

Software Design Document

for

Visulyfe

Version 1.0 approved

Prepared by Adam Dixon, Fernando Serrano Perez, Geovanny Montano, Jacob Hertz, Nicholas Trigueros, Ryan Torrez, Steven Partida, Washika Afrozi

Visual Corp.

03/19/2024

Table of Contents.....	pg # 2
Revision History.....	pg # 7
1. Introduction.....	pg # 8
1.1. Purpose.....	pg # 8
1.2. Document Conventions.....	pg # 8
1.3. Intended Audience and Reading Suggestions.....	pg # 8
1.4. System Overview.....	pg # 9
Design Considerations.....	pg # 10
2.1. Assumptions and dependencies.....	pg # 11
2.2. General Constraints.....	pg # 11
2.3. Goals and Guidelines.....	pg # 12
2.4. Development Methods.....	pg # 12
Architectural Strategies.....	pg # 13
System Architecture.....	pg # 15
4.1.	pg # 15
4.2.	pg # 15
4.3.	pg# 17
Policies and Tactics.....	pg # 19
5.1. Specific Products Used.....	pg # 19
5.2. Requirements traceability.....	pg # 19
5.3. Testing the software.....	pg # 19
5.4. Engineering trade-offs.....	pg # 19
5.5. Guidelines and conventions.....	pg # 19
5.6. Protocols.....	pg # 19
5.7. Maintaining the software.....	pg # 19
5.8. Interfaces.....	pg # 19
5.9. System's deliverables.....	pg # 20
5.10. Abstraction.....	pg # 20
Detailed System Design.....	pg # 21
6.1 Front-End Module.....	pg # 21
6.1.1 Responsibilities.....	pg # 21
6.1.2 Constraints.....	pg # 21
6.1.3 Composition.....	pg # 21
6.1.4 Uses/Interactions.....	pg # 22
6.1.5 Resources.....	pg # 22
6.1.6 Interface/Exports.....	pg # 22
6.2 Back-End Module.....	pg # 23
6.2.1 Responsibilities.....	pg # 23
6.2.2 Constraints.....	pg # 23
6.2.3 Composition.....	pg # 24
6.2.4 Uses/Interactions.....	pg # 24
6.2.5 Resources.....	pg # 24
6.2.6 Interface/Exports.....	pg # 25
6.3 Database Module.....	pg # 25
6.3.1 Responsibilities.....	pg # 25
6.3.2 Constraints.....	pg # 26

6.3.3	Composition.....	pg # 26
6.3.4	Uses/Interactions.....	pg # 27
6.3.5	Resources.....	pg # 27
6.3.6	Interface/Exports.....	pg # 28
Detailed Lower level Component Design		
7.1	about.html.....	pg # 29
7.1.1	Classification.....	pg # 29
7.1.2	Processing Narrative(PSPEC).....	pg # 29
7.1.3	Interface Description.....	pg # 29
7.1.4	Processing Detail.....	pg # 29
7.1.4.1	Design Class Hierarchy.....	pg # 29
7.1.4.2	Restrictions/Limitations.....	pg # 29
7.1.4.3	Performance Issues.....	pg # 30
7.1.4.4	Design Constraints.....	pg # 30
7.1.4.5	Processing Detail For Each Operation.....	pg # 30
7.2	base.html.....	pg # 30
7.2.1	Classification.....	pg # 30
7.2.2	Processing Narrative(PSPEC).....	pg # 30
7.2.3	Interface Description.....	pg # 30
7.2.4	Processing Detail.....	pg # 30
7.2.4.1	Design Class Hierarchy.....	pg # 30
7.2.4.2	Restrictions/Limitations.....	pg # 30
7.2.4.3	Performance Issues.....	pg # 31
7.2.4.4	Design Constraints.....	pg # 31
7.2.4.5	Processing Detail For Each Operation.....	pg # 31
7.3	home.html.....	pg # 31
7.3.1	Classification.....	pg # 31
7.3.2	Processing Narrative(PSPEC).....	pg # 31
7.3.3	Interface Description.....	pg # 31
7.3.4	Processing Detail.....	pg # 31
7.3.4.1	Design Class Hierarchy.....	pg # 31
7.3.4.2	Restrictions/Limitations.....	pg # 31
7.3.4.3	Performance Issues.....	pg # 31
7.3.4.4	Design Constraints.....	pg # 32
7.3.4.5	Processing Detail For Each Operation.....	pg # 32
7.4	login.html.....	pg # 32
7.4.1	Classification.....	pg # 32
7.4.2	Processing Narrative(PSPEC).....	pg # 32
7.4.3	Interface Description.....	pg # 32
7.4.4	Processing Detail.....	pg # 32
7.4.4.1	Design Class Hierarchy.....	pg # 32
7.4.4.2	Restrictions/Limitations.....	pg # 33
7.4.4.3	Performance Issues.....	pg # 33
7.4.4.4	Design Constraints.....	pg # 33
7.4.4.5	Processing Detail For Each Operation.....	pg # 33
7.5	newPassForm.html.....	pg # 33

7.5.1	Classification.....	pg # 33
7.5.2	Processing Narrative(PSPEC).....	pg # 33
7.5.3	Interface Description.....	pg # 33
7.5.4	Processing Detail.....	pg # 33
7.5.4.1	Design Class Hierarchy.....	pg # 33
7.5.4.2	Restrictions/Limitations.....	pg # 34
7.5.4.3	Performance Issues.....	pg # 34
7.5.4.4	Design Constraints.....	pg # 34
7.5.4.5	Processing Detail For Each Operation.....	pg # 34
7.6	passRecovery.html.....	pg # 34
7.6.1	Classification.....	pg # 34
7.6.2	Processing Narrative(PSPEC).....	pg # 34
7.6.3	Interface Description.....	pg # 34
7.6.4	Processing Detail.....	pg # 34
7.6.4.1	Design Class Hierarchy.....	pg # 34
7.6.4.2	Restrictions/Limitations.....	pg # 34
7.6.4.3	Performance Issues.....	pg # 34
7.6.4.4	Design Constraints.....	pg # 34
7.6.4.5	Processing Detail For Each Operation.....	pg # 35
7.7	signup.html.....	pg # 35
7.7.1	Classification.....	pg # 35
7.7.2	Processing Narrative(PSPEC).....	pg # 35
7.7.3	Interface Description.....	pg # 35
7.7.4	Processing Detail.....	pg # 35
7.7.4.1	Design Class Hierarchy.....	pg # 35
7.7.4.2	Restrictions/Limitations.....	pg # 35
7.7.4.3	Performance Issues.....	pg # 35
7.7.4.4	Design Constraints.....	pg # 35
7.7.4.5	Processing Detail For Each Operation.....	pg # 36
7.8	init.py.....	pg # 36
7.8.1	Classification.....	pg # 36
7.8.2	Processing Narrative(PSPEC).....	pg # 36
7.8.3	Interface Description.....	pg # 36
7.8.4	Processing Detail.....	pg # 36
7.8.4.1	Design Class Hierarchy.....	pg # 36
7.8.4.2	Restrictions/Limitations.....	pg # 36
7.8.4.3	Performance Issues.....	pg # 36
7.8.4.4	Design Constraints.....	pg # 36
7.8.4.5	Processing Detail For Each Operation.....	pg # 37
7.9	auth.py.....	pg # 37
7.9.1	Classification.....	pg # 37
7.9.2	Processing Narrative(PSPEC).....	pg # 37
7.9.3	Interface Description.....	pg # 37
7.9.4	Processing Detail.....	pg # 38
7.9.4.1	Design Class Hierarchy.....	pg # 38
7.9.4.2	Restrictions/Limitations.....	pg # 38

	7.9.4.3	Performance Issues.....	pg # 38
	7.9.4.4	Design Constraints.....	pg # 38
	7.9.4.5	Processing Detail For Each Operation.....	pg # 38
7.10		models.py.....	pg # 38
	7.10.1	Classification.....	pg # 38
	7.10.2	Processing Narrative(PSPEC).....	pg # 38
	7.10.3	Interface Description.....	pg # 38
	7.10.4	Processing Detail.....	pg # 38
	7.10.4.1	Design Class Hierarchy.....	pg # 39
	7.10.4.2	Restrictions/Limitations.....	pg # 39
	7.10.4.3	Performance Issues.....	pg # 39
	7.10.4.4	Design Constraints.....	pg # 39
	7.10.4.5	Processing Detail For Each Operation.....	pg # 39
7.11		grapher.py.....	pg # 39
	7.11.1	Classification.....	pg # 39
	7.11.2	Processing Narrative(PSPEC).....	pg # 39
	7.11.3	Interface Description.....	pg # 39
	7.11.4	Processing Detail.....	pg # 39
	7.11.4.1	Design Class Hierarchy.....	pg # 39
	7.11.4.2	Restrictions/Limitations.....	pg # 40
	7.11.4.3	Performance Issues.....	pg # 40
	7.11.4.4	Design Constraints.....	pg # 40
	7.11.4.5	Processing Detail For Each Operation.....	pg # 40
7.12		views.py.....	pg # 40
	7.12.1	Classification.....	pg # 40
	7.12.2	Processing Narrative(PSPEC).....	pg # 40
	7.12.3	Interface Description.....	pg # 40
	7.12.4	Processing Detail.....	pg # 40
	7.12.4.1	Design Class Hierarchy.....	pg # 40
	7.12.4.2	Restrictions/Limitations.....	pg # 40
	7.12.4.3	Performance Issues.....	pg # 41
	7.12.4.4	Design Constraints.....	pg # 41
	7.12.4.5	Processing Detail For Each Operation.....	pg # 41
7.13		main.py.....	pg # 41
	7.13.1	Classification.....	pg # 41
	7.13.2	Processing Narrative(PSPEC).....	pg # 41
	7.13.3	Interface Description.....	pg # 41
	7.13.4	Processing Detail.....	pg # 41
	7.13.4.1	Design Class Hierarchy.....	pg # 41
	7.13.4.2	Restrictions/Limitations.....	pg # 41
	7.13.4.3	Performance Issues.....	pg # 41
	7.13.4.4	Design Constraints.....	pg # 41
	7.13.4.5	Processing Detail For Each Operation.....	pg # 41
7.14		.gitignore.....	pg # 42
	7.14.1	Classification.....	pg # 42
	7.14.2	Processing Narrative(PSPEC).....	pg # 42

7.14.3	Interface Description.....	pg # 42
7.14.4	Processing Detail.....	pg # 42
7.14.4.1	Design Class Hierarchy.....	pg # 42
7.14.4.2	Restrictions/Limitations.....	pg # 42
7.14.4.3	Performance Issues.....	pg # 42
7.14.4.4	Design Constraints.....	pg # 42
7.14.4.5	Processing Detail For Each Operation.....	pg # 42
7.15	Dockerfile.....	pg # 43
7.15.1	Classification.....	pg # 43
7.15.2	Processing Narrative(PSPEC).....	pg # 43
7.15.3	Interface Description.....	pg # 43
7.15.4	Processing Detail.....	pg # 43
7.15.4.1	Design Class Hierarchy.....	pg # 43
7.15.4.2	Restrictions/Limitations.....	pg # 43
7.15.4.3	Performance Issues.....	pg # 43
7.15.4.4	Design Constraints.....	pg # 43
7.15.4.5	Processing Detail For Each Operation.....	pg # 44
7.16	requirement.txt.....	pg # 44
7.16.1	Classification.....	pg # 44
7.16.2	Processing Narrative(PSPEC).....	pg # 44
7.16.3	Interface Description.....	pg # 44
7.16.4	Processing Detail.....	pg # 44
7.16.4.1	Design Class Hierarchy.....	pg # 44
7.16.4.2	Restrictions/Limitations.....	pg # 44
7.16.4.3	Performance Issues.....	pg # 44
7.16.4.4	Design Constraints.....	pg # 44
7.16.4.5	Processing Detail For Each Operation.....	pg # 44
User Interface		
8.1.	Overview of User Interface.....	pg # 45
8.2.	Screen Frameworks or Images.....	pg # 45
8.3.	User Interface Flow Model.....	pg # 46
Database Design		
Requirements Validation and Verification.....		pg # 46
Glossary.....		pg # 48
References.....		pg # 50

Revision History

Name	Date	Reason For Changes	Version
Jacob Hertz	4/29/2024	Sections 6 & 7	v0.9
Adam Dixon	5/1/2024	Lower Level Components	v0.9.1
Geovanny Montano	5/2/2024	Section 7.12 views.py	v0.9.1
Fernando Serrano	5/3/2024	Section 4: System Architecture - updated the section to be more in line with what tools we are using for the web app	v0.9.2
Steven Partida	5/3/2024	Section 8: Database design	v0.9.3
Adam Dixon, Ryan Torrez	5/3/2024	Table of contents	v1.0
Adam Dixon	5/7/2024	Edits to Sections: 6	v1.1
Jacob Hertz, Nicholas Trigueros	5/7/2024	Edits to Sections 4.1 and 4.4	v1.2

<Add rows as necessary when the document is revised. This document should be consistently updated and maintained throughout your project. If ANY requirements are changed, added, removed, etc., immediately revise your document.>

1. Introduction (Washika Afrozi)

1.1 Purpose

- **To Help Focus on Design Goals:** The software design document helps to focus on the goals of the project and to provide a clear vision of what needs to be done. The document should also define the scope of the project.
- **Managing Change:** Managing change is one of the most challenging aspects of software development. It is important to understand why and when changes are required, how they should be implemented, and how they should be tested before the release.
- **To Improves Feedback and Implementation:** A software design document improves communication between different members of the team while they are working on the project. The software design document also helps in improving feedback and implementation of the project as well as reducing cost, time, and effort.
- **Prevents Miscommunication:** A design document also helps prevent miscommunication between developers and non-developers. For example, when we write a web application, we may want to create a separate page for each feature to avoid having too many pages on one page. However, if there's no clear definition of these features, it can lead to confusion among non-technical members of our team who are not familiar with programming languages or frameworks .
- **Tracking Progress:** The design document is important because it allows your team to understand what needs to be built, how you are going to build it, and what the end users will experience when they use the product. It also gives you a starting point for tracking progress and communicating with your stakeholders about where things are at in terms of schedule, quality, etc

1.2 Document Conventions

A software design document should be purely subjective and a matter of personal preference. Also a design document should only be useful if it's actively read and updated. SDD will be open, collaborative process, and organized, also feedback loops and reviews are crucial for a successful outcome. These design docs should be easily accessible to all stakeholders by keeping them in their shared drive and should be possible to understand and implement the design document.

1.3 Intended Audience and Reading Suggestions

Software design document is written for a more general audience, this document is intended for individuals directly involved in the development of software developers, project consultants, and team managers. The software design document should not be long as written of the system architecture or design. They should write user requirements in natural language, with simple tables, forms, and intuitive diagrams. Use clear language. That is important. Ideally , a

technical writer should work on an SDD. Writers should pay attention to how to write: no unclear words for the audience, clear sentences, the right amount of terms so that everyone can understand the documents.

1.4 System Overview

Software design and implementation are inter-leaved activities. The level of detail in the design depends on the type of system and whether you are using a plan-driven or agile approach.

- The process of object-oriented design includes activities to design the system architecture, identify objects in the system, describe the design using different object models and document the component interfaces. For example, the Linux operating system is widely used as a server system and, increasingly, as a desktop environment.
- Other important open source products are Java, the Apache web server and the MySQL database management system.

2. Design Considerations(Geo)

2.1 Assumptions and Dependencies

1. Related Software or Hardware
 - Assumption: The web application relies on modern web technologies such as HTML5, JavaScript, and CSS for front-end development.
 - Dependency: The application depends on specific libraries or frameworks for graph generation, such as Plotly. Compatibility with these libraries and their dependencies must be ensured.
2. Operating Systems:
 - Assumption: The web application is designed to be platform-independent and accessible from various operating systems, including Windows, macOS, and Linux.
 - Dependency: Compatibility testing across different web browsers (e.g Chrome, Firefox, Safari, Edge) and their versions are necessary to ensure consistent performance and rendering of graphs.
3. End-user Characteristics:
 - Assumption: Users accessing the web application have basic internet connectivity and familiarity with web browsing.
 - Dependency: User interfaces should be intuitive and user-friendly to accommodate users with varying levels of technical expertise. Additionally, the application's responsiveness should cater to different screen sizes and device types.
4. Possible and/or Probable Changes in Functionality:
 - Assumption: The requirements for graph generation may evolve over time based on user feedback or changing business needs.
 - Dependency: The application's design should allow for scalability and flexibility to incorporate new graph types, features, or data visualization techniques in response to changing requirements without requiring significant redevelopment.

2.2 General Constraints

1. Hardware or Software Environment:
 - Constraint: The application needs to be compatible with a wide range of devices, like desktops, laptops, phones and tablets, and browsers like stated in section 2.1
 - Impact: Designing responsive and cross-platform user interfaces to ensure consistent user experience across various devices and screen sizes is essential.
2. End-user Environment:
 - Constraint: Users may have different levels of technical expertise and may access the application from different network environments.

- Impact: User interfaces should be intuitive and easy to navigate, and the application should perform well even in low-bandwidth or high-latency network conditions.
3. Availability or Volatility of Resources:
 - Constraint: Resources such as server capacity, storage, and bandwidth may be limited or subject to fluctuations.
 - Impact: Implementing efficient resource management strategies to optimize performance, scalability, and reliability. This includes caching mechanisms, load balancing, and scalability planning for handling peak loads.
 4. Interoperability Requirements:
 - Constraint: The application may need to integrate with other systems or platforms, such as data sources or APIs.(Kaggle API)
 - Impact: Designing flexible and standardized interfaces to facilitate seamless integration with external systems while maintaining data integrity and security.
 5. Verification and Validation Requirements (Testing):
 - Constraint: Thorough testing is necessary to ensure the application functions correctly, performs well, and meets user requirements.
 - Impact: Implementing automated testing processes, including unit tests, integration tests, and end-to-end tests, to validate functionality, performance, and reliability throughout the development lifecycle.

2.3 Goals and Guidelines

1. KISS Principle:

- Goal: Prioritize simplicity in design and functionality to ensure ease of use and maintenance.
- Reason for desirability: A simple and intuitive user interface reduces the learning curve for users, leading to higher adoption rates and increased user satisfaction. Additionally, simpler designs are easier to maintain and iterate upon, reducing development time and costs.

2. Meeting Mandatory Delivery Date:

- Goal: Ensure the web application is delivered by the specified deadline, such as the end May 10,2024.
- Reason for desirability: Meeting delivery deadlines is essential for project success and customer satisfaction. It allows for proper planning, coordination, and deployment, ensuring that users receive the product within the expected timeframe.

3. Emphasis on Speed versus Memory Use:

- Goal: Strike a balance between speed and memory usage to optimize performance without sacrificing user experience or system stability.
- Reason for desirability: Users expect web applications to load quickly and respond promptly to user interactions. Optimizing speed enhances user satisfaction and engagement. However, it's also essential to consider memory usage to ensure the

application remains lightweight and responsive across various devices and network conditions.

4. Consistency with Existing Products:

- Goal: Design the web application to work, look, or "feel" like existing products or industry standards.
- Reason for desirability: Familiarity breeds comfort and confidence in users. By emulating existing products or adhering to established design patterns and conventions, users can navigate the application more easily, leading to a better user experience and reducing the learning curve.

2.4 Development Methods

Agile Development Methodology:

- Description: Agile is an iterative and incremental approach to software development that prioritizes flexibility, collaboration, and adaptability. It involves breaking down the project into smaller, manageable tasks or user stories, which are completed in short iterations called sprints. Throughout each sprint, there is continuous collaboration between the development team and stakeholders, allowing for adjustments and refinements based on evolving requirements and feedback.
- Application: Agile methodologies were chosen for their ability to accommodate changing requirements, promote transparency, and deliver working software iteratively. By breaking down the project into smaller, achievable goals, we can maintain focus, adapt to changing priorities, and deliver value incrementally. Daily stand-up meetings, sprint planning, retrospectives, and regular demos are key practices within Agile that facilitate communication, collaboration, and continuous improvement.
- Flexibility: Agile allows for flexibility in accommodating changing requirements and priorities, ensuring that the software remains aligned with stakeholder needs.
- Iterative Delivery: By delivering working software in short iterations, Agile enables early validation of assumptions and feedback-driven refinement, reducing the risk of costly rework.
- Transparency: Agile promotes transparency through regular communication, visibility into progress, and collaboration between team members and stakeholders.
- Customer Satisfaction: Agile prioritizes customer satisfaction by focusing on delivering value early and continuously incorporating feedback to improve the product.

3. Architectural Strategies (Nick)

Architectural Pattern: Model View Controller (MVC)

- **Model:** Represents the data and business logic of the application
- **View:** Handles the presentation (visuals) layer of the application.
- **Controller:** Acts as an intermediary between the Model and View components.

The MVC architectural pattern was selected because it promotes modularity, separation of concerns, and maintainability. By decoupling the user interface from the underlying data and business logic, it allows for easier maintenance and scalability of the system. Additionally, it enables parallel development by different teams or individuals working on different components of the application.

Programming Language and Libraries:

The system is implemented using Python as the primary programming language, with the incorporation of Flask (a python based web application framework), Javascript, Html, and CSS. Other programming languages such as R or Java were considered for their strong capabilities in statistical analysis and web development, respectively. However, Python was preferred for its ease of use, extensive libraries for data analysis, and its growing popularity in the data science community.

Javascript, HTML, and CSS are used for our front end website layout and design. JavaScript being used in conjunction with bootstrap makes for our more dynamic functionalities such as drop downs and UI. While CSS uses are more in line with the styling of the web application UI.

Python was chosen for its simplicity, readability, and extensive ecosystem of libraries for data analysis and visualization. Pandas provides powerful data structures and functions for data manipulation, while Plotly offers flexible and customizable plotting capabilities. Flask was selected for its lightweight and minimalist framework, suitable for building web applications with minimal overhead.

Future Plans for Extension:

The system is designed with extensibility in mind, allowing for easy integration of new features and functionalities in the future. This includes the ability to incorporate additional data sources, support for different types of visualizations, and scalability to handle larger datasets. By designing the system with extensibility in mind, it ensures that the software can adapt to evolving user needs and technological advancements. This promotes longevity and reduces the need for complete overhauls or rewrites as requirements change over time.

Error Detection and Recovery:

The system implements robust error detection and recovery mechanisms to handle unexpected failures or errors gracefully. This includes proper exception handling, logging, and user-friendly error messages to assist users in troubleshooting issues.

Error detection and recovery are crucial for ensuring the reliability and usability of the system. By providing informative error messages and logging, users can quickly identify and resolve issues, minimizing downtime and frustration.

These design decisions and strategies are aimed at achieving the stated goals of streamlining the data visualization process, simplifying data analysis, and providing a user-friendly platform for users. Each decision was carefully evaluated based on its alignment with these goals, balancing factors such as simplicity, flexibility, scalability, and usability.

4. System Architecture (Fernando)

4.1 Logical View

The logic of this application contains many files with essential classes and packages:

main.py:

The driver code that imports the folder holding the other files that make up the application, and runs it.

__init__.py:

The class that creates the application and all of its essential variables and functions, like database creation, downloading csv files, etc, that will be used by the other files. It also imports classes and functions from those other files, like auth.py, models.py, and views.py to make sure its own functions are complete

auth.py:

The class that creates functions for many of the interactive elements within the html that is then displayed to the user.

models.py:

This file contains 3 classes that instantiate the parameters that will be passed to the other files that need it: Datasets, Favorites, and User.

views.py:

The class that is responsible for the functionality of the login page of the application. It imports the User from models.py to do so

grapher.py:

This class is responsible for creating the graphs that are the main purpose of this application.

HTML:

These files contain the code that displays the frontmost information to the user on the application. Any interactive capabilities are imported from the python code.

SQL and csv:

These SQL files contain code that stores data distinguished by the python code into a database. The csv files are data files pulled from the Kaggle library that are stored within the local database, and are used to give the graphs info to use.

Flask library:

This Python library is essential to this application in order to allow functionality between the python code and the html. All classes use some form of the library in order to run. This even includes sub libraries, like SQLAlchemy, which allow functionality between the python code and the sql database.

Kaggle library:

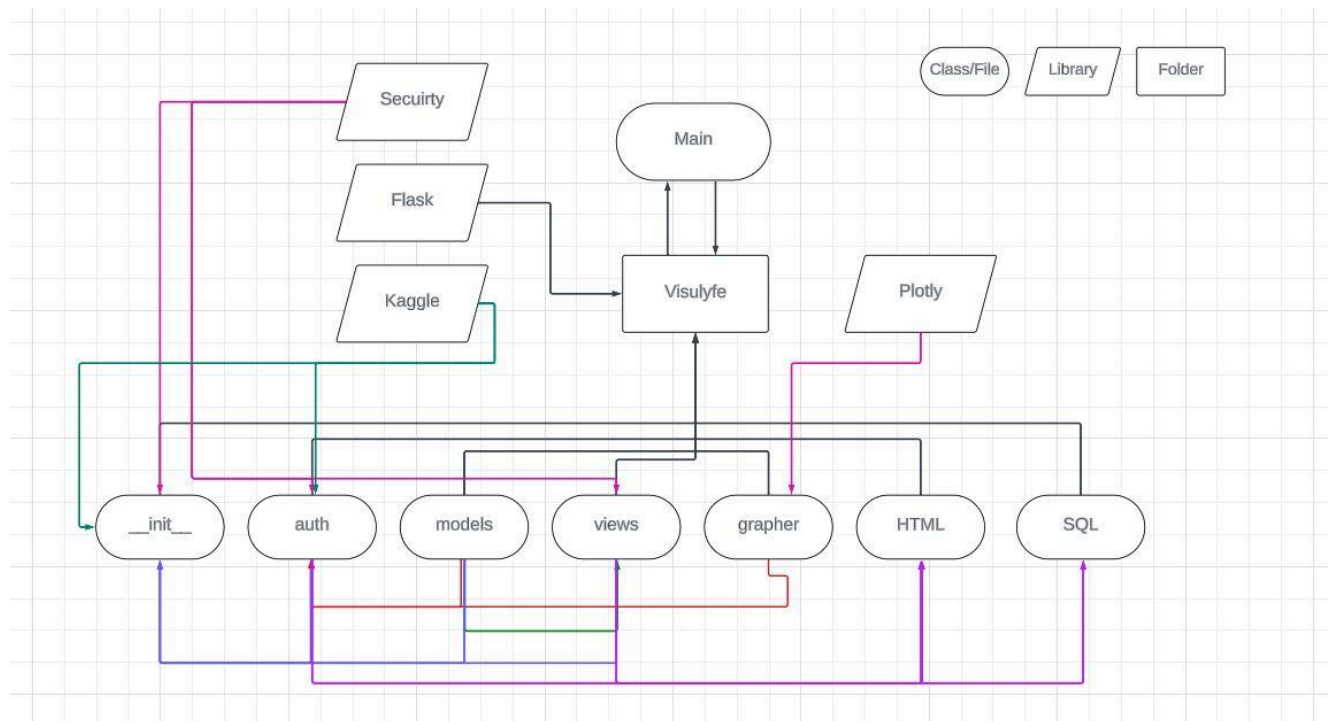
This database library is essential to this application in order for the graphs to contain any kind of data. `__init__.py` and `auth.py` import this in order to gain access to its vast array of csv files to use for graphs.

Plotly and Pandas:

This python library is essential to this application in order to allow the code to be able to create graphs that contain data from the Kaggle library. It is mainly used by `grapher.py`.

Security libraries:

These libraries mainly consist of `werkzeug` and `itsdangerous`, which are python libraries related to security measures. `Werkzeug` focuses more on hashing and encrypting any private information that is stored in the database, and `itsdangerous` focuses more on testing the security measures themselves.



4.2 Development View

For the development of this application we decided to partition it into three modules:

Frontend Modules:

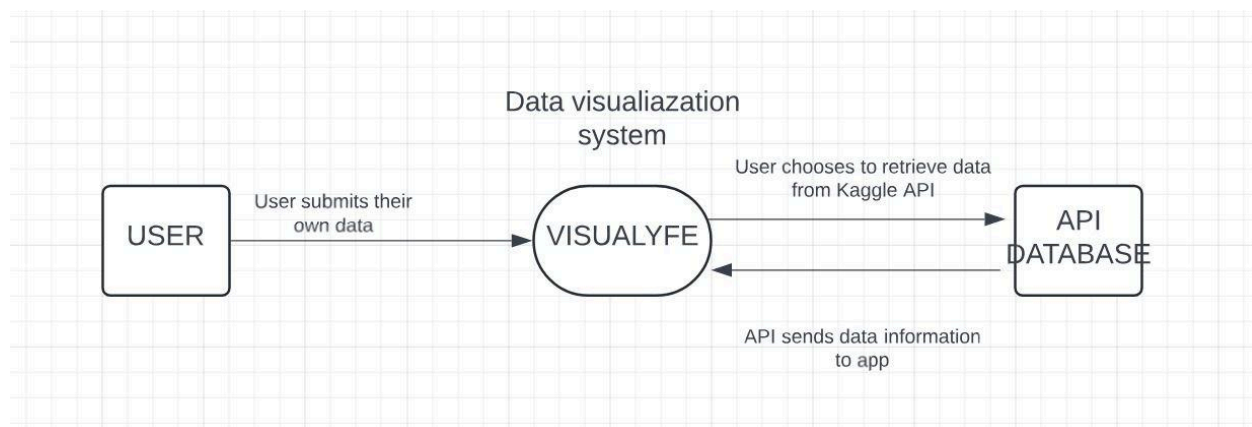
The responsibilities of the front end team would be to manage the user interface as well as the client-side interactions. Using HTML, CSS, and Flask, a Python framework, the front end team would work together to implement clients-side logic for a great user experience as well as the data visualization.

Back-End Module:

The responsibilities of the back-end modules would be to handle the server side processing. Account creation will also be handled by this team. Using Python, they will handle the data visualization process. Also communication between the database and Kaggle API.

Database Module:

The responsibilities of the database module would be to create a database that will aid in account creation (authentication) as well as store User info if the user decides to create an account.



FLASK (Python) :

Flask was chosen for our project because of how flexible it is. Also, Flask is an easy to use framework to create a web application. This along with Python's versatility makes creating our web app much simpler.

SQL :

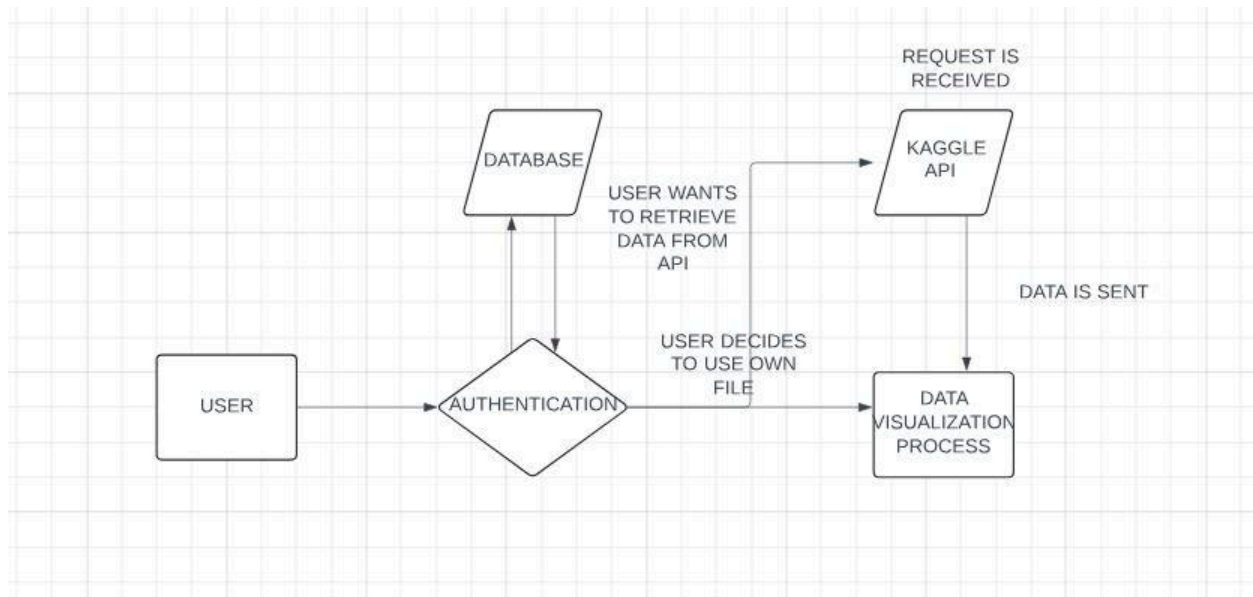
For the database we will be using SQLAlchemy to ensure efficient data management. We will be able to create structured tables to store user information and data that will be used for data visualization. SQLAlchemy is a Python SQL toolkit that allows us to access and manage an SQL database using python language.

Visualization:

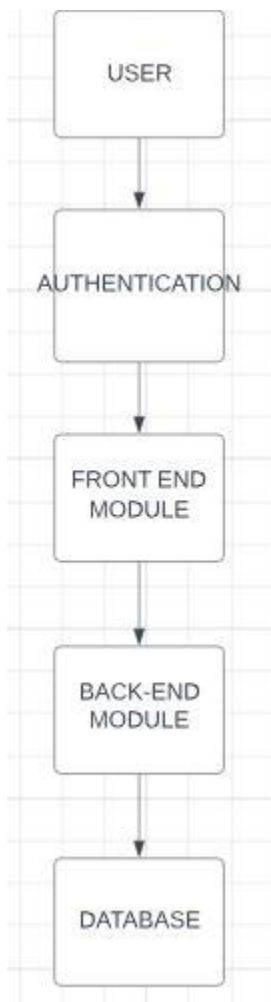
The backend team will be responsible for generating visualizations based on data that will be inputted from a csv file by the user. They will be using Plotly for the creation of the graphs. Also, the user can choose to grab an already existing dataset from the Kaggle API and see a visual of it.

Authentication:

Users can choose to create their own accounts to save different datasets if they choose to.



4.3 Process View

**User Interaction:**

Users will interact with the web app through a user interface. In order to use the web application, a user must create an account with a name, email, and password. Once signed in, the user will be able to upload their own dataset. They will also be able to search datasets using a search bar that is connected to the Kaggle API.

Front End Module:

The front end will receive and manage user inputs as well as the presentation of the website to ensure a smooth and satisfying user experience. They will also be responsible for sending input from the user to the backend modules.

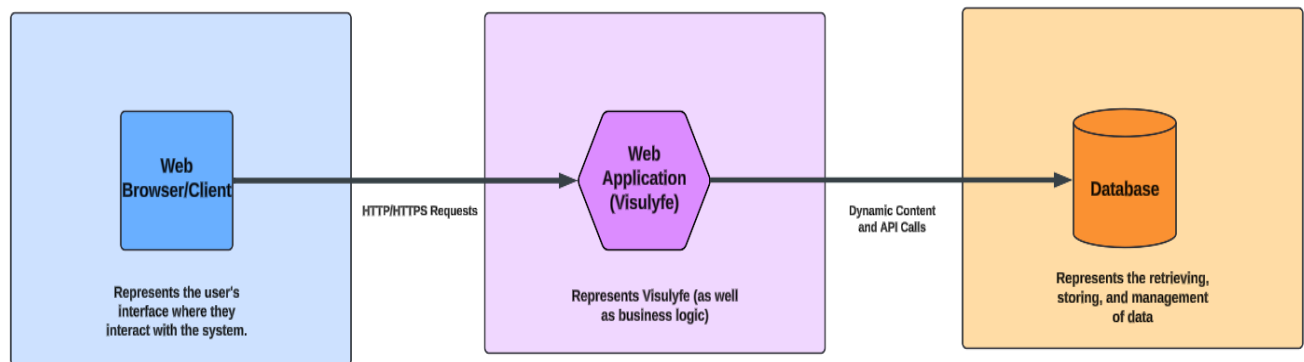
Back End Module:

The backend module will receive requests from the frontend and process them accordingly. The backend module will handle authentication of the user as well as dataset retrieval from the API and will be responsible for generating data visualization of data.

Database Module:

The database will store user account information such as names and passwords. The database will also be able to store user datasets. It will allow for the retrieval of data that the backend module will need.

Each module will be responsible for a specific functionality of the system which is why we broke it up this way. It will be easier to maintain and test since each part can be worked on independently. Also, with it being broken up this way scaling is possible.

4.4 Physical view

5. Policies and Tactics (Jacob)

5.1 Choice of which specific products used

- Code with html, python, and sql
 - HTML for front-end
 - Python for back-end
 - using Flask to be compatible with html, and Plotly libraries for generating the graphs
 - SQL for the database storing user data
 - using Amazon Web Service (AWS) for hosting
 - using Kaggle for the API
- Everyone uses Visual Studio Code to write their programs
 - The IDE is capable of producing different coding language files
 - The IDE also comes with its own compiler
- Use the GitHub repository to work collaboratively with the team

5.2 Plans for ensuring requirements traceability

- The essential Python classes needed to run the app are split up into separate files
- Each of these files have their own import library based off of Flask and Plotly
- Each file's distinct set of imports help make it clear which class uses certain design components, code, etc
- Subclasses and functions are clearly defined in each class file
- If any file calls upon another, that is also made explicitly clear

5.3 Plans for testing the software

- The simplest way to test the software is to run it and see what happens.
- Changing variables to test different outputs is something that anyone can do, and can help possibly see where any holes/bugs in the code are.
- If possible, we could also get some outside help from a third party to also help test the code.

5.3.1 Engineering trade-offs

- As simple as this method is, it could easily become time consuming if we don't manage our time correctly
- The issue could also be that there are just too many possibilities to test out on our own

5.3.2 Coding guidelines and conventions

- Code will be worked on collaboratively through GitHub repository
- HTML for front-end
 - CSS (style sheet) will be on the same file as the main code
- Python for back-end
 - no camel-case, use underscores

- SQL for the database
 - work through certain python classes to work on sql database
- Leave comments for things that require help or uncertainty

5.3.3 The protocol of one or more subsystems, modules, or subroutines

- Each subsystem or the like are categorized based on their design implementation and file category
 - For instance, auth.py is for back-end programmers to connect the html to the database

5.3.4 The choice of a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality

- The team decided on using Flask since it is a widely used and helpful module for coding a website
- Flask comes with an array of submodules that come with their own uses as well
 - for instance, sqlalchemy has a hash function that can be used to protect user information that is stored in the database
- Plotly is an easy to use graphing library for Python, which helps us make our main purpose for the project

5.3.5 Plans for maintaining the software

- It is doubtful that everything on the backlog will be implemented by the deadline
- The most essential tasks will be prioritized
- However, devs will be continuously working on the code to improve it, and possibly add more functionality that was not there prior

5.3.6 Interfaces for end-users, software, hardware, and communications

- This will start out as a website
 - Once the website is finished, a mobile app will be made later if time allows
- Users will simply need a computer of their choice and a connection to the internet

5.3.7 Hierarchical organization of the source code into its physical components (files and directories).

- Code split the most essential classes into separate files
- These files are kept in a folder alongside the main file
- Each class file is distinct in its purpose and function

5.3.8 How to build and/or generate the system's deliverables (how to compile, link, load, etc.)

- Code is done thru Visual Studio Code
- VSCode has its own compiler
- Running code produces a local website
- AWS gives a local host to use to display the website

5.3.9 Describe tactics

*refer to 5.3.4 and others above

6. Detailed System Design (Jacob)

6.1 Front-End (Module)

6.1.1 Responsibilities

The front-end serves as the interface between the user and the system. Its main responsibilities include:

- Displaying Information
 - Presenting a clear, intuitive layout that guides the user through the website's features.
 - Displaying data, forms, and visualizations in an organized manner.
- User Interaction
 - Providing elements like buttons, forms, and dropdowns to enable user interaction.
 - Handling user events, such as clicks, form submissions, and data input.
- Visual Appeal
 - Ensuring the design is aesthetically pleasing, with a clean and modern look.
 - Maintaining consistency in style and branding throughout the website.
- Navigation and Usability
 - Offering intuitive navigation to help users find what they're looking for.
 - Ensuring accessibility and responsiveness across different devices and screen sizes.

6.1.2 Constraints

Constraints on the front-end are primarily tied to its role and interactions with other components:

- Limited Computation
 - The front-end should avoid heavy computations, delegating such tasks to the back-end or database. Complex processing on the front-end can lead to performance issues.

- Dependency on Other Components
 - The front-end relies on the back-end to provide data and handle business logic. If the back-end or database fails, it becomes visible through the front-end.
- Consistent User Experience
 - The front-end must offer a consistent experience, even if there are issues with the back-end or database. Error handling and feedback to users are crucial to maintain trust.
- Accessibility and Responsiveness
 - The design must be accessible to all users, including those with disabilities. Responsiveness ensures a smooth experience on different devices and screen sizes.

6.1.3 Composition

The composition of the front-end consists of:

- about.html
 - The "About" page provides information about the website's purpose, mission, and background.
- home.html
 - The home page serves as the main landing page. It includes key features.
- login.html
 - The login page allows existing users to authenticate by entering their credentials. It contains a form with fields for email and password.
- signup.html
 - The sign-up page is designed for new users to create an account. It includes a form for user details such as name, email, and password, along with validation logic.
- newPassForm.html
 - The password reset page allows users to request a password reset. It includes a form where users can enter a new password after they've been verified through a recovery process.
- passrecovery.html
 - This page facilitates the password recovery process. It involves sending an email to the user with a password reset link for verification.
- base.html
 - The base template serves as the primary layout for all other pages. It includes elements like navigation bars and general styling. Other pages extend from this template to ensure consistency across the site.
 - It also includes a carousel displaying various Kaggle Datasets
- Carousel

- A carousel component that displays a rotating series of images, text, or other content. It is typically used on the home page to showcase key features or updates.
- Graphing Feature
 - A custom component utilizing Plotly or similar libraries to visualize data. It can be used to display interactive charts and graphs for data analysis or insights.
- Kaggle API Search Function
 - A component or script that interacts with the Kaggle API to fetch data. This function may include a search form allowing users to query Kaggle datasets or retrieve specific information for visualization.

6.1.4 Uses/Interactions

The front-end interacts closely with the back-end and the database. As stated in 6.1.1 and 6.1.2, this component displays information, and any function, like with the graphs, is done with the back-end. The front-end interacts with the database by storing user data when a new user signs up for the site, and recalls that data whenever returning users log in to the site. It also calls the Kaggle API whenever a user selects a data set to be displayed as a graph.

6.1.5 Resources

Resources used/needed were as follows:

- Visual Studio Code(VSCode)
 - A lightweight yet powerful code editor with a rich ecosystem of extensions. Ideal for front-end development with support for multiple languages, integrated terminal, and Git integration.
- Node.js and npm/yarn
 - Essential for running JavaScript on the server-side (Node.js) and managing project dependencies (npm or yarn).
- HTML
 - The foundational language for structuring web content. Used for building the layout of web pages.
- CSS
 - Stylesheet language used to control the appearance of HTML elements. Essential for creating visually appealing designs and responsive layouts.
- JavaScript
 - The scripting language for adding interactivity and dynamic behavior to web pages. Used extensively for client-side logic and DOM manipulation.
- Plotly
 - A JavaScript library for creating interactive data visualizations.
- GitHub

- collaborate with others with this software
- Modern Desktop or Laptop
 - A reliable computer capable of running development software, with sufficient processing power and memory for smooth development.
- Web Browser
 - A modern browser (e.g., Chrome, Firefox, Edge) for testing and debugging front-end code.

6.1.6 Interface/Exports

- Data Types:
 - Forms
 - Flask-WTForms for user input, such as login, registration, or data query parameters.
 - Form fields for text, passwords, numbers, dropdowns, etc.
 - Scripts
 - JavaScript scripts for interactive features, client-side form validation, dynamic content updates, or user interactions.
 - Blocks
 - Flask Jinja2 templates for rendering HTML components.
 - Blocks for displaying formatted data, charts, and tables.
- Interactions:
 - Backend Database Operations
 - SQLAlchemy for ORM (Object-Relational Mapping) to interact with the SQL database.
 - Database models defining tables for user data, Kaggle data, and related relationships.
 - CRUD operations for storing and retrieving data from the database.
 - Frontend Data Display
 - Flask routes serving HTML templates to display the frontend.
 - Data fetched from the database to display in forms or as part of visualization.
 - Plotly used to generate interactive visualizations, such as line charts, bar charts, or scatter plots.
 - User Authentication
 - Flask-Login for user authentication and session management.
 - Secure password storage and hashing (e.g., using `werkzeug.security.generate_password_hash`).
 - Login and registration routes for user authentication.
 - Data Retrieval from Kaggle API
 - Requests or HTTP-based library for fetching data from Kaggle API.

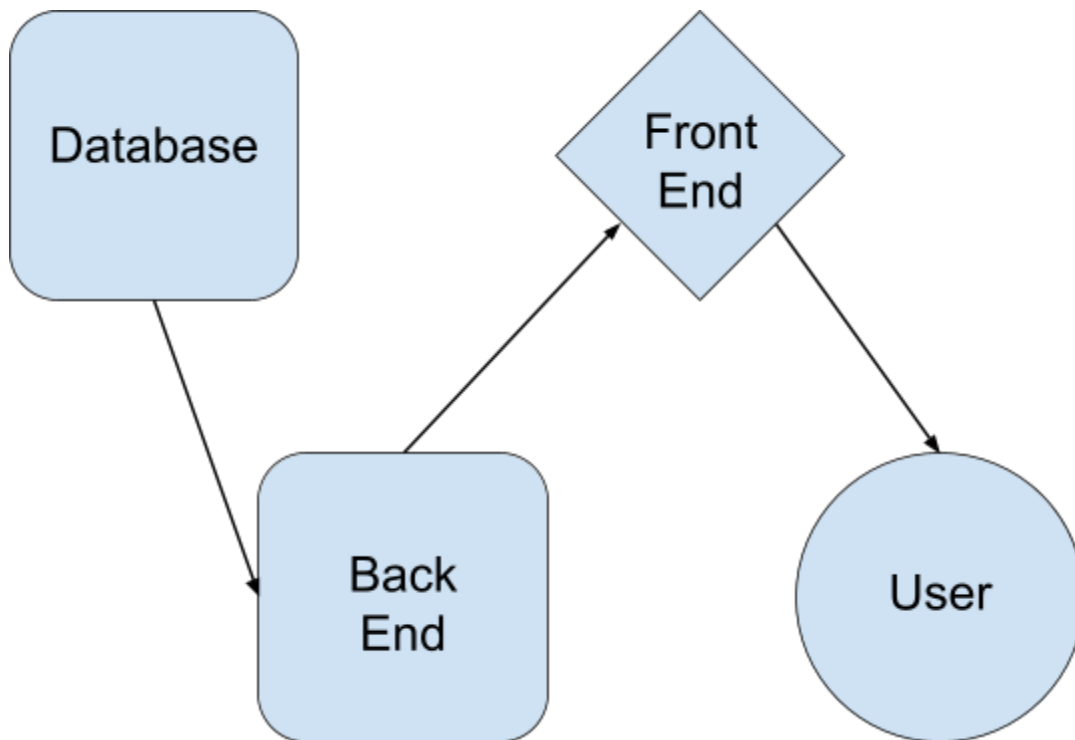
- Authentication via Kaggle API key for secure access.
 - Transformation of Kaggle data into a suitable format for storage in SQL database.
- Data Manipulation and Storage
 - Data cleaning and processing in Python before storing it in the SQL database.
 - Normalization or denormalization depending on the specific requirements of the visualization.
- Client-Server Communication
 - HTTP-based interactions between Flask backend and frontend clients.
- Security Considerations
 - CSRF protection for form submissions.
 - Secure sessions to maintain user authentication.
 - Data validation and sanitization to prevent SQL injection and other security vulnerabilities.

6.2 Back-End (Module)

6.2.1 Responsibilities

The primary responsibilities of the back-end module include:

- Data Handling and Processing
 - Interacting with the database to store and retrieve user data and datasets.
 - Managing data transformation and processing for front-end visualization.
- User Authentication and Authorization
 - Validating user credentials for login and ensuring secure session management.
 - Handling sign-up, password recovery, and password reset functionality.
- Graph Generation
 - Generating visualizations based on user-selected datasets.
 - Providing data to the front-end for rendering graphs and charts.
- Interfacing with External APIs
 - Integrating with the Kaggle API to fetch datasets based on user input.
 - Handling API authentication and secure data transfer.



Data Flow Diagram 6.2

6.2.2 Constraints

Given its role as the link between the front-end and the database, the back-end must function seamlessly to ensure the stability and reliability of the system. Constraints include:

- **Stability and Reliability**
 - If the back-end encounters errors or downtime, it affects the entire system's functionality.
- **Security**
 - The back-end must enforce strict security measures to protect user data and ensure safe authentication.
- **Scalability**
 - The architecture must be designed to scale with increasing user demand and data complexity.

6.2.3 Composition

The composition of the back-end consists of:

- `__init__.py`
 - Initializes the Flask application and sets up routing and configuration.
- `auth.py`
 - Contains the logic for user authentication, sign-up, and password management.
 - Includes session management and secure storage of user credentials.
- `models.py`
 - Defines the database models using SQLAlchemy ORM.
 - Manages the structure for user information, datasets, and other related entities.
- `grapher.py`
 - Implements the logic for creating graphs using Plotly.
 - Functions to process data and return it in a format suitable for visualization.
- `views.py`
 - Handles the interaction between the back-end and front-end, including data retrieval and user authentication.
 - Contains routes for front-end requests and data fetching.
- `main.py`
 - The entry point to run the Flask application.
 - Initializes all necessary components and starts the web server.

6.2.4 Uses/Interactions

The back-end interacts with both the front-end and the database, providing a communication layer and data processing capabilities:

- Front-End Interaction
 - The back-end provides data to the front-end for visualization and manages user authentication.
 - Processes user requests and sends appropriate responses, including HTML templates and JSON data.
- Database Interaction
 - Stores and retrieves user information, authentication data, and datasets.
 - Manages data integrity and enforces constraints for security and consistency.

6.2.5 Resources

To develop the back-end, various resources are required:

- Development Environment
 - Visual Studio Code for code editing and debugging.
 - Python, Flask, and related libraries for building the back-end.
- External Libraries
 - Plotly for data visualization.
 - Pandas for data manipulation and analysis.
 - Flask-WTF for form handling and validation.
- Version Control and Collaboration
 - GitHub for source control and team collaboration.
- Security Tools
 - Werkzeug for secure hashing and encryption.
 - ItsDangerous for secure token generation and handling.
- Hardware Requirements
 - A modern desktop or laptop for development and testing.

6.2.6 Interface/Exports

The back-end provides various interfaces and exports data to the front-end through defined operations:

- Flask Operations
 - HTTP methods like GET and POST to retrieve or submit user data.
 - Flask routing for front-end requests and API endpoints.
- Plotly
 - Functions for generating interactive graphs and visualizations based on user data and datasets.
- Data Types
 - Objects for encapsulating related functions and classes.
 - DataFrames from Pandas for handling data used in graph generation.
- Interactions
 - The back-end interacts with the database to store and retrieve user data.
 - The front-end displays data fetched from the database, while the back-end handles user authentication and security.

6.3 Database (Module) (Adam Dixon)

6.3.1 Responsibilities.

Data Storage

- Primary Storage: The database stores critical application data, including user information and datasets. It acts as the main repository for persistent data.

- **Metadata Management:** It manages metadata related to the datasets, such as upload dates and filenames, ensuring that this information is accurately recorded and easily retrievable.

Data Integrity

- **Data Consistency:** The database ensures data consistency through constraints, relationships, and validations. It maintains referential integrity by linking related tables using foreign keys.
- **Data Validation:** It is responsible for validating incoming data to ensure it conforms to the expected format and meets the necessary requirements, reducing the risk of data corruption.

Data Retrieval and Querying

- **Data Access:** The database provides mechanisms to query and retrieve data as required by other components. This includes simple lookups, complex queries, and relationships between tables.
- **Indexed Searches:** It supports efficient data retrieval through indexing, allowing for quick access to frequently searched fields, such as user emails or dataset filenames.

Data Security

- **Access Control:** The database has mechanisms to ensure only authorized users or components can access or modify the data. This might include user authentication and authorization.
- **Data Protection:** It is responsible for safeguarding sensitive data, such as hashed passwords, to prevent unauthorized access or data breaches.

Data Transactions

- **Transactional Consistency:** The database ensures that operations on data are transactional, meaning that they are atomic, consistent, isolated, and durable (ACID). This ensures that database operations are reliable and predictable.
- **Concurrency Control:** The database manages concurrent access to data, preventing race conditions or deadlocks when multiple users or components interact with the database simultaneously.

Data Backup and Recovery

- **Backup Strategy:** The database component is responsible for maintaining data backups, allowing for recovery in case of data loss or corruption.
- **Disaster Recovery:** It should support mechanisms for recovering from disasters, ensuring the resilience of the data storage.

6.3.2 Constraints

Storage and Size Constraints

- **Storage Capacity:** The database has a limited storage capacity, which must be considered when adding new data. If the volume of data exceeds this limit, it could cause performance issues or require additional storage resources.
- **Data Retention:** There may be constraints on how long data is retained in the database, potentially for compliance or data management reasons. This can affect how often data is archived or deleted.

Performance Constraints

- **Query Performance:** There may be constraints on query performance, especially with large datasets. Indexing and query optimization techniques are crucial to maintaining acceptable performance levels.
- **Concurrency Limits:** The database might have limitations on the number of concurrent connections or transactions. Exceeding these limits could lead to slowdowns or failures.

Data Integrity Constraints

- **Foreign Key Constraints:** The database enforces foreign key relationships to maintain data integrity. This means that you cannot delete a user if they have related records in the Datasets table.
- **Unique Constraints:** Fields like email in the User table must be unique. This constraint ensures data consistency but can cause issues if not properly handled.

Interaction Constraints

- **Data Access Permissions:** The database enforces permissions to ensure that only authorized users or components can access certain data. This can limit interaction if permissions are not configured correctly.
- **Synchronization and Locking:** To prevent data corruption, the database might use synchronization mechanisms or locks. This could lead to contention if multiple components try to access the same resource simultaneously.

Environmental Constraints

- **Database Server:** The database relies on a stable server environment. Constraints could arise from server capacity, resource allocation, or network issues.
- **Backup and Recovery:** Constraints related to backup and disaster recovery processes. Frequent backups might be required, affecting system performance.

6.3.3 Composition

- **Tables:** The database consists of several tables. In the provided code, these are:

- User: Stores user information such as email, password, first name, and last name.
- Datasets: Stores metadata about datasets, including filename, upload date, and user ID.
- Indexes: Although not explicitly mentioned in the code, indexes can be part of the database composition to speed up data retrieval. If used, you would describe what fields are indexed and why.
- Relationships: The User table has a one-to-many relationship with the Datasets table. This is established through the foreign key `user_id` in the Datasets table and the `db.relationship` in the User class.

6.3.4 Uses/Interactions

Visulyfe's database stores user data collected from back-end or front-end systems. The database stores the user data for first-time users signing up, and can help the back-end check for returning users who are logging back in. The back-end retrieves datasets from the database to calculate graphs, and sends that to the front-end to be displayed.

6.3.5 Resources

Resources used/needed are as follows:

- Visual Studio Code
 - coding software used
- SQL
 - coding language used
- Kaggle
 - a separate API with data for the graphs
- GitHub
 - collaborate with others with this software

6.3.6 Interface/Exports

- Data Access Methods:
 - `db.Model`: The base class for defining database models.
 - `db.relationship`: Establishes relationships between tables.
 - `db.Column`: Defines a column in a table with a specific data type.
- CRUD Operations: Using SQLAlchemy we created a database that handles basic Create Read Update and Delete operations.
- Data Types:
 - Integer: Used for IDs and foreign keys.
 - String: Used for storing text data such as emails, filenames, and names.
 - DateTime: Used for storing date and time information, like upload dates.
- Interactions:
 - The backend accessing the database to store user data.

- The frontend displaying data fetched from the database.
- User authentication

7. Detailed Lower level Component Design

7.1 about.html (Adam Dixon)

7.1.1 Classification

Type: HTML File

Component: Templet

7.1.2 Processing Narrative (PSPEC)

When a user accesses the about.html file, the web browser will parse the HTML content and render it according to the specified structure and styling. The page will display information about the project, such as its mission, team members, contact information, and any other relevant details.

7.1.3 Interface Description

The about.html file has a simple user interface with basic navigation elements. It includes headings, paragraphs, lists, and images to present information in a structured and visually appealing manner.

7.1.4 Processing Detail

N/A

7.1.4.1 Design Class Hierarchy

There is no class hierarchy for this static HTML file. It contains HTML elements such as <h1>, <p>, , <a>, and to structure and format the content.

7.1.4.2 Restrictions/Limitations

TheThe about.html file is a static page and does not support dynamic content or user interaction beyond basic navigation. Any updates to the content must be made manually in the HTML file.

7.1.4.3 Performance Issues

There are currently no performance issues.

7.1.4.4 Design Constraints

The about.html file must adhere to standard HTML syntax and best practices to ensure proper rendering and cross-browser compatibility.

7.1.4.5 Processing Detail For Each Operation

- `<h1>` element: Displays the main heading for the page, usually containing the name of the organization or project.
- `<p>` element: Displays paragraphs of text containing information about the organization or project.
- `` element: Creates an unordered list to present information in a structured manner, such as a list of features, services, or team members.
- `<a>` element: Creates hyperlinks to other web pages or resources, allowing users to navigate to additional information.
- `` element: Embeds images into the page to enhance visual appeal and provide additional context for the content.

7.2 base.html (Steven)

7.2.1 Classification

Type: HTML File

Component: Template

7.2.2 Processing Narrative (PSPEC)

This template serves as the base for all other HTML templates, defining common structural elements, styles, and scripts that are consistent across all pages.

7.2.3 Interface Description

Includes global stylesheets, JavaScript files, and a responsive header that adjusts based on user authentication status. It defines the visual and interactive foundation for the application

7.2.4 Processing Detail

Handles the initial loading of the CSS and JavaScript resources that are required for rendering the basic components of the website.

7.2.4.1 Design Class Hierarchy

Parent Class: N/A

Child Classes: All other HTML templates extend base.html

7.2.4.2 Restrictions/Limitations

Depends heavily on external libraries for frontend frameworks, potentially leading to longer loading times if not managed properly.

7.2.4.3 Performance Issues

No significant performance issues; however, optimizations in script loading and external resource management is continually assessed.

7.2.4.4 Design Constraints

Must ensure compatibility with all major browsers and devices to maintain accessibility and user experience.

7.2.4.5 Processing Detail For Each Operation

Each operation within the template, such as user navigation and interaction responses, is handled by JavaScript, ensuring smooth transitions and dynamic content updates

7.3 home.html (Steven)

7.3.1 Classification

Type: HTML File

Component: Template

7.3.2 Processing Narrative (PSPEC)

Serves as the main dashboard view for users, displaying personalized content, links, and data visualizations based on user interactions and preferences.

7.3.3 Interface Description

Contains widgets and interactive elements such as charts, graphs, and user-specific notifications and recommendations.

7.3.4 Processing Detail

Manages dynamic content generation based on user data, employing AJAX calls for real-time updates without needing to reload the page.

7.3.4.1 Design Class Hierarchy

Parent Class: base.html

Child Classes: N/A

7.3.4.2 Restrictions/Limitations

The complexity of user data processing and real-time update mechanisms requires robust backend APIs.

7.3.4.3 Performance Issues

No major issues; continuous monitoring and optimizing of AJAX calls and client-side rendering are in place.

7.3.4.4 Design Constraints

UI must be responsive and capable of adjusting to various screen sizes and orientations without compromising on functionality.

7.3.4.5 Processing Detail For Each Operation

Includes fetching user data, rendering charts, and updating UI components based on interactions, all handled via JavaScript and Flask backend routes

7.4 login.html (Nicholas)

7.4.1 Classification

Type: File

Component: Template

7.4.2 Processing Narrative (PSPEC)

Handles user authentication, allowing users to log in to the system using their credentials. It also provides options for password recovery and user registration.

7.4.3 Interface Description

Features form inputs for email and password, along with buttons for submission and navigation to registration or password recovery.

7.4.4 Processing Detail

Incorporates form validation both client-side and server-side, with secure transmission of credentials and error handling for login issues.

7.4.4.1 Design Class Hierarchy

Parent Class: base.html

Child Classes: N/A

7.4.4.2 Restrictions/Limitations

Relies on secure server-side authentication mechanisms, necessitating strong encryption and security protocols.

7.4.4.3 Performance Issues

There are currently no performance issues.

7.4.4.4 Design Constraints

Must adhere to security best practices, including secure storage and handling of user credentials and session data.

7.4.4.5 Processing Detail For Each Operation

Operations include validating user input, authenticating credentials against the database, and managing user sessions.

7.5 newPassForm.html (Nicholas)

7.5.1 Classification

Type: File

Component: Template

7.5.2 Processing Narrative (PSPEC)

This template is used for users to input a new password during the password reset process.

7.5.3 Interface Description

Features form fields for new password entry and password confirmation, with validation to ensure security requirements are met.

7.5.4 Processing Detail

Manages the password update operation with checks for password strength and match before submission.

7.5.4.1 Design Class Hierarchy

Parent Class: base.html

Child Classes: None

7.5.4.2 Restrictions/Limitations

Must enforce password complexity requirements to enhance security and safety of user information.

7.5.4.3 Performance Issues

There are currently no performance issues.

7.5.4.4 Design Constraints

Password fields must be securely handled to prevent unauthorized access.

7.5.4.5 Processing Detail For Each Operation

Includes collecting, validating, and updating the user's password in the database securely.

7.6 passrecovery.html (Nicholas)

7.6.1 Classification

Type: File

Component: Template

7.6.2 Processing Narrative (PSPEC)

Handles the interface for users to initiate a password recovery request by submitting their registered email.

7.6.3 Interface Description

Contains an email input form that allows users to submit a request for password recovery.

7.6.4 Processing Detail

Sends a password reset link to the provided email after verifying its existence in the database.

7.6.4.1 Design Class Hierarchy

Parent Class: base.html

Child Classes: None

7.6.4.2 Restrictions/Limitations

Dependent on the email service provider for delivering password reset emails.

7.6.4.3 Performance Issues

There are currently no performance issues.

7.6.4.4 Design Constraints

Must handle email data securely and ensure the password reset process is robust against attacks.

7.6.4.5 Processing Detail For Each Operation

Verifies user email, generates a secure link, and sends an email with the password reset instructions.

7.7 signup.html (Nicholas)

7.7.1 Classification

Type: File

Component: Template

7.7.2 Processing Narrative (PSPEC)

Used for new user registration, collecting essential information such as name, email, and password.

7.7.3 Interface Description

Includes multiple input fields for user data, such as first name, last name, email, and password, along with validation feedback.

7.7.4 Processing Detail

Processes user registration requests, including validation of input data and creation of new user accounts in the database.

7.7.4.1 Design Class Hierarchy

Parent Class: base.html

Child Classes: None

7.7.4.2 Restrictions/Limitations

Must ensure all user input is sanitized to prevent SQL injection and other security threats.

7.7.4.3 Performance Issues

There are currently no performance issues.

7.7.4.4 Design Constraints

Must comply with legal requirements for data protection and user privacy.

7.7.4.5 Processing Detail For Each Operation

Includes checking the uniqueness of the email, encrypting the password, and storing user data securely

7.8 `__init__.py` (Adam Dixon)

7.8.1 Classification

The `__init__.py` file is a Python package initialization file. It contains code that is executed when the package is imported, making it suitable for initializing package-level variables, importing submodules, and setting up the package's environment.

7.8.2 Processing Narrative (PSPEC)

When the package is imported, the Python interpreter executes the code in `__init__.py`. This file contains setup and initialization tasks, such as configuring package-level settings and making submodules available for other modules within the package. It also includes package metadata, version information, and dependency checks.

7.8.3 Interface Description

The `__init__.py` file does not have a user interface, as it is a background script that runs when the package is imported.

7.8.4 Processing Detail

7.8.4.1 Design Class Hierarchy

There is no class hierarchy in `__init__.py` since it does not define classes, but rather performs package-level initialization tasks.

7.8.4.2 Restrictions/Limitations

The code in `__init__.py` must be Python-compatible and should be written to handle any potential errors or exceptions that may occur during package initialization.

7.8.4.3 Performance Issues

There are currently no performance issues.

7.8.4.4 Design Constraints

The `__init__.py` file only contains initialization and setup tasks and avoids complex logic or application-specific code. It must follow Python syntax and best practices.

7.8.4.5 Processing Detail For Each Operation

The operations in `__init__.py` can vary greatly depending on the package's requirements. Some common operations include:

- `db = SQLAlchemy()`: Instantiates the SQLAlchemy object `db`, which will be used for managing the database within the Flask application..
- `mail = Mail()`: Instantiates the Flask-Mail object `mail`, which will be used for handling email-related functionality within the application.
- `api = KaggleApi()`: Creates an instance of the Kaggle API object `api`.
- `def create_app()`:: Defines a function `create_app()` for creating and configuring the Flask application.
- `app.config['SECRET_KEY'] = 'visualkey'`: Sets the secret key for the Flask application to 'visualkey', which is used for session management and security purposes.
- The `create_database(app)` function checks for and creates the SQLite database file if it does not exist.
- The `load_user(id)` function retrieves a User object from the database based on the given ID and returns it for use by the LoginManager.
- The `search_datasets()` function searches Kaggle datasets using a query parameter, formats the results, and returns them as a JSON response, handling any potential exceptions during the API call.

7.9 auth.py (Nicholas)

7.9.1 Classification

Type: File

Component: Module

7.9.2 Processing Narrative (PSPEC)

This module handles authentication logic, including user login, logout, and session management.

7.9.3 Interface Description

Provides functions for user authentication, including password verification and session creation.

7.9.4 Processing Detail

Ensures secure user authentication and session management, interfacing with the database to validate credentials.

7.9.4.1 Design Class Hierarchy

Parent Class: None

Child Classes: Utilizes Flask-Login for managing user sessions.

7.9.4.2 Restrictions/Limitations

Dependent on secure password hashing and environmental variables for configuration.

7.9.4.3 Performance Issues

Optimized for performance; however, dependent on database response times.

7.9.4.4 Design Constraints

Must adhere to best practices for secure authentication and data protection.

7.9.4.5 Processing Detail For Each Operation

Handles operations like user login, user logout, password hashing, and session validation.

7.10 models.py (Nicholas)

7.10.1 Classification

Type: File

Component: Module

7.10.2 Processing Narrative (PSPEC)

Defines the database models and their relationships, facilitating data storage and retrieval operations.

7.10.3 Interface Description

Contains class definitions for each database model, including methods for querying the database.

7.10.4 Processing Detail

Manages ORM interactions, ensuring efficient and secure communication with the database.

7.10.4.1 Design Class Hierarchy

Parent Class: None

Child Classes: Each model acts as a child of SQLAlchemy's Base class.

7.10.4.2 Restrictions/Limitations

7.10.4.3 Performance Issues

Performance is optimized through efficient query design and database indexing.

7.10.4.4 Design Constraints

Must ensure that all model fields are properly indexed and queries are optimized for performance.

7.10.4.5 Processing Detail For Each Operation

Includes CRUD operations for each model, relationship management, and data validation.

7.11 grapher.py (Fernando)

7.11.1 Classification

Type: File

Component: Module

7.11.2 Processing Narrative (PSPEC)

Responsible for generating and managing graphical data representations for the application.

7.11.3 Interface Description

Provides APIs to create, update, and display various types of graphs based on user data.

7.11.4 Processing Detail

Handles the computation and rendering of graphs, integrating with front-end technologies for display.

7.11.4.1 Design Class Hierarchy

Parent Class: None

Child Classes: Utilizes libraries such as Matplotlib or D3.js for graph rendering.

7.11.4.2 Restrictions/Limitations

Graphical rendering is dependent on the efficiency of the underlying library.

7.11.4.3 Performance Issues

Optimizations are required to handle large datasets and real-time data rendering.

7.11.4.4 Design Constraints

Graphs must be responsive and compatible across different devices and browsers.

7.11.4.5 Processing Detail For Each Operation

Includes data fetching, data processing, graph generation, and updating graphs dynamically based on user interaction.

7.12 views.py(Geo)

7.12.1 Classification

This component is classified as a subsystem within the Flask application, focusing specifically on managing user views related to login processes.

7.12.2 Processing Narrative (PSPEC)

- Login View ('/'): Manages the login process, including user authentication and session initialization.

7.12.3 Interface Description

- Inputs: User credentials (email and password) entered through the login form.
- Outputs: User feedback via flash messages and redirections either to the home page upon successful login or back to the login page with the error message.

7.12.4.1 Design Class Hierarchy

The 'views' Blueprint is an extension of Flask's Blueprint class, designed to handle specific routes related to user interactions that don't involve direct database manipulation beyond user validation.

7.12.4.2 Restrictions/Limitations

- The system is designated to handle user login with pre-registered email addresses only.

- Password validation strictly required hashed comparison, limiting quick modifications or bypasses.

7.12.4.3 Performance Issues

- Reliance on database lookups for every login attempt could slow down performance under high-load scenarios, especially if the user database grows significantly.

7.12.4.4 Design Constraints

- The system must adhere to security best practices, especially in handling user passwords and sessions.

7.12.4.5 Processing Detail For Each Operation

- **Login**
 - Retrieves user input from the form.
 - Performs a database query to fetch the user by email.
 - If the user exists, it compares the hashed password stored in the database with the one provided using a secure hash check.
 - If the password matches, the user is logged in and redirected to the home page.
 - If there is no match or user does not exist, appropriate error messages are flashed and the user is prompted to try again or register.

7.13 main.py (Jacob)

7.13.1 Classification

The main.py file is the driving file for all of the code

7.13.2 Processing Narrative (PSPEC)

The Python interpreter always starts the execution of code at the main function or file. From there, Python will go to any other files that initialize and configure all packages of code, like `__init__.py`, to make sure the program has everything it needs to run.

7.13.3 Interface Description

The main.py file does not have any user interface.

7.13.4 Processing Detail

It acts as the first domino in the operation that starts the code. The program cannot run without it.

7.13.4.1 Design Class Hierarchy

There is no class hierarchy in main.py

7.13.4.2 Restrictions/Limitations

The program is too complex to be solely contained within main.py

7.13.4.3 Performance Issues

There are currently no performance issues.

7.13.4.4 Design Constraints

Refer to 7.13.4.3

7.13.4.5 Processing Detail For Each Operation

The file imports the folder containing the other files, so the program can be run.

7.14 .gitignore (Adam Dixon)

7.14.1 Classification

The file specifies intentionally untracked files that Github should ignore.

7.14.2 Processing Narrative (PSPEC)

The .gitignore file creation process involves creating a plain text file named .gitignore in the root directory of the Git repository and adding patterns or filenames of files and directories that should be ignored by Git.

7.14.3 Interface Description

The .gitignore file is a simple text file that developers create and manage within the Git repository, using any text editor or the GitHub web interface.

7.14.4 Processing Detail

Creating a .gitignore file involves the following steps: Navigate to the root directory of the Git repository. Create a new plain text file named .gitignore. Open the file using a text editor or the GitHub web interface. Add patterns or filenames of files and directories that should be ignored by Git. Save the file.

7.14.4.1 Design Class Hierarchy

N/A

7.14.4.2 Restrictions/Limitations

The .gitignore file can only ignore files that are not already tracked by Git.

7.14.4.3 Performance Issues

There are no significant performance issues related to the .gitignore file.

7.14.4.4 Design Constraints

The .gitignore file must follow specific patterns and rules for ignoring files, as defined by the Git documentation.

7.14.4.5 Processing Detail For Each Operation

Add a pattern or filename to the .gitignore file. Save the file. Git automatically ignores the specified files or directories when committing changes to the repository.

7.15 Dockerfile (Adam Dixon)

7.15.1 Classification

The Dockerfile is a configuration file used by Docker to build Docker images. It contains a set of instructions that specify the environment, dependencies, and file structure of the Docker image.

7.15.2 Processing Narrative (PSPEC)

The Dockerfile creation process involves creating a text file named Dockerfile in the project's root directory and adding instructions to build a Docker image with the desired environment, dependencies, and file structure.

7.15.3 Interface Description

The Dockerfile is a text file that developers create and manage within their IDE or text editor. Docker reads the instructions in this file to build Docker images.

7.15.4 Processing Detail

Creating a Dockerfile involves the following steps: Create a text file named Dockerfile in the project's root directory. Define the base image for the Docker image. Add instructions for setting up the environment, installing dependencies, adding project files, exposing ports, and defining the entry point. Save the file.

7.15.4.1 Design Class Hierarchy

N/A

7.15.4.2 Restrictions/Limitations

The Dockerfile must be named precisely as "Dockerfile" (case-sensitive) for Docker to recognize it. Each instruction in the Dockerfile creates a new layer in the Docker image, which can affect image size and build time. Build-time variables and multi-stage builds can increase complexity and maintenance requirements.

7.15.4.3 Performance Issues

Inefficient or incorrect usage of Dockerfiles can result in larger Docker images, longer build times, and potential runtime performance issues.

7.15.4.4 Design Constraints

The Dockerfile must adhere to Docker's syntax and best practices, including proper ordering of instructions, using specific instruction types for certain tasks, and minimizing the number of layers for efficient image building.

7.15.4.5 Processing Detail For Each Operation

1. FROM <base_image>: Specify the base image for the Docker image.
2. RUN <command>: Execute a command to install dependencies, set environment variables, or modify the filesystem.
3. ADD <src> <dest> or COPY <src> <dest>: Add or copy files from the host to the Docker image.
4. EXPOSE <port>: Inform Docker that the specified port should be accessible to the container.
5. ENTRYPOINT <command>: Define the command to be executed when the container starts.
6. CMD <default_parameters>: Set default parameters for the command defined in ENTRYPOINT.

7.16 requirements.txt (Adam Dixon)

7.16.1 Classification

The requirements.txt file is a component of a Python project that lists the project's Python dependencies with their respective versions.

7.16.2 Processing Narrative (PSPEC)

The requirements.txt file creation process involves creating a plain text file named requirements.txt in the project's root directory and adding a list of Python packages with their versions required to run the project.

7.16.3 Interface Description

The requirements.txt file is a simple text file that developers create and manage within their IDE or text editor.

7.16.4 Processing Detail

Creating a requirements.txt file involves the following steps: Create a text file named requirements.txt in the project's root directory. Identify the Python packages and their respective versions required to run the project. Add each package to the file, specifying the version using the following format: package_name==package_version. Save the file.

7.16.4.1 Design Class Hierarchy

N/A

7.16.4.2 Restrictions/Limitations

Package names and versions must be accurately specified to ensure the project can successfully install and run all required dependencies. The file should be regularly updated as the project evolves and new dependencies are added or removed.

7.16.4.3 Performance Issues

There are no significant performance issues related to the requirements.txt file.

7.16.4.4 Design Constraints

The requirements.txt file must adhere to the format recognized by Python package managers, with each package and its version specified on a new line.

7.16.4.5 Processing Detail For Each Operation

1. pip freeze: Generate a list of installed Python packages with their versions.
2. pip install -r requirements.txt: Install all packages listed in the requirements.txt file with the specified versions.

8. Database Design (Steven)

SQL (Structured Query Language) is a powerful tool for managing and querying databases. SQL will play a crucial role in handling the database where user information and datasets are stored.

8.1 User Profile Table

user_id	user_firstname	user_lastname	user_email	user_password
001	example1_firstname	example1_lastname	example@gmail.com	password
002	example2_firstname	example2_lastname	example2@gmail.com	password2

user_id: User's unique id

user_firstname: User's first name

user_lastname: User's last name

user_email: User's account email

user_password: User's account password

8.2 User Favorites Table

favorite_id	user_id	link_url	link_title
001	001	https://example.com/database1	database1

favorite_id	user_id	link_url	link_title
002	001	https://example.com/database2	database2
003	002	https://example.com/database1	database1

favorite_id: Unique id for each favorited link

user_id: User's unique id

link_url: URL of favorited link

link_title: Title of favorited link

8.3 User Datasets Table

dataset_id	user_id	dataset_title	filepath
001	001	example_title1	/user1/datasets/exmaple_title1.csv
002	001	example_title2	/user1/datasets/exmaple_title2.csv
003	002	example_title3	/user2/datasets/exmaple_title3.csv

dataset_id: Unique id for each imported dataset

user_id: User id for which the user imported the dataset

dataset_title: Title of the dataset

filepath: path to the file where the dataset is stored

9. User Interface (Ryan)

9.1 Overview of User Interface

The functionality of Visulyfe's UI from the user's perspective will be aimed to be fluid, and easy to use. The user will first be met with the login page, which after logging in, then the user will be directed to the Visulyfe home page. Once there the user can view and manipulate any previous data set, charts, and graphs that they'd like to. On the home page as well, there will be links at the top of the page to redirect to other useful pages that Visulyfe has to offer. They'll also be given the option to enter a new data set and generate different charts and graphs. These graphs will be fully customizable, and the user can customize them as their heart desires. They'll be able to save any new and edited data to their profile, should they wish to end the session they can rest assured their data will be there safe and secure when they want to come back.

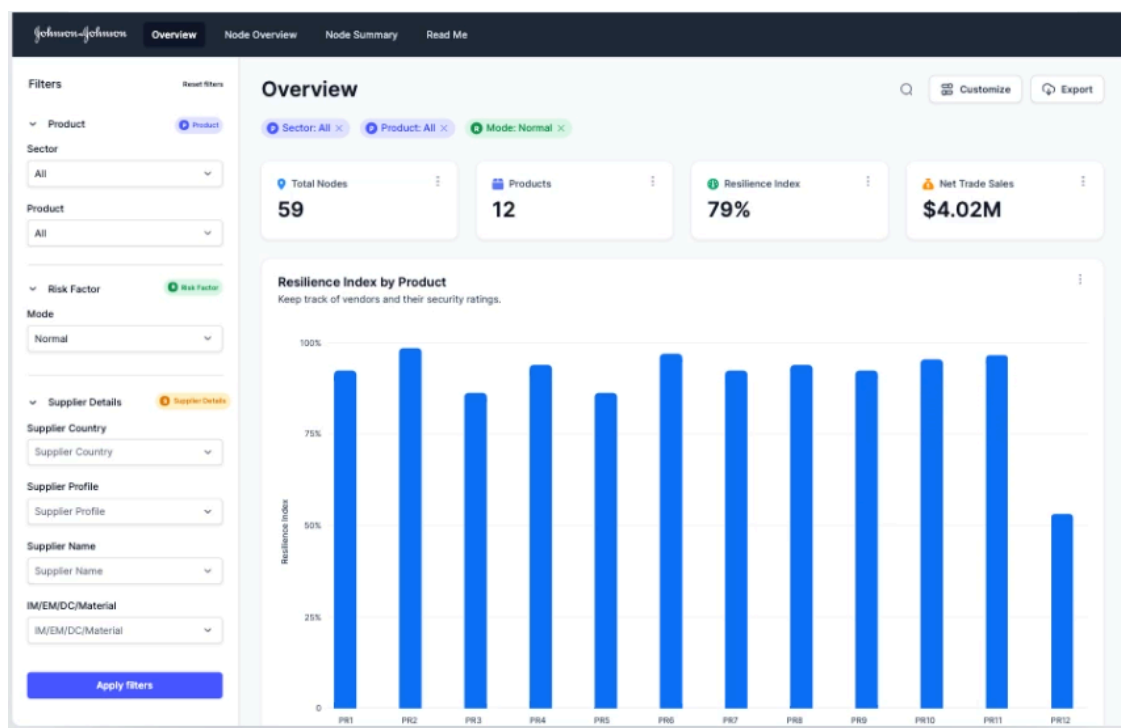
9.2 UX Standards

The Visulyfe team will adhere to these guidelines:

- Zeigarnik Effect
 - People remember uncompleted or interrupted tasks better than completed tasks.
- Peak-End Rule
 - People judge an experience largely based on how they felt at its peak and at its end, rather than the total sum or average of every moment of the experience.
- Law of Uniform Connectedness
 - Elements that are visually connected are perceived as more related than elements with no connection.
- Jakob's Law
 - Users spend most of their time on other sites. This means that users prefer your site to work the same way as all the other sites they already know.
- Aesthetic-Usability Effect
 - Users often perceive aesthetically pleasing design as design that's more usable.
- Doherty Threshold
 - Productivity soars when a computer and its users interact at a pace (<400ms) that ensures that neither has to wait on the other.

9.3 Screen Frameworks or Images

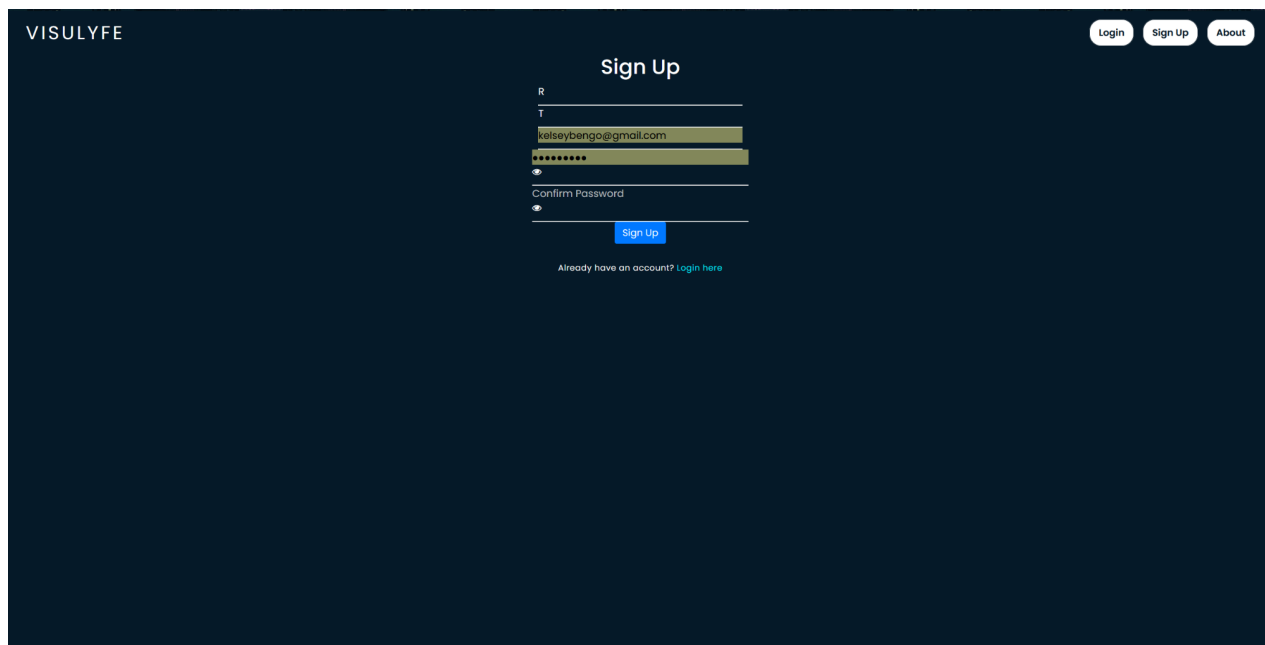
These can be mockups or actual screenshots of the various UI screens and popups.



This mockup will serve as inspiration for the end product of Visulyfe's UI.

The login page features a dark blue background with the Visulyfe logo in the top left. In the top right, there are links for 'Login', 'Sign Up', and 'About'. The main content area is titled 'Welcome' and contains a login form with fields for email and password. Below the form is a 'Login' button. At the bottom, there are links for 'Need an Account? Please Register Here' and 'Forgot Your Password? Recover Password Here'.

Current UI for the login of Visulyfe



The image shows the 'Sign Up' page for Visulyfe. The page has a dark blue background. In the top left corner, the word 'VISULYFE' is written in white. In the top right corner, there are three buttons: 'Login', 'Sign Up', and 'About'. The 'Sign Up' button is highlighted. The main content area is centered and contains the following form fields: a first name field with the letter 'R' above it, a last name field with the letter 'T' above it, an email field with the text 'kelseybengo@gmail.com', a password field with a strength indicator (a bar with dots) and a toggle icon, and a 'Confirm Password' field with a toggle icon. Below the password fields is a blue 'Sign Up' button. At the bottom of the form, there is a link that says 'Already have an account? Login here'.

VISULYFE

Sign Up

R

T

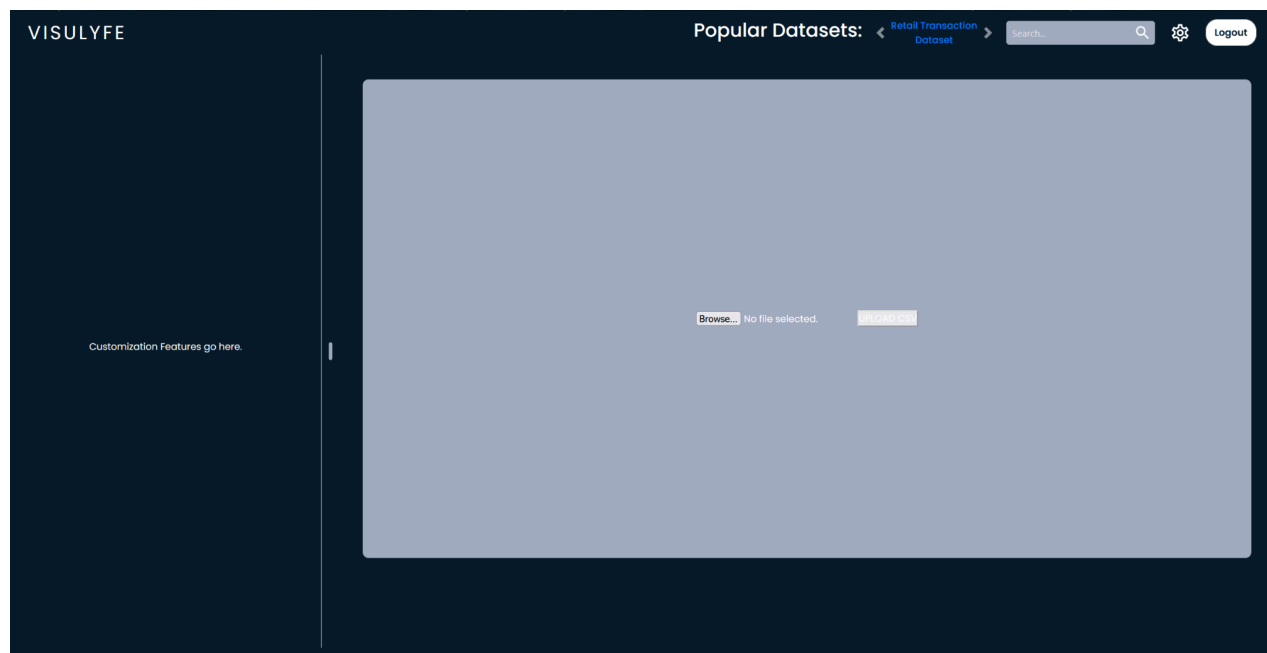
kelseybengo@gmail.com

Confirm Password

Sign Up

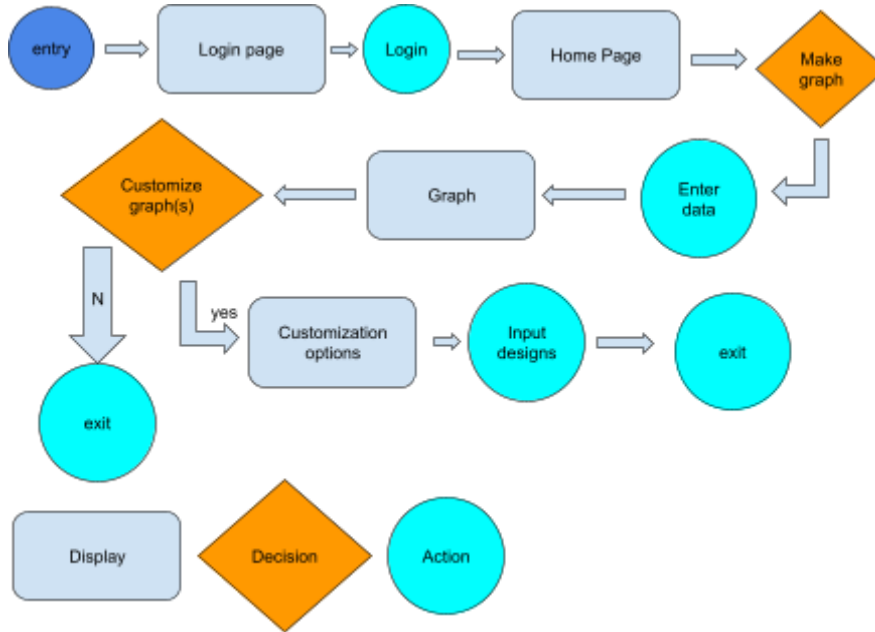
Already have an account? [Login here](#)

Sign up page for Visulyfe



Current Home Page for Visulyfe

9.4 User Interface Flow Model



Flow chart for how a user should use our app.

10. Requirements Validation and Verification (Adam)

Requirement	Component Module	UI Elements/Low Level Components	Testing Method
Validity checks on the inputs	Data Management System (MySQL)	Web Interface, Data Input Devices	Test input of invalid data and ensure the system rejects it appropriately.
Exact sequence of operation	Programming Language and Framework (Python), Web Framework (Flask)	Web Interface, GUI, Graph Display	Create test cases to validate the sequence of operations outlined in the system flow.
Responses to abnormal situations (e.g. overflow)	Error Handling, Programming Language and Framework (Python), Web Framework	Web Interface, GUI, Feedback Mechanism	Simulate overflow scenarios and verify that the system detects and handles errors gracefully.

	(Flask)		
Accurate representation of input/fetched data	Data Management System (MySQL), Graph Display	Web Interface. GUI, Graph Display	Input known data and verify that the generated visual output matches the input accurately.
Web Interface	Web Framework (Flask)	Responsive Design, GUI, Feedback Mechanism, Security	Perform usability testing on different browsers (e.g., Safari, Chrome, OperaGX, Edge) to ensure compatibility.
Responsive Design	Web Framework (Flask)	GUI, Display Devices	Test the application on different screen sizes and resolutions to ensure dynamic resizing and layout adjustments.
GUI	Web Framework (Flask)	Customization Options, Error Handling	Conduct user acceptance testing to ensure ease of interaction and functionality.
Graph Display	Graph Display(Plotly)	Customization Options	Input various datasets and verify that the generated graphs are accurate and customizable.
Customization Options	GUI	Web Interface, Responsive Design	Allow users to customize graph appearance and verify changes are applied as expected.
Error Handling	Error Handling	GUI, Feedback Mechanism	Test accessibility and completeness of the help documentation and reference manual.
Help and Documentation	Web Interface, Programming Language and Framework (Python), Web Framework (Flask)	GUI, Feedback Mechanism	Test accessibility and completeness of the help documentation and reference manual.
Feedback Mechanism	Feedback Mechanism	Web Interface, GUI	Monitor system updates and bug fixes, and ensure users receive timely notifications.
Security	Security	Web Interface, GUI	Perform penetration testing to ensure user data is securely stored and encrypted.
Data Input Devices	Data Input Devices	Web Interface, GUI	Test usability and functionality with various input devices (e.g., keyboard, mouse).
Display Devices	Display Devices	Web Interface, Graph Display	Ensure the application can output visualizations to different devices such as screens, monitors, or projectors.
Data Management System (MySQL)	Data Management System (MySQL)	Programming Language and Framework	Validate data storage, retrieval, and integrity through unit and integration testing.

		(Python), Web Framework (Flask)	
Programming Language and Framework (Python)	Programming Language and Framework (Python)	Web Interface, GUI, Data Management System (MySQL)	Test functionality and performance of Python code components.
Web Framework (Flask)	Web Framework (Flask)	Web Interface, GUI, Data Management System (MySQL)	Verify routing, request handling, and rendering capabilities through integration testing.
Kaggle API	Programming Language and Framework (Python)	Web Interface, Data Management System (MySQL)	Test API integration by fetching data from Kaggle and verifying its availability for visualization.

11. Glossary(Adam)

1. Software Design Document (SDD): A comprehensive document that outlines the design goals, architecture, and development considerations for a software project.
2. Scope of the Project: The boundaries and objectives of the project, defining what is included and excluded from the project's deliverables and activities.
3. Change Management: The process of planning, implementing, and controlling changes to a software project in a systematic manner to minimize risks and ensure successful outcomes.
4. Feedback Loop: A process where information about the performance of a system is used to make adjustments or improvements, creating a continuous cycle of improvement.
5. Object-Oriented Design: A design methodology that focuses on modeling software systems as a collection of objects that interact with each other to accomplish tasks.
6. Model View Controller (MVC): An architectural pattern used in software design, where the application is divided into three interconnected components: the model (data and business logic), the view (presentation layer), and the controller (handles user input and updates the model).

7. Flask: A lightweight web application framework for Python, used for developing web applications quickly and with minimal boilerplate code.
8. SQL (Structured Query Language): A domain-specific language used for managing and querying relational databases.
9. Kaggle API: An interface provided by Kaggle, a platform for data science and machine learning competitions, allowing users to access datasets, competitions, and other resources programmatically.
10. Agile Development Methodology: An iterative and incremental approach to software development, emphasizing flexibility, collaboration, and customer feedback.
11. GitHub Repository: A remote location where developers can store and manage their source code, enabling collaboration and version control.
12. IDE (Integrated Development Environment): A software application that provides comprehensive facilities to programmers for software development, typically including a code editor, compiler, and debugger.
13. Peak-End Rule: A psychological principle stating that people tend to judge an experience based on how they felt at its peak (the most intense moment) and at its end, rather than considering the total duration or average of the experience.
14. Law of Uniform Connectedness: A Gestalt principle of perception stating that elements that are visually connected are perceived as more related than elements with no connection.
15. Aesthetic-Usability Effect: The phenomenon where users perceive aesthetically pleasing design as more usable, even if it offers no functional advantage.
16. Doherty Threshold: A principle stating that productivity increases when a computer and its users interact at a pace where neither has to wait on the other, typically within 400 milliseconds.
17. Data Flow Diagram (DFD): A graphical representation of the flow of data within a system, depicting the processes, data stores, and data flows involved in a system's operation.

12. References(Adam)

Software Requirements Specification for Visulyfe Version 1.0:

Title: "XYZ Software System Requirements Specification"

Author: Adam Dixon, Fernando Serrano Perez, Geovanny Montano, Jacob Hertz, Nicholas Trigueros, Ryan Torrez, Steven Partida, Washika Afrozi

Version Number: 1.0

Date: 03/06/2024

Source/Location: <https://github.com/csula-cs3337swe/202401Group4-repo/issues/50>

Software Test Plan for Visulyfe Version 1.0:

Title: "Software Test Plan for Visulyfe"

Author: Adam Dixon, Fernando Serrano Perez, Geovanny Montano, Jacob Hertz, Nicholas Trigueros, Ryan Torrez, Steven Partida, Washika Afrozi

Version Number: 1.0

Date: 03/06/2024

Source/Location: <https://github.com/csula-cs3337swe/202401Group4-repo/issues/46>