parameters as *null*. (node-postgres)

The API also has two POST routes for user registration and login. These routes receive form data from their respective forms. Kushawa (2023) recommends that to protect password security, all passwords in the SQL database are store as hashes using the **bcrypt** library. This protects users’ passwords in the case of a security breach by not storing them as plaintext. Bcrypt uses a hash and salt technique to cryptographically encode passwords. These passwords are then stored in the password column of the users table. By encrypting the password, attackers are unable to obtain the plaintext password, protecting users who may use the same login credentials across several applications. These passwords can still be obtained if an attacker gains access, but requires substantial effort to crack the encryptions, giving time to alert users to reset their passwords. If the validation check is passed during the login phase, the user’s profile information is retrieved from the Mongo app\_users.users collection. A token is generated and signed, authenticating the login and sending user information in the response object.

Registration is accessed through the register route, which like login sends form data after submission. The submitted password is hashed using bcrypt, and a check is performed to see if the chosen username already exists. If it does exist, a 400 (failure) response is sent. If the check passes, the is added to the mongo collection first, to generate the mongo id. After the user is entered into the mongo collection, the ID along with username and the hashed password are entered into the SQL database and a 201 status is sent in the response object.

## State Management

During a user’s session in the application, Pinia stores are used to manage data that is used throughout the application. A store is like the PHP session variable, it holds data that can be modified or updated by the user during their time in the application. In this application, there are stores for the user’s team, the profile information, authentication for login, and the Smogon usage stats. All stores are initialised to have a state property with default values.

### TeamStore

The TeamStore manages the user’s team input in the application. The team is parsed from input using the koffing library, a JavaScript library created to parse Pokémon Showdown teams, and enriched by adding types to the moves for colour coded presentation. Types in this case refers to the Pokémon types such as fire, water, grass, etc. The team store saves the user team in the state object, which is rendered through the display after form submission. In future versions of the project, users can also save their teams, which are stored in **the app\_users.user.teams** field.

### UsageStore

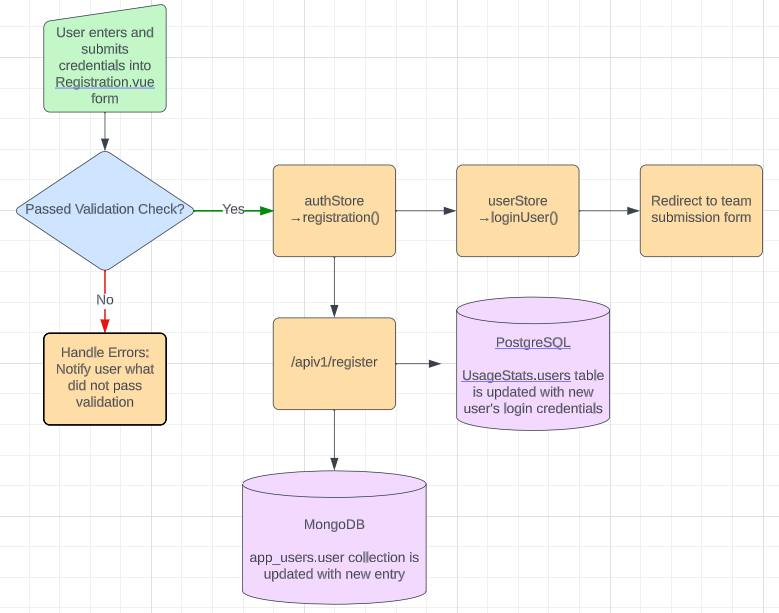
The UsageStore retrieves the usage data available for the tier of the user-submitted team, which is selected via a dropdown menu in the form area. This data is fetched from the local database through the server and displayed in the bottom half of the screen, providing the user with context for their current team performance in the current metagame.

### UserStore and AuthStore

The userStore and authStore work to manage user information and validate login and registration functionality from the user interface. The userStore contains the user’s profile information, as well as any saved teams, which are stored in a MongoDB instance. AuthStore manages authorization for login and registration storing the signed user token and enabling login if a successful response was reached from the API. The userStore is called by the authStore during login and registration, to set the user data after login and make accessible throughout the application.

## Login and Registration

The authStore is a sort of middleman between the user interface and API for login and registration. The authStore receives objects of supplied user credentials and sends them through a header to the API’s login and register routes. When the API is queried and supplied with body headers, these headers are destructured to get the credentials. All passwords for registration are encrypted with bcrypt before being stored in the SQL database, to protect user passwords from being easily identified by potential attackers. If a login is successful, the user is redirected to the main page. In the registration component, Yup, a JavaScript library used for form field validation is used to ensure that parameters and requirements for each field are correct, if the check passes, registration continues through the authStore and then the API. User login does not persist if a user navigates away from the application or refreshes, because this application does not make use of cookies to track login.

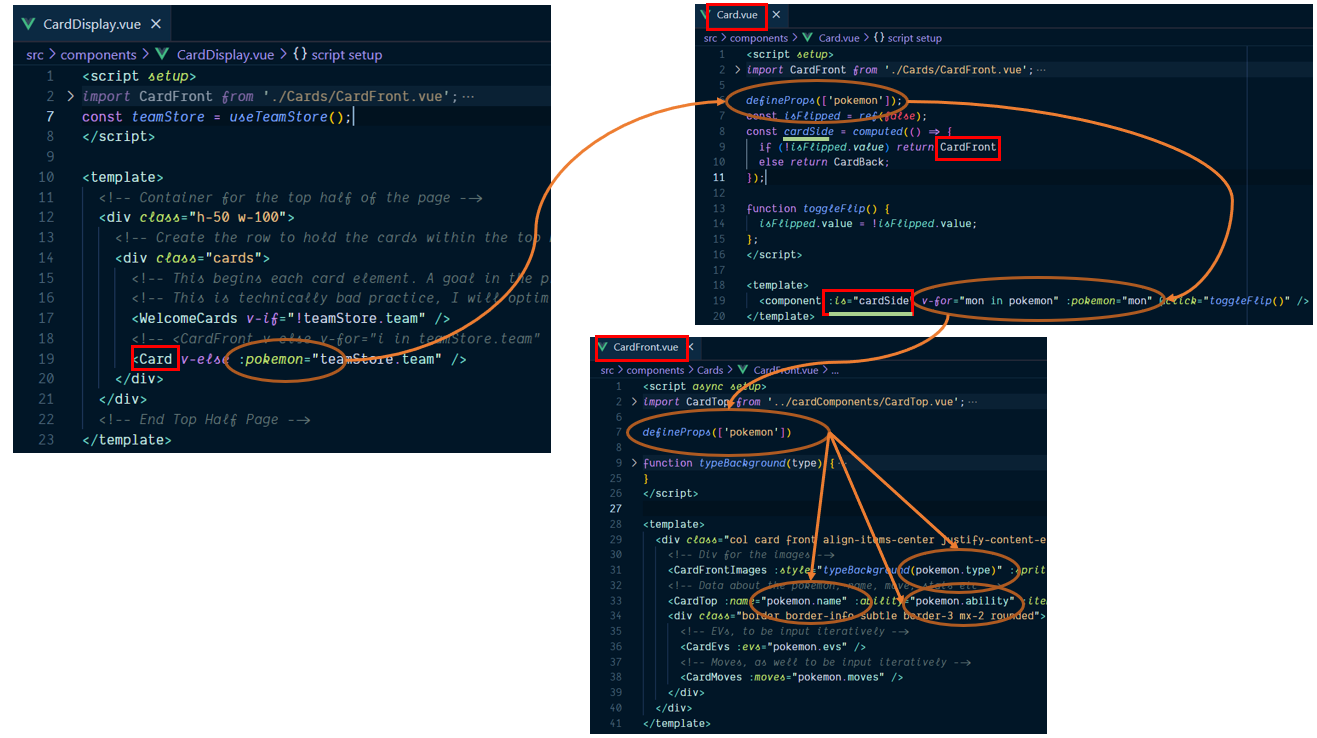


User registration workflow. Login is very similar, authStore uses *login()*, rather than *registration()* and does not update the databases.

## User Interface and Interaction

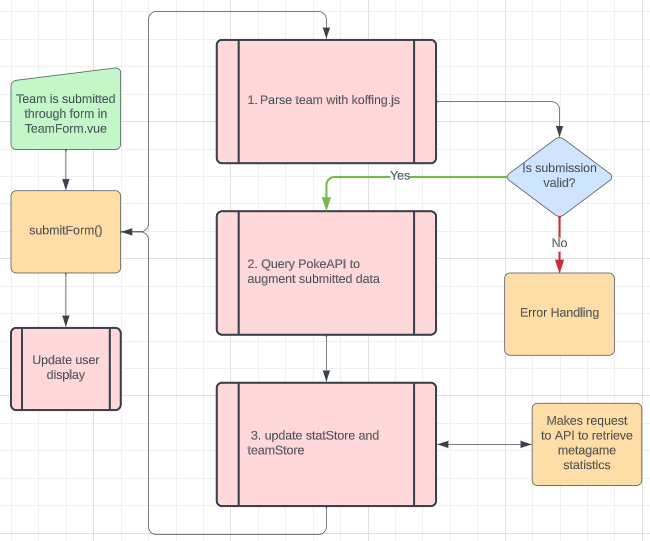
The main page, path ‘/’, is a welcome page for a user. It contains legal disclaimers and a broad overview of the application. A button link on the bottom directs the user to the main part of the application, path /main/, where all the functionality takes place. On the main page, a form on the right side contains a handful of inputs and a large textarea to paste the user team, exported from Pokémon Showdown. Once the form is submitted, the business logic parses the team and displays a set of cards along the top half of the page. These cards contain information and images corresponding to the data of the submitted team. A navigation bar at the top of every page provides user navigation to different pages in the application, such as returning to the home page or logging in via the login page. The interface is designed to be simple and intuitive to navigate, using large green buttons for submission or accept buttons and red for cancel or reset buttons.

Vue (2023) creates web pages through use of components each of which contains at the least a <template> tag, where the HTML is written. Optional are the <script> and <style> tags containing, respectively, JavaScript logic and CSS-styling. Values can be accessed in the <template> areas by use of binding either through “moustache” syntax or using vue’s binding syntax. (Vue 2023) In this project, each part of the user interface is broken down into a component, and props are passed through to each component and sub-component as required after team submission. Some components, such as Login.vue and Registration.vue, do not receive props as they are components separate from the main application.



How props are passed to components and subcomponents in the application.

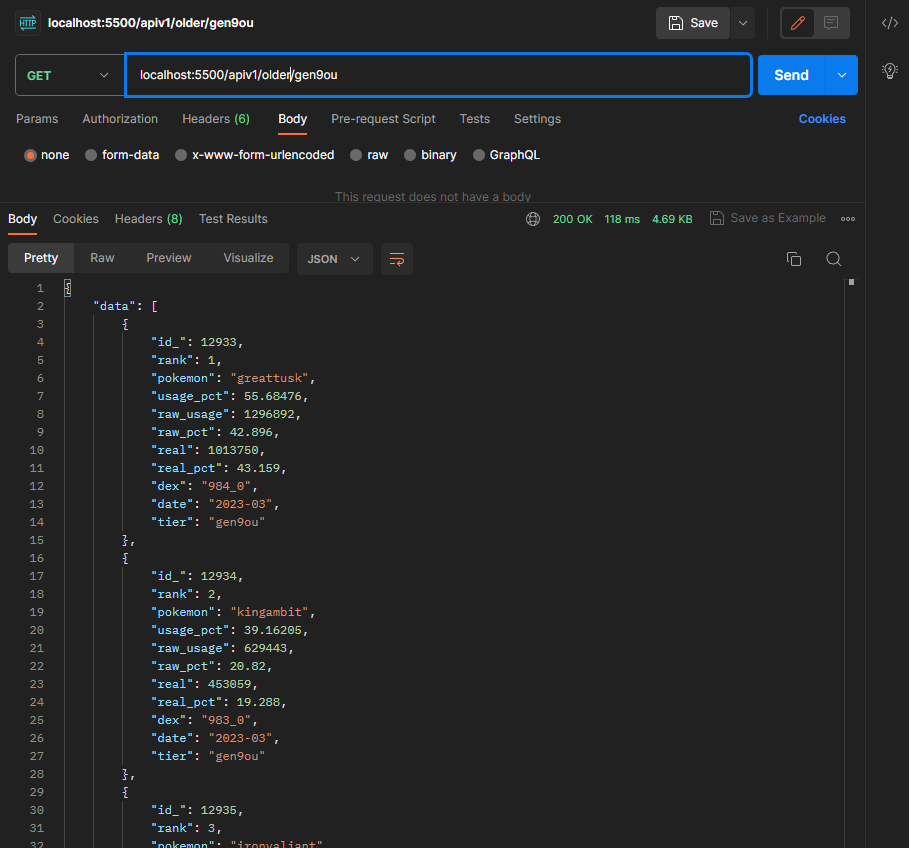
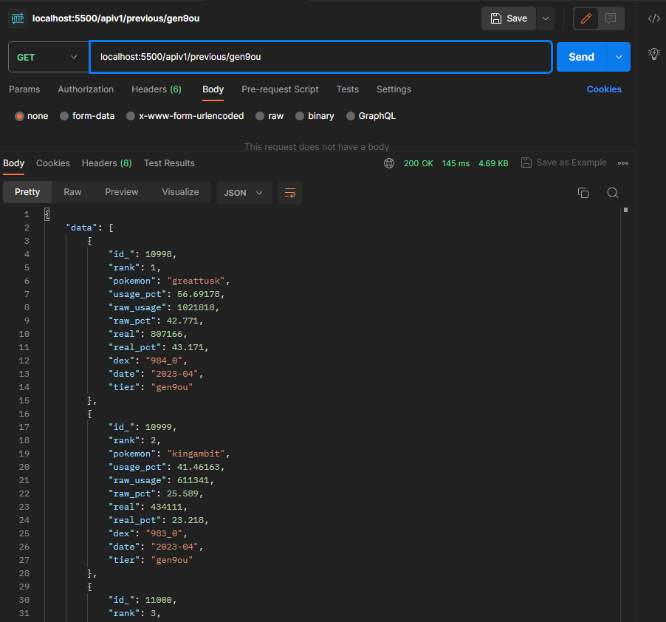
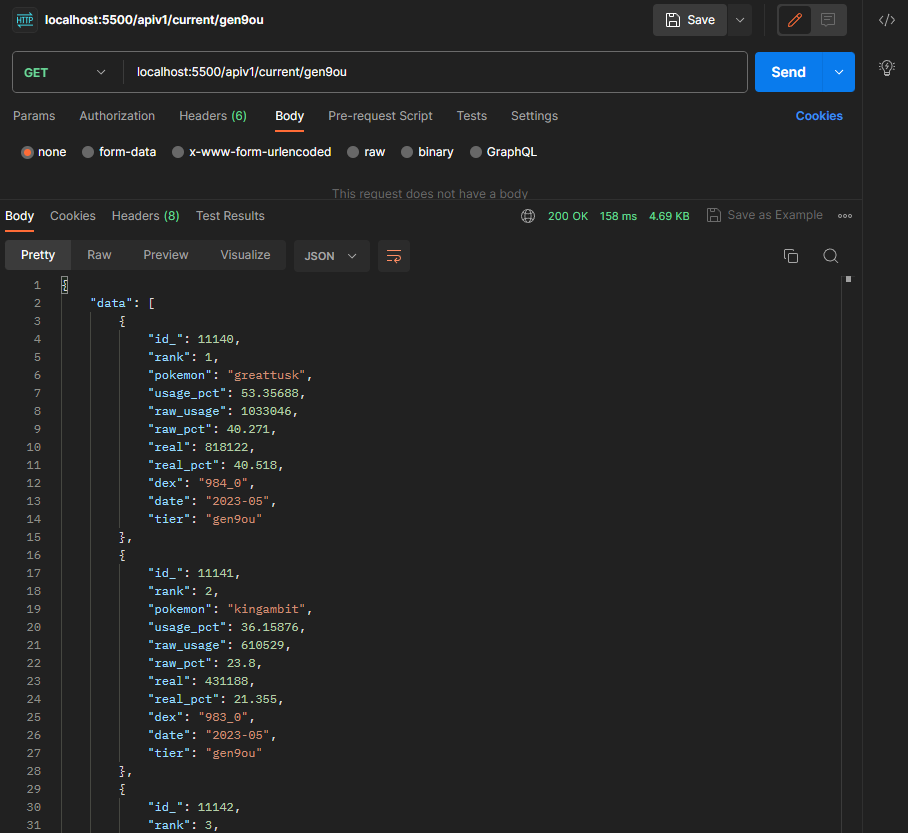
The TeamForm.vue component is where the business logic takes place to parse and store the user’s submitted team. After submission, the koffing library (See Appendix C) transforms the plaintext submission into a team object. Elements of this object are enhanced by performing queries against the pokedex-promise-v2 library (See Appendix C), which is a JavaScript wrapper for PokeAPI and reduces strain on the API by handling requests more optimally. If a team is successfully parsed, it is stored in the teamStore, in accordance with the official Pinia (2023) documentation. The statStore is also called upon to query the local database and find the usage stats data for the supplied metagame. Once the teamStore is updated, the upper left half of the display is updated to reflect the submitted team, and the bottom left half reflects the usage stats ranking of the metagame of interest.



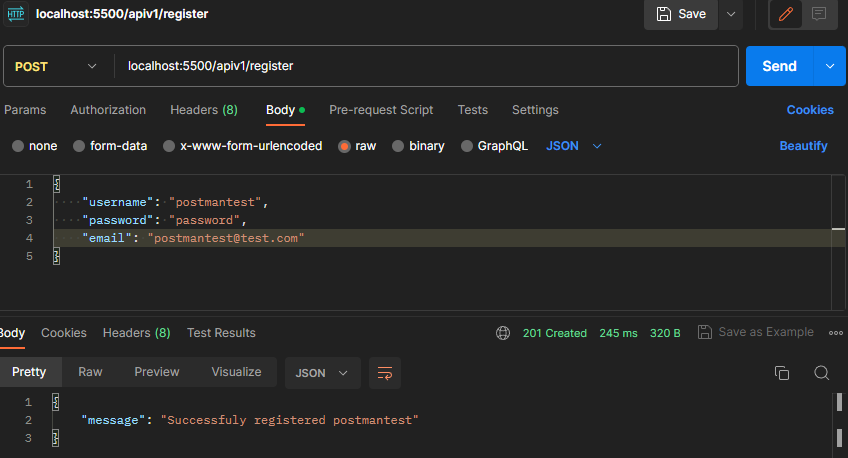
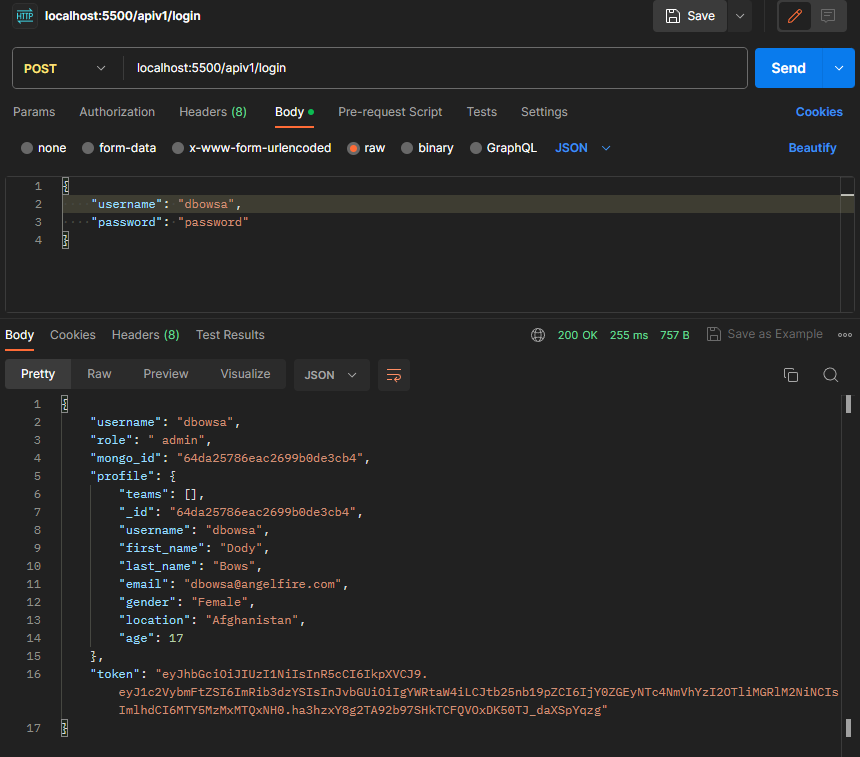
Front-end workflow overview

# Testing

Testing for this project was handled with multiple testing methods. During initial production, testing was done ad-hoc, to ensure components were displaying expected values and manipulating data in expected ways. To test the API, Postman, a third-party API testing application, was used to ensure that routes were succeeding or failing, based on values passed to router endpoints. All endpoints work and display results as expected.



Successful Postman test of ‘/apiv1/current/gen9ou’, ‘/apiv1/previous/gen9ou’ and ‘apiv1/older/gen9ou’ Express routes.



Successful Postman tests of the ‘/apiv1/login/’ and ‘apiv1/register’ Express routes.

Unit testing was not implemented for this project, due to time constraints and scope. Given more time, however, unit testing would be done with Vitest, a testing suite created to work seamlessly with Vite applications. Vitest works by testing functions and Vue components, which take in parameters or props, respectively, and comparing the output to the expected value. If the output and expected values match, the test is successful. Vitest has support for common web patterns and features like glob imports and SSR (server-side rendering) has many plugins and integrations. Compared to Jest, another common testing suite, Vitest was built to be a drop-in replacement for Jest, in most cases, and has similar syntax and a similar API. (Vitest) To test Vue components, Vitest requires the **vue@test-utils** library, which allows components to be imported to a Vitest script and allows for passing of props and other variables into components.

# Discussion

This project created a data pipeline that cleans, modifies and stores data into a PostgreSQL database, makes it available to an application through a REST API which responds to user input. This application protects users by storing encrypted passwords, and separating login credentials and user information. State management, using Pinia’s stores, allows data entered by the user to be made accessible throughout the use of the application. Using Bootstrap, the application has a somewhat modern, yet simple appearance and intuitive interface. The project has room for improvements in the future, discussed below. As this was developed in a constrained timeline, some features that are may be considered standard for modern web applications, such as “dark mode” toggle were unable to be implemented. The core functionality objectives have been met. This application provides some measure of cyber security by storing only encrypted and salted passwords in the PostgreSQL database, and compartmentalizing user information by storing it in a separate MongoDB. By meeting these objectives, the aims of understanding creating a data pipeline, effectively and efficiently creating

# Conclusion

In conclusion, this project explored the lifecycle development of a full-stack application, from raw data collection to front-end user interface delivery. By using a technology stack of Python, PostgreSQL, Express, NodeJS, MongoDB, Bootstrap and Vue, this project delivered:

1. A Python-managed data pipeline that collects, parses, modifies and stores raw information into meaningful and useful data into an SQL database.
2. An Express RESTful API server connects to the SQL database and makes the data available through get requests.
3. The Vue + Bootstrap front-end application allows users to paste in their team text from an outside source and compare their team members’ usage across a three-month period.
4. Users can register and login to the application. User data is stored in a MongoDB instance, separate from the SQL database.

## Future Considerations

There are several areas where this project could be made to be improved or expanded. One improvement would be to increase the speed at which the processing of the user team submission takes place. Currently, on a fresh visit, it takes between five (5) and eight (8) seconds on average to process the data and return a modified view to the user. This is due to the design of the submitForm() function in the TeamForm.vue file. It makes several asynchronous calls through both the pokedex-promise-v2 package and the TeamStore and StatStore Pinia stores. This causes latency due to the nature of asynchronous requests. Batch calling could help to reduce latency, as it does not wait for one promise to be fulfilled before moving onto the next request, by using the Promise.all() method, or querying by a list of values in the pokedex-promise-v2 lookup requests, rather than iterating in a series of for loops.

A feature that was desired but unable to be implemented was user profiles, where users could send and receive messages to each other, store personal information such gender, age or location and save their teams. However, due to time constraints this was unable to be implemented. It would require additional Express routes, an inbox system, and functionality to retrieve the saved teams from within the submission form.

Cyber security is always a concern, and certainly no web application can be fully protected against all threats. Measures could be put in place to increase cyber security such as preventing bots from registering by sending confirmation emails to the user’s supplied email address, requiring completion of captcha’s on registration, or by using two-factor authentication to further bolster site security. These features were deemed unnecessary for the scope of this project, as it is not hosted on a live server, but would be under consideration if this were to change.

Jira or a similar task management tool could have been implemented better. At times during development, production could have been smoother and more organised, which would have reduced wasted time. Time management in general could have been made better with better task breakdown and more detailed sub-tasking. In a real-world situation, this would be managed in a SCRUM or an alternative planning and tasking method.

# References

Anderson, B., Nicholson, B., 2022. *Explore key differences between SQL and NoSQL databases and learn which type of database is best for various use cases.* [online] Available at: <https://www.ibm.com/blog/sql-vs-nosql/> [Accessed 20 Jun 2023]

Antar., 2023. *Everything You Ever Wanted to Know About Ratings*. Available at: <https://www.smogon.com/forums/threads/gen-9-smogon-university-usage-statistics-discussion-thread.3711767/> [Accessed 23 Jun 2023]

CodeAcademy., 2023. *Comparison of Build Tools.* [online] Available at: <https://www.codecademy.com/article/comparison-of-build-tools> [Accessed 8 Jul 2023]

Eddie, J., 2022*. FrontEnd vs. BackEnd vs. Full Stack Development: Choose The Right Stack*. [online] Avaialable at: <https://medium.com/geekculture/frontend-vs-backend-vs-full-stack-development-choose-the-right-stack-bb24799721b2> [Accessed 29 Jun 2023]

Harrington, J., L., 2016. *Relational database design and implementation*, 4th Ed. Morgan Kaufmann. pp. 47-60. [Accessed 9 Jul 2023]

Heap., 2023. *What is a Tech Stack: Examples, Components, and Diagrams.* [online] Available at: <https://www.heap.io/topics/what-is-a-tech-stack#why-is-getting-your-tech-stack-right-so-crucial> [Accessed 6 Jul 2023]

Kushawa, N., 2023. *Best practices for storing passwords securely in a database*. [online] Available at: <https://www.learncsdesign.com/best-practices-for-storing-passwords-securely-in-a-database/> [Accessed 16 July 2023]

Leff, A., Rayfield, J., 2001. *Web-Application Development Using the Mode / View / Controller Design Pattern.* IEEE. [e-journal] Available at: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=950428> [Accessed 6 Jul 2023]

Lucidchart., 2023. *Database design best practices*. Available at: <https://www.lucidchart.com/blog/database-design-best-practices> [Accessed 22 Jun 2023]

Jatana, N., Puri, S., Ahuja, M., Kahturia, I., Gosain, D., 2012. A Survey and Comparison of Relational and Non-Relational Database*. International Journal of Engineering Research & Technology (IJERT)* [e-journal] 1 (6) pp, 1-5. Available at: <https://www.ijert.org/research/a-survey-and-comparison-of-relational-and-non-relational-database-IJERTV1IS6024.pdf> [Accessed 2 Jul 2023]

Microsoft., 2023. *Database Design Basics.* Available at: <https://support.microsoft.com/en-gb/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5> [Accessed 24 Jun 2023]

MongoDB (a)., 2023 *Full Stack Development Explained.*Available at: <https://www.mongodb.com/languages/full-stack-development> [Accessed 6 Jul 2023]

MongoDB (b)., 2023. *What is a Tech Stack and How Do They Work?* Available at: <https://www.mongodb.com/basics/technology-stack> [Accessed 6 Jul 2023]

MongoDB (c)., 2023. *NoSQL Database Design & Data Modeling.* [online] Available at: <https://www.mongodb.com/nosql-explained/data-modeling> [Accessed 7 Jul 2023]

Muldrow, L., 2020. What is DRY Development?. [online] Available at: <https://www.digitalocean.com/community/tutorials/what-is-dry-development> [Accessed 11 Jul 2023]

Nas, E. 2023. *Vite: Webpack killer or something else?* [online] Available at: <https://medium.com/trendyol-tech/vite-webpack-killer-or-something-else-87019b4aeca2> [Accessed 15 Jul 2023]

NpmTrends., 2023. *Parcel vs Rollup vs Snowpack vs Vite vs Webpack.* [online] Available at: <https://npmtrends.com/parcel-vs-rollup-vs-snowpack-vs-vite-vs-webpack> [Accessed 13 July 2023]

Panchal, B., 2023. *Node.js Usage Statistics: What They Tell Us About the Future of Web Development*. Available at: <https://radixweb.com/blog/nodejs-usage-statistics> [Accessed 25 Jul 2023]

Pandas., 2023. *Package overview.* [online] Available at: <https://pandas.pydata.org/docs/getting_started/overview.html> [Accessed 22 Jul 2023]

PostgreSQL., 2023. *PostgreSQL 15.4 Documentation*. Available at: <https://www.postgresql.org/docs/15/index.html> [Accessed 21 Jun 2023]

Postman., 2023 *API Platform.* [online] Available at: <https://www.postman.com/api-platform> [ Accessed 17 Jul 2023]

Stevens, E., 2022*. 7 fundamental UX design principles all designers should know.* [online] <https://www.uxdesigninstitute.com/blog/ux-design-principles/> [Accessed 21 Jun 2023]

Shropshire, J., Landry, J. P., Presley, S. S., 2018. Towards A Consensus Definition of Full-Stack Development. SAIS 2018 Proceedings. [e-journal] pp. 1-6. Available at: <https://aisel.aisnet.org/sais2018/17> [Accessed 15 Jun 2023]

Tejada, Z., 2023. *Non-relational data and NoSQL*. Available at: <https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data> [Accessed 3 Jul 2023]

Vayas, R., 2022. Comparitive Analysis of Front-end Frameworks for Web Applications. *International Journal for Research in Applied Sciences & Engineering Technology (IJRASET)*. [e-journal] 10 (7), pp. 298 – 307) Available through: <https://www.ijraset.com/research-paper/comparative-analysis-on-front-end-frameworks-for-web-applications> [Accessed 2 Jul 2023]

Vite., 2023. *Getting Started.* [online] Available at: <https://vitejs.dev/guide/> [Accessed 29 Jun 2023]

Vue (a)., 2023. *Style Guide.* [online] Available at: <https://vuejs.org/style-guide/> [Accessed 11 Jul 2023]

Vue (b)., 2023 *Component Basics* [online] Available at: <https://vuejs.org/guide/essentials/component-basics.html> [Accessed 11 Jul 2023]

Vue (c)., 2023 *Template Syntax.* [online] Available at: <https://vuejs.org/guide/essentials/template-syntax.html> [Accessed 11 Jul 2023]

U.S. General Services Administration (USGSA), 2006. *User Interface Design Basics.* [online] Available at: <https://www.usability.gov/what-and-why/user-interface-design.html> [Accessed 14 Jul 2023]

W3Schools., 2023. *JavaScript Coding Conventions.* [online] Available at: <https://www.w3schools.com/js/js_conventions.aspw> [Accessed 27 Jun 2023]

Wildermuth, S., 2022. *Vite vs Webpack.* [online] Available at: <https://www.vuemastery.com/blog/vite-vs-webpack/> [Accessed 29 Jun 2023]

You, E., 2023. *Hello, My name is Evan You.* [online] Available at: <https://evanyou.me/> [Accessed 8 Jul 2023]

# 

# Appendix

## A. Source code

The source code for this project can be found at <https://www.github.com/stu-gotz/pgmp>.

## B. Ethics Approval

## C. Third-Party Resources

<https://pokeapi.co/> The PokeAPI and its JavaScript wrapper pokedex-promise-v2 (<https://github.com/PokeAPI/pokedex-promise-v2>) are vital components of this application, used to enhance aspects of the user submitted information.

<https://github.com/itsjavi/koffing> The koffing JavaScript library by itsjavi is also an important library for this project, as it is made to parse the expected user information for this application. Without this library, it would require creating a collection of scripts to parse the user input data.

<https://www.mockaroo.com/> mockaroo.com, a website for creating filler data for testing, was used to generate user data for testing the login and registration components.

<https://www.smogon.com/stats/> is the resource for the raw PokémonShowdown usage stats data. This is the where data is collected to be modified and ingested into the database.

[https://play.Pokémonshowdown.com/](https://play.pokemonshowdown.com/) is the Pokémon battle simulator that the raw data is generated from by the Smogon community. More information may be found at their GitHub repository for the application: [https://github.com/smogon/Pokémon-Showdown](https://github.com/smogon/Pokemon-Showdown).