RTS Analyzer: Sprint One

Software Engineers:

* Cody Strange
* Jaden Albrecht
* Hoyoung Kim

Institution: Utah Valley University

Course: CS4550-601 2024 Spring

# Organization

## Coding Standards

### Commenting

* Comment each function by what it should be doing
* Comment each class by what it should be doing

### Programming Paradigm

* Object oriented

### Naming Conventions

* Classes: CapWords
* Functions: snake\_case
* Variables: snake\_case
* Constants: ALLCAPS
* Files/Folders: snake\_case

## Scheduling

### Meetings

* Whenever people feel like it

# Requirements Gathering

## Functional Requirements

* Analyze replays from user’s computer.
  + Determine build order
  + Determine win rates based on races
  + Determine win rates of build ‘A’ vs build ‘B’
* Analyze replays from professional tournaments
  + Determine build order
  + Determine win rates based on races
  + Determine win rates of build ‘A’ vs build ‘B’
* Import and display build order to live game.

## Non-Functional Requirements

Summary: *These requirements do not pertain to specific behaviors or functionalities of the application but rather to its overall attributes and characteristics.*

##### Performance and Responsiveness

* The application should be capable of processing and displaying data with minimal latency.
* It should handle high volumes of concurrent users and data requests efficiently.

##### Scalability

* The system should be scalable to accommodate a growing number of users and an increasing amount of data.

##### Reliability and Availability

* The application should have high uptime, with minimal downtime for maintenance or updates.
* It should be reliable in delivering accurate and consistent analytics data.

##### Security

* Strong measures for data security, including encryption of sensitive data and secure handling of user information.
* Implementation of proper authentication and authorization mechanisms to protect user accounts and data.

##### Maintainability and Modularity

* The codebase should be well-organized and documented for ease of maintenance and updates.

##### Usability and Accessibility

* The user interface should be intuitive and user-friendly, catering to both novice and experienced gamers.
* The application should be accessible to users with disabilities, complying with relevant accessibility standards.

##### Compliance and Legal Requirements

* Adherence to legal and regulatory requirements, such as data protection laws (e.g., GDPR, if applicable).

## Risk Analysis

### Technical Risks

##### Risk of Inaccurate Analysis

There's a risk that the program may not accurately analyze replays due to incorrect logic, outdated algorithms, or compatibility issues with different SC2 versions.

* Mitigation: Regularly update the program to align with the latest game patches, and thoroughly test the program with a variety of replays
* Contingency: Temporarily remove feature that is inaccurate until we can guarantee accuracy

##### Risk of Incompatibility with Future SC2 Updates

Future updates to SC2 might change the replay format or introduce new features not supported by the current program.

* Mitigation: Plan for regular updates and maintenance and stay informed about upcoming SC2 updates.
* Contingency: Make it so the program doesn’t accept replays past the date of the new update until the program is compatible with the new version of sc2 replays.

### Legal and Compliance Risks

##### Risk of Data Privacy Violations

If the analytics tool collects user data, it must comply with data protection regulations like GDPR or CCPA.

* Mitigation: Implement strong data privacy policies and only collect necessary data with user consent.
* Contingency: Shut down program until it complies with data protection regulations.

### Operational Risks

##### Risk of Dependency on External Libraries

The project might rely on external libraries (like sc2reader) which could become outdated or unsupported.

* Mitigation: None
* Contingency: Drop project

##### Risk of Insufficient Testing

Inadequate testing can lead to undetected bugs and issues in production.

* Mitigation: Implement comprehensive testing strategies, including unit tests, integration tests, and user acceptance tests.
* Contingency: Fix bugs, possibly rollback to previous version of product and add more comprehensive testing

# High Level Design

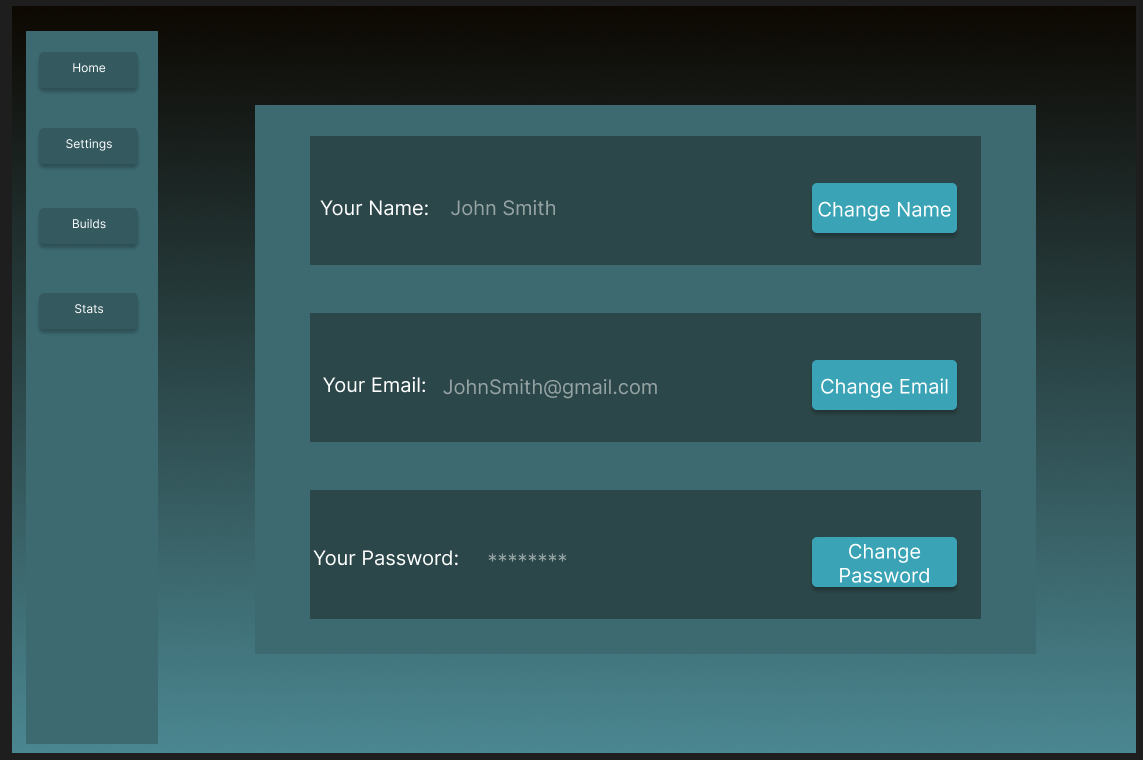
## Architecture

A diagram of a block diagram

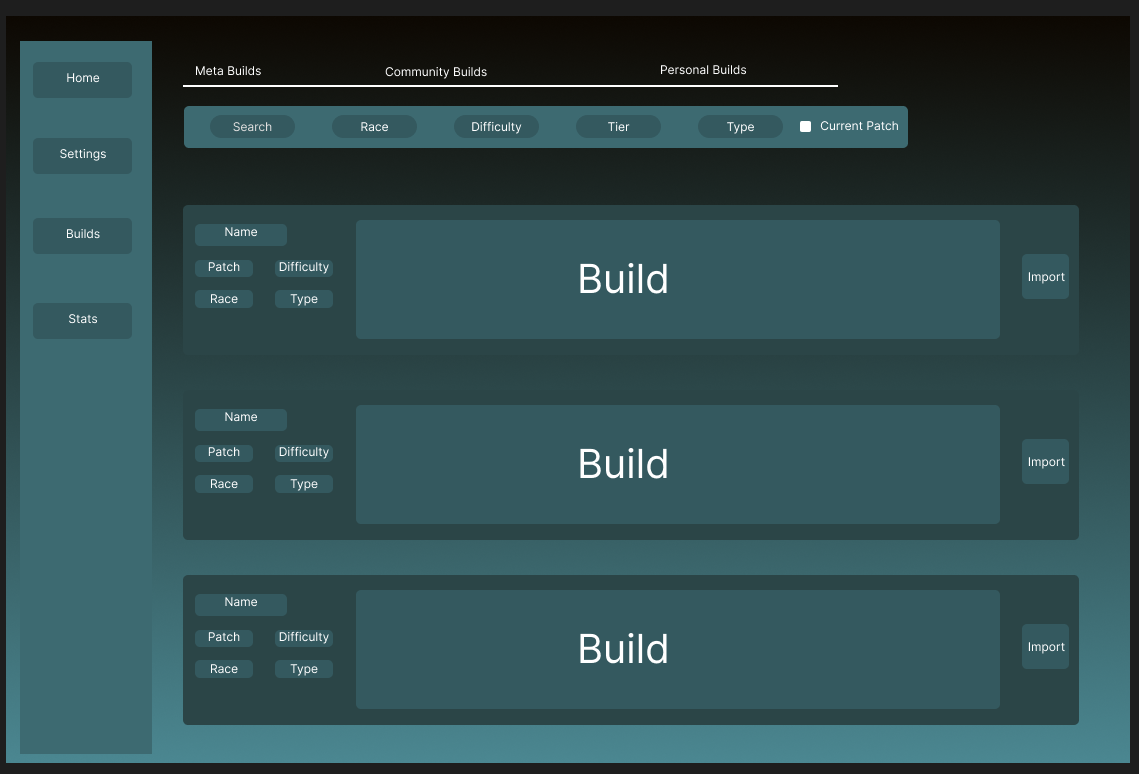
Description automatically generated

## Wireframes

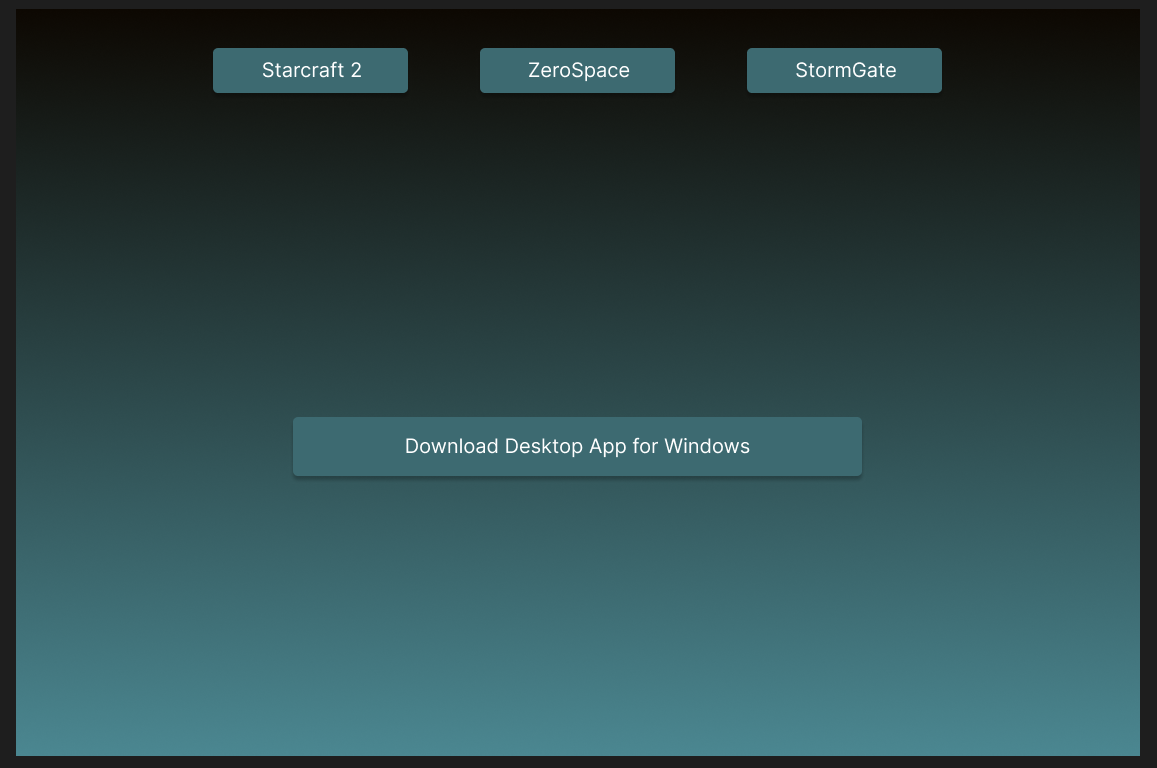
### Accounts



### Builds



### Home



### Stats Page

### 

# Low Level Design

# Development

## Prototype

### Extractor

### Database

### Analyzer

### UI

# Testing