Yuppy City Simulator: Final Report

Meandering Armadillos

Nick Montaquila, Michael Nishida, Andrew Ricci

[**Problem Statement**](#_5o1x68ii8qyg) **3**

[Objective](#_9z41rowp9fjn) 4

[Rationale](#_gvjyyzfrzol6) 4

[Existing Systems](#_k9paa0eijgtm) 4

[Proposed Systems](#_syw4dwdp4vth) 4

[**Team Details and Plan of Action**](#_bcj3elhh3qi0) **5**

[Team Roles](#_on7b0j906bto) 5

[Action Plans](#_uo7inmyhqaew) 5

[**Requirements Analysis**](#_u6dl9evay0sa) **5**

[**GENERAL DESCRIPTION**](#_zdlxprct485w) **5**

[User Characteristics](#_cofj47azzly9) 5

[Product Perspective](#_h1r6ipqs7jzt) 5

[Overview of Functional Requirements](#_9hi2weahjz51) 6

[USER REQUIREMENTS](#_irzn9c7wc7ze) 6

[APPLICATION REQUIREMENTS](#_h5z7yi4crhly) 6

[Overview of Data Requirements](#_83rf8i64y8xf) 6

[General Constraints, Assumptions, Dependencies, Guidelines](#_nke8f44oevpt) 7

[User View of Product Use](#_2basv6e5wwsr) 7

[**SPECIFIC REQUIREMENTS**](#_sqcuiji5a9nu) **8**

[External Interface Requirements](#_kecdlku15qbo) 8

[Detailed Description of Functional Requirements](#_if1ohh6r5em9) 8

[Detailed User Requirements](#_9hwn2y1ex0g7) 8

[Detailed Application Requirements](#_yme9thtilej) 12

[Performance Requirements](#_j892q38d8ay) 14

[Quality Attributes](#_oczfc2mfw1im) 14

[**System Design**](#_bh2h03jdwty8) **15**

[Major software requirements](#_nw2qgbsmft1u) 15

[Design constraints, limitations](#_kg806nwbej2z) 15

[Design Goals](#_294sc5ud4md4) 16

[Changes to requirements](#_l5gzt8ywyc3n) 16

[Overview of Document](#_ugmqx2i1g8u6) 16

[Data Design](#_gxeynrwmn0as) 16

[Data Objects and resultant data structures:](#_hmz5q2pw5y7n) 16

[File and Database Structures](#_1ofi4jxwnruu) 17

[File and Data Cross Reference   
N/A](#_mpv15igqj8ei) 17

[System Architecture Description](#_4000g0t1mcqu) 17

[Overview of Modules / Components:](#_8yqsgr2wehrc) 17

[Structure and relationships:](#_7zn8tv9c2mf9) 18

[Detailed description of components](#_5mfpfjqcjr6g) 18

[Algorithm](#_rvtxs3dwk9c4) 18

[SQL Database](#_p8aa6a9732m) 19

[User Interface: Quiz](#_baf3ej394i6f) 20

[User Interface: User Profile](#_waac34vg4i30) 20

[User Interface: General](#_hj3cyyxudtcu) 21

[UI Controller:](#_creymgb3zwju) 21

[Interface Design:](#_2ebpd9xl3yud) 22

[**Test Document**](#_nu6wsk7skxq8) **23**

[Algorithm Test Cases](#_9dmqby9u2j29) 23

[Controller Test Cases](#_7as8xb97vkni) 24

[Integration Testing](#_jiqq5gsjex0j) 26

[**User Manual / User Guide**](#_vw7rgazh5fjk) **26**

[**Glossary**](#_lwjl9cfwfmwg) **27**

[**References**](#_fijcm6ymdh28) **27**

# Problem Statement

A problem faced by many yuppies in today’s day and age of increased globality is that of location. Where exactly is it in this wondrous world that suits them best? There are so many factors that trying to search them all and compare becomes quite the headache, due to different studies done by different groups with little standardization or consistency. This can result in a confounded and disillusioned yuppie, with little more to do than sit in their favorite espresso bar and sulk.

However, this can all be solved by the Yuppie City Simulator Platform. A platform that will allow users to compare dozens of statistics from cities all over the world, weighed by their own preferences. A platform that contains rigidly standardized indexes that consider a variety of economic and geographical factors. A platform that will deliver consistently to users the place that suits them best, be it Salt Lake City or Seoul.

## Objective

With our YCS project, our objective was to create a working demonstration of this very concept. It would be far outside of the scope of the class and our current resources to take cities from all over the world, so instead we opted to compare statistics of the top 25 cities in the United States. If we could implement a weighting system for several major and easy to measure factors of these major cities, we could deliver something that could one day be built into the dream platform that would be coveted by yuppies everywhere.

## Rationale

Our rationale for YCS was to choose factors that were easy to differentiate and uncontroversial. Comparing the sunniness of Baltimore and San Francisco leaves little to be debated, so it was factors like these that we primarily based our quiz questions upon. We knew that the most fundamental feature of YCS would be the quiz, but there would need to be extraneous elements to expand upon this as well. If a user had taken a quiz and wanted to save their result, we would need to create a user login system and a quiz storage system as well. It was the needs of the users that primarily determined how we approached the design of our project.

## Existing Systems

Outside of the Yuppie City Simulator, there are few ways for a person to decide what city suits them best. They would likely have to sift through curated lists that really reflect a person’s opinion rather than exclusively hard data. Either that, or sifting through statistics like “livability” which do very little to truly inform the reader of how that statistic is derived. While these methods would likely give the user an indication of where the best place to live would be, there would be little truly enlightening them as to what the best place would be in regards to their personal tastes and priorities.

## Proposed Systems

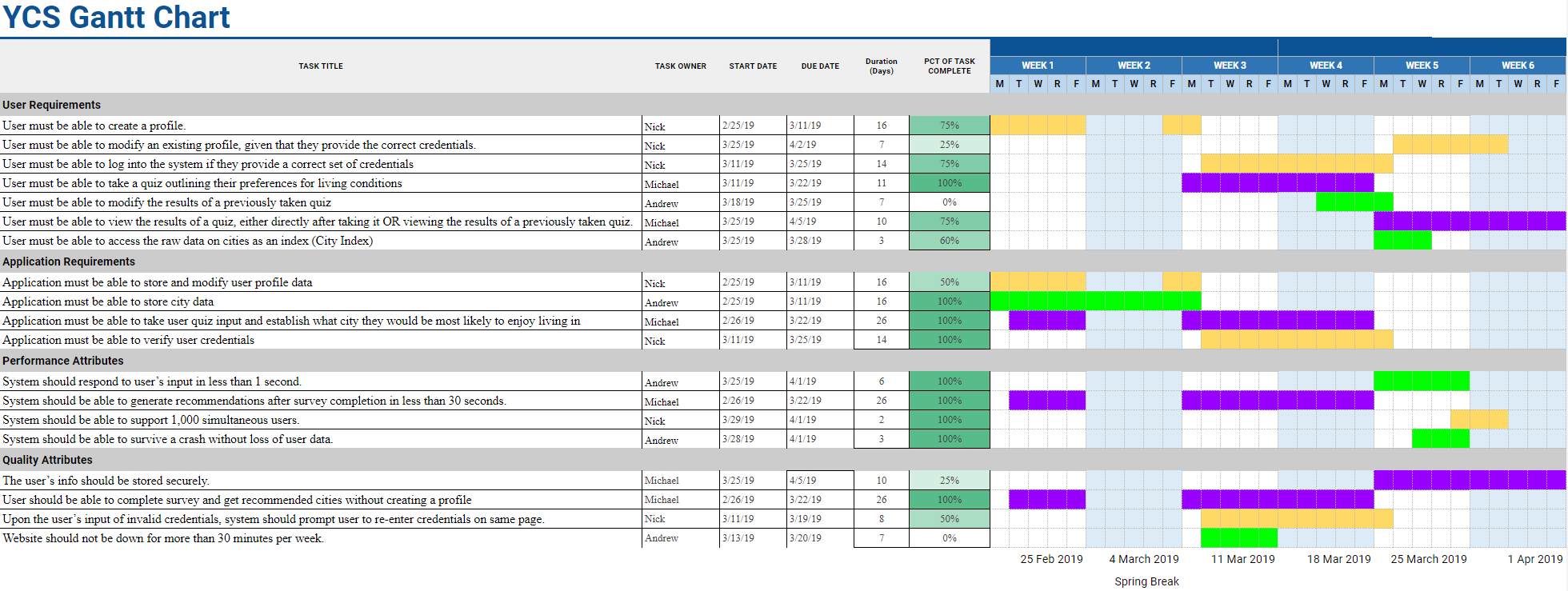
Our solution, the Yuppie City Simulator, is a quiz based website that allows users to choose from a variety of factors what they prefer in their desired city. They rate each factor from -10 to 10, with 10 being the highest priority and the highest weight, 0 being a null priority and no weight, and -10 being the lowest priority and the lowest weight. This allows users to specify exactly how important each element of the city selection is to them, and ignore certain factors entirely. The user can then view and update their prior quizzes, and view the data that Yuppie City Simulator uses to calculate city scores in its entirety to verify our process.

# Team Details and Plan of Action

## Team Roles

* + 1. Michael
       1. Backend specific algorithms
       2. Database interaction
       3. Recommendation algorithm
    2. Andrew
       1. Data collection
       2. Database setup and maintenance
       3. Database interaction
    3. Nick
       1. Front-end design
       2. Flask app setup and maintenance
       3. Web-specific backend
       4. Database interaction

## Action Plans



# Requirements Analysis

## GENERAL DESCRIPTION

### User Characteristics

Our users will expect a granular and detailed approach to city selection, as well as more general categories that can give a broad result that still appeals to the users’ choices. This will allow users to get a result they desire in a short time, or a more precise result with all categories entered. Users will also enjoy easy access to the YCS platform, through a straightforward login, simple profile UI and listed results of prior quizzes. The YCS City Index will also need to be easy to access and presented in a fashion that is easy to read and sort.

### Product Perspective

Yuppie City Simulator will be a stand alone website. Though all user interactions will be with data within the YCS site and databases, all data will be gathered from external sources. This means that YCS will be comprised of data from large university studies, compilations of local government statistics, and other aggregates like Wikipedia. Other than the initial collection of data in MA’s databases, all information will be accessed locally to the Yuppie City Simulator platform.

### Overview of Functional Requirements

#### USER REQUIREMENTS

* + - * 1. User must be able to create a profile.
        2. User must be able to modify an existing profile, given that they provide the correct credentials.
        3. User must be able to log into the system if they provide a correct set of credentials
        4. User must be able to take a quiz outlining their preferences for living conditions
        5. User must be able to modify the results of a previously taken quiz
        6. User must be able to view the results of a quiz, either directly after taking it OR viewing the results of a previously taken quiz.
        7. User must be able to access the raw data on cities as an index (City Index)

#### APPLICATION REQUIREMENTS

* + - * 1. Application must be able to store and modify user profile data
        2. Application must be able to modify user profile data
        3. Application must be able to store city data
        4. Application must be able to take user quiz input and establish what city they would be most likely to enjoy living in
        5. Application must be able to verify user credentials

### Overview of Data Requirements

There are three main types of data used on the Yuppie City Simulator platform, which interact with each other in various forms.

* User profile data
* User Quiz Data
* City Data

All data related to user profiles that is deemed non-confidential will be stored in a user profile database. This will include all data that is public on a user’s profile, which will include username, number of completed quizzes, top cities, etc. The user quiz data will be stored in a separate table with a corresponding user ID for each quiz. The quiz data will include a score from 1-10 for each category the user has chosen to answer. The city data will be stored in a third table, which will be accessed by the quiz algorithm to determine best fitting cities. The city data will also be accessed when the user opens the city index section of YCS..

### General Constraints, Assumptions, Dependencies, Guidelines

* + - 1. The system must be web-based. The user must have a working internet connection. If internet connection is temporarily interaction, system should maintain its state until connection is reestablished or session times out.
      2. The user’s information must be stored securely. Encryption method should include salting and hashing of user’s data. System should be able to survive a crash without loss of user data.
      3. System should be able to display correctly on multiple devices (desktop, laptop, phone/tablet, etc.).
      4. The system should begin responding to user’s input within one second. System should include a method of notification when the system is loading other pages so as to avoid the user’s assumption of a crash.

### User View of Product Use

Our website, yuppiecitysimulator.ru.biz, will be themed around nostalgia for 1990s web use. This means that it will be written primarily in HTML, and minimally use Cascading Style Sheets or other more fancy web development. On login, the user will be greeted with a simple home page that displays a list of links to each service, as follows: Profile Management, Take Quiz, Edit Quizzes, and City Index. Upon selecting Profile Management, the user can edit features of their account like username and password. Every page will contain a link back to the home page. The Take Quiz page will have a single box for each city feature. The user can type from 0-10 the importance they place on each feature, and submit the quiz when completed. Selecting edit quizzes will allow a user to click on a quiz and perform a similar functionality. Our City Index will be a simple table that shows each city as a row, each attribute as a column, and displays a value based on that particular city’s performance. Included below are examples of what data from a user quiz and the City Index might look like.

Sample Quiz Data:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| user\_id | quiz\_id | Walkability | Annual Sunshine | Population | Healthcare |
| 61 | 1425 | 10 | 7 | 0 | 8 |

Sample City Index:

|  |  |  |  |
| --- | --- | --- | --- |
| city\_id | Walkability | Annual Sunshine | Healthcare |
| Boston | 9 | 7 | 10 |
| New York City | 10 | 7 | 8 |
| Los Angeles | 4 | 10 | 7 |

## SPECIFIC REQUIREMENTS

### External Interface Requirements

The website must be able to display appropriately on various display sizes, resolutions, and aspect ratios. Each page of the website should have a menu that allows the user to navigate to the other main pages of the site. It must also support various devices (monitors, laptops, and smartphones. The website must be able to query the database to retrieve city data according to various attributes.

### Detailed Description of Functional Requirements

Class diagrams outlining the overall structure of our project, as we currently understand it, are available in the appendix section 5.2.1.

Similarly, sequence diagrams for each use case are provided in appendix section

#### Detailed User Requirements

* + - * 1. User Profile Creation

|  |  |
| --- | --- |
| Purpose | Enable the user to create a profile for the application to store and modify. See Appendix 5.2.1 for diagram |
| Inputs to Component | User designated:   * Username * Password * Email Address |
| Processing | System verification that username and email are NOT already in use. In the event that either are, throw an error the user and have them input a new set. |
| Outputs | A User profile object that will be handed to the application’s backend for storage. (3.2.2.1) |

* + - * 1. Modify User Profile

|  |  |
| --- | --- |
| Purpose | Enable the user to modify their existing profile data.  See Appendix 5.2.1 for diagram |
| Inputs to Component | If NOT logged in:   * Current username for login * Current password for login   Updated values for:   * Username (optional) * Password (optional) * Email Address (optional) |
| Processing | IF NOT LOGGED IN:  System follows login procedure outlined in 3.2.1.3  System verification that new username / email are NOT already in use. In the event that either are, throw an error the user and have them input a new set. |
| Outputs | Modified user profile that will be handed to the application’s backend for storage. (3.2.2.2) This new profile will override the existing one in the backend. |

* + - * 1. User Login

|  |  |
| --- | --- |
| Purpose | Enable the user to log into the application with an existing set of credentials and begin using the application with the linked user profile  See Appendix 5.2.1 for diagram |
| Inputs to Component | User given:   * Username * Password |
| Processing | System verification that the username / password combination does exist in the system. If the combination does NOT match any existing records, throw an error to the user and have them try again. |
| Outputs | Matching profile is activated and user is brought to the home screen. |

* + - * 1. User taking quiz

|  |  |
| --- | --- |
| Purpose | Enable the user to take the quiz that is the cornerstone of the YCS project.  See Appendix 5.2.2 for diagram |
| Inputs to Component | User designated:   * Responses to quiz |
| Processing | System validates the the quiz responses are valid (for free-response type questions. e.g type a value from 1-10). If not, system throws an error asking the user to re-input their response.  System then moves to process outlined in 3.2.2.4 |
| Outputs | User profile is updated with a new quiz-response entry and is handed to the application’s algorithm for use in classification and storage. See outline in 3.2.2.4  Results are then presented in accordance with 3.2.1.6 |

* + - * 1. User modifying existing quiz.

|  |  |
| --- | --- |
| Purpose | Enable the user to modify an existing set of quiz responses in order to reflect updated views on preferences.  See Appendix 5.2.2 for diagram |
| Inputs to Component | User designated:   * Quiz Selection * Updated Quiz Responses (Optional)   System pull (automatic):   * User Profile: Quiz result set |
| Processing | Once the user designates the quiz they’d like to modify, the system pulls up that response set to display. User can then update answers as they see fit.  System validates free response questions in the same way as it does for 3.2.1.4 and then moves to process outlined in 3.2.2.4 |
| Outputs | User profile is updated with the updated quiz-response entry and is handed to the application’s algorithm for use in classification and storage. See outline in 3.2.2.3 |

* + - * 1. User Viewing Results

|  |  |
| --- | --- |
| Purpose | Enable the user to see the results of their quiz(zes) at any time, provided they already have some existing.  See Appendix 5.2.2 for diagram |
| Inputs to Component | User designated:   * Quiz result selection   System pulls:   * User Profile: Quiz Results. |
| Processing | System verifies that the User profile DOES have at least one set of quiz results. Returns an error to the user if they do NOT.  System then pulls the results of the chosen quiz from the user profile for display.  Alternate: If coming from 3.2.1.4 (and subsequently 3.2.2.3), system will automatically pull results from the just-completed quiz. |
| Outputs | Displays the city that YCS has calculated would best fit the user’s response. |

* + - * 1. User Use of City Index

|  |  |
| --- | --- |
| Purpose | Enable the user to view an index of raw data on various aspects of city measurements.  See Appendix 5.2.2 for diagram |
| Inputs to Component | User designated:   * Categories for filtering |
| Processing | System queries application backend for data corresponding to user filter. |
| Outputs | Displays information secured from backend. |

#### Detailed Application Requirements

* + - * 1. Application Storage of User Profile Data

|  |  |
| --- | --- |
| Purpose | Enable the system to have a persistent database of user profiles for reference in various functions. |
| Inputs to Component | User profile consists of:   * Username * Password * Email * Quiz Responses Set * Quiz Results set |
| Processing | System stores User profile in the back end. |
| Outputs | System returns user profile when requested. Success message. |

* + - * 1. Application Modification of User Profile Data

|  |  |
| --- | --- |
| Purpose | Enable the system to update information in existing user profiles. |
| Inputs to Component | System receives updated user profile from 3.2.1.2 or 3.2.2.4 |
| Processing | System overrides original profile stored in backend with new profile data. |
| Outputs | System returns Success message. |

* + - * 1. Application Storage of City Data

|  |  |
| --- | --- |
| Purpose | Enable the system to locally store a persistent database of city data. |
| Inputs to Component | City Data (format currently TBD) |
| Processing | System stores City Data in the back end. |
| Outputs | System returns City Data when queried by 3.2.2.3.4 or 3.2.2.1.7 |

* + - * 1. Application Algorithm Execution

|  |  |
| --- | --- |
| Purpose | Enable the system to run calculations on user quiz responses in order to determine an ideal city to live in |
| Inputs to Component | User profile handed over from either sections 3.2.1.4 or 3.2.1.5  City Data from query to database, outlined in sec 3.2.2.3 |
| Processing | System runs algorithms to match the user responses to the available city data.  Modifies the User Profile to include a proper result from calculations. |
| Outputs | System returns results to user in accordance with sec 3.2.1.6  System stores updated user profile in accordance with section 3.2.2.2 |

* + - * 1. Application Credential Validation

|  |  |
| --- | --- |
| Purpose | Enable the System to verify that the user attempting to login is using correct credentials |
| Inputs to Component | User designated:   * Username * Password |
| Processing | System queries backend for any user profile (in accordance with sec 3.2.2.1) matching the given credentials. If NONE found, throws an error. |
| Outputs | If no credentials are found, error is thrown. Otherwise, success message is given. |

### Performance Requirements

System should respond to user’s input in less than 1 second.

System should be able to generate recommendations after survey completion in less than 30 seconds.

System should be able to support 1,000 simultaneous users.

System should be able to survive a crash without loss of user data.

### Quality Attributes

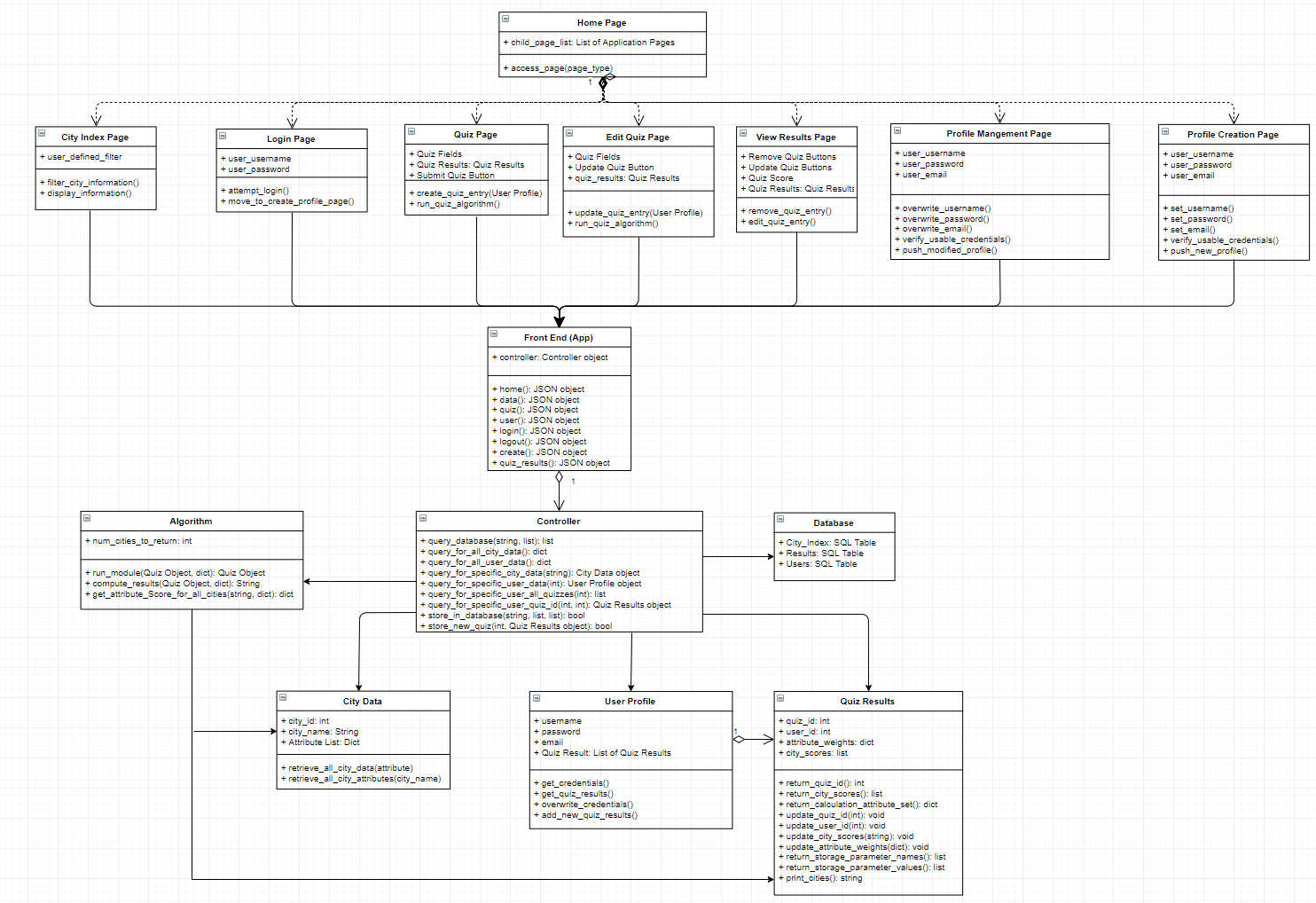
The user’s info should be stored securely.

User should be able to complete survey and get recommended cities without creating a profile, however their answers will not be saved unless they then complete a profile.

Upon the user’s input of invalid credentials, system should prompt user to re-enter credentials without needing to reload the entire page.

Website should not be down for more than 30 minutes per week.

* 1. Class Diagram



# System Design

## Major software requirements

The most crucial aspects of our software, and the ones that will allow us to achieve success at the most fundamental level, are the quiz and the city database. With these two elements, a user will be able to take a quiz that is reliant on the data from our database and receive a result based on their answers. Beyond this, a website that houses our quiz and stores past results on a per user basis will allow our software to provide a more customizable user experience.

### Design constraints, limitations

In our project’s current state, our main limitation is time. In order to implement many of the features of our project, we will need time to develop a programming strategy and test out the features we are implementing. If we have any hitches in the development of our software, it could result in us being unable to implement major features. Beyond this, our project requires a few elements be present in order for others to be built around them, and until those elements, such as the database, are completed it will be difficult to finish any work on other elements.

Additionally, our group is specialized towards specific areas of development, which means that if one of us has an issue with our particular section of work it may be difficult to assist each other. Our team is not web development focused, which means we will be learning as we go in regards to the creation of our project’s website.

### Design Goals

Reliability: The YCS system should be reliable in uptime and general availability to any client [generalization of Non-Functional Requirements 3,8 (NFR-3,8)]

Fault Tolerant: The YCS system should be fault tolerant to the loss of connectivity without corruption of data [generalization of NFR-4]

Response Time: The YCS system must be able to act upon user input, including generation of quiz results, in reasonable amounts of time [generalization of NRF-1,2]

Security: The YCS system must securely store user information to prevent unauthorized access [generalization of NFR-5]

### Changes to requirements

Due to the nature of us not completing any new coding developments since the previous requirements documents, we have not elected to make any changes to our requirements. However, we have considered the importance of bagels, which is now included in our sources.

### Overview of Document

The rest of our SDS will follow the format that has been given. If it strays in any way, it is a result of errors from Meandering Armadillos personnel exclusively.

## Data Design

### Data Objects and resultant data structures:

User Profile:

User Name: String

Password: String

Email: String

Quiz Results: Quiz Results

City Data:

City Name: String

Attributes: Map <String, Int>

Quiz Results:

Answers: Map <String, Int>

Result Cities : List <String, Int, City Data>

### File and Database Structures

* + 1. **External File Structure**

Our file structure will consist of our website’s homepage and corresponding pages including the quiz and user profile page. A separate python file will house the algorithm that runs our quiz and calculates the user’s results.

* + 1. **Global Data**

Our database is our only “global data” as it is accessed by both our python algorithm and our website. This is housed on a MySQL server hosted by AWS.

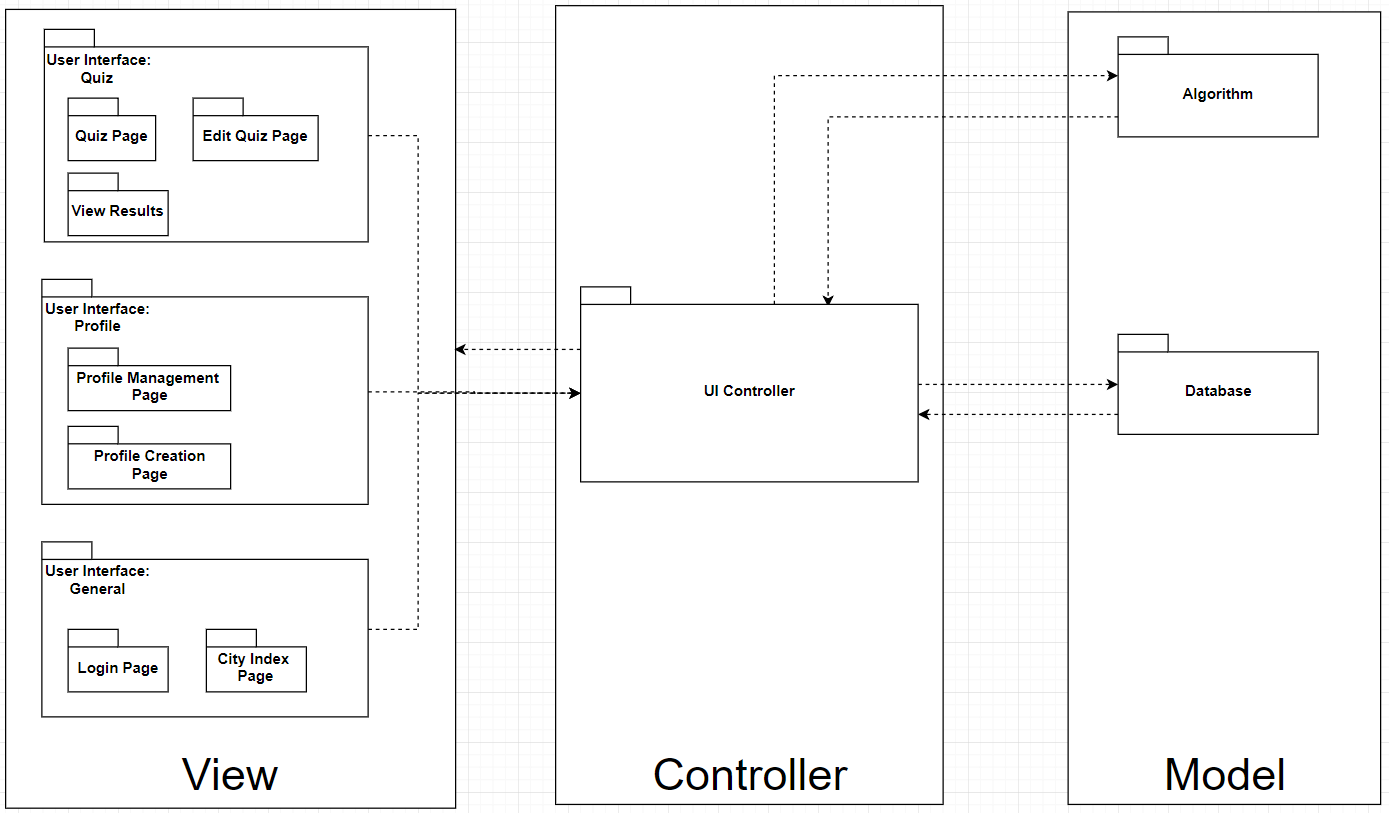
### **File and Data Cross Reference** N/A

## System Architecture Description

### Overview of Modules / Components:

1. **User Interface:** Composed of all pages accessible to the user. Provides user with the means to interact with the Yuppy City Simulator. Broken down into three main sub-components:
   1. Quiz: Contains all pages necessary for the user to execute a quiz workflow or modification to an existing quiz.
   2. User Profile: Contains all pages necessary to create or modify a user profile.
   3. General: General / miscellaneous pages, including the login, and general city data pages.
2. **UI Controller:** Module that handles the processing of user information into the UI, including database querying and algorithm processing.
3. **Algorithm:** Module that allows the data gathered from the UI to be processed and a result determined.
4. **Database:** The repository for all persistent information in the system, including user profiles, quiz results and city data.

### Structure and relationships:



## Detailed description of components

### **Algorithm**

* **ID:** Algorithm
* **Type:** Python Script
* **Purpose:** To create a list of ideal cities for a user to live in based on their quiz results. Should take no longer than 30 seconds as specified by SRS section 3.3. If a user is logged in, it will save the results to their profile for later viewing, otherwise results can still be generated but no persistent results will be created, as specified by SRS section 3.4
* **Function:** The inputs for the Algorithm are:
  1. Quiz Results: as specified in section 2
     + Pulled either from the User Profile, if user is logged in, or directly handed to the algorithm if the user is NOT logged in.
  2. City Data: as specified in section 2

Outputs:). Raw Quiz Results (if user not logged in).

Algorithm will use a weighted technique, where scores are based on user designated weights of certain attributes and matched against a city’s score in that attribute. User input only requires a Quiz Results object and will return a copy of the object with the resulting cities assigned. All city data is pulled straight from the database in an initial query and is not technically an input.

* **Subordinates / Modules used:** Quiz Results, City Data interfaces
* **Dependencies:** The Algorithm will rely on the Database for pulling the city data necessary to compare user results with prefered living conditions.
* **Resources:** CPU execution time, python math library
* **Processing:**  run\_module(){

Compute\_results()  
 Return current\_quiz

}

Compute\_results(Quiz\_results){

for all attributes in Quiz Results{

Get scores for all cities in attributes listed in Quiz\_results.

Go through all scores, compute new total city score by weight \* city attribute score

}

Return top x number of cities by total score

}

* **Data**: Algorithm does not keep any initial data values. Will all be either handed to the algorithm or queried fresh from the database for each run.

### SQL Database

* **ID:** SQL\_DB
* **Type:** MySQL Database
* **Purpose:** To house all city and user data for utilization by our website and algorithm. The user’s info should be stored securely.
* **Function:** Data is held in individual tables for city data and user data.
* **Subordinates:** N/A
* **Dependencies:** N/A
* **Resources:** AWS Server running MySQL
* **Processing:** Calls will be made retrieving table data on the basis of quiz specific and user specific needs. The entire table will also be retrieved for our City Index page.
* **Data:** Each table will contain rows that define a city or a user, respectively.
  + City\_Index
    - City\_ID: int, not null, auto-increment
    - City\_Name: varchar
    - Walkability, Transit, Population, Population Density, etc: double
  + Users
    - User\_ID: int, not null, auto-increment
    - User\_name: varchar
    - Password: varchar
    - E-mail: varchar

### User Interface: Quiz

* **ID:** Quiz
* **Type:** Web Form
* **Purpose:** To gather the necessary information from the user in order to calculate unique recommendations.
* **Function:** The user will fill out each field of the form. The quiz page will then ask the user to save their answers and then redirect them to the results page.
* **Subordinates:** Quiz results
* **Dependencies:** The quiz will depend on the algorithm to generate recommendations, as well as the city database so that the city data of each recommendation can be presented to the user. It will also rely on the UI controller module to handle user input.
* **Resources:** N/A
* **Processing:** When the form is submitted, the quiz will pass the user’s answers to the algorithm in order to generate city recommendations. It will then wait for a response from the algorithm, and upon receiving one, will display the results to the user.
* **Data:** The user’s answers will be stored in JSON format and passed to the algorithm.

### User Interface: User Profile

* **ID:** User\_Profile
* **Type:** Web Page
* **Purpose:** To provide the user a method of viewing and editing their account information.
* **Function:** The page will initially display the user’s information as read-only. The user can then selectively edit each section and then save their changes.
* **Subordinates:** User Profile Page
* **Dependencies:** The user profile page will rely on the user database in order to retrieve and update the user’s information. It will also rely on the UI controller module to handle user input.
* **Resources:** N/A
* **Processing:** Each piece of user information will initially be read-only. The password will appear as bullets or asterisks. Upon clicking the edit button next to each piece of information, the user will be able to edit that information, and then save their changes.
* **Data:** The only data is the username, password, and email address of the user.

### User Interface: General

* **ID:** UI
* **Type:** Web page
* **Purpose:** To provide an interface for the user to view and interact with the system**.**
* **Function:** The general functionality of the interface will be to process the user input and perform the appropriate action in response.
* **Subordinates:**
* **Dependencies:** Dependencies will vary from page to page, but most pages will at least depend on the database being functional. As always, it will also rely on the UI controller module to handle user input.
* **Resources:** All pages will make use of JQuery, as well as other page-specific libraries.
* **Processing:** Each page is only utilized when it is the current active page. From every page the user can access the sidebar menu and navigate to the other main pages of the site.

### UI Controller:

* **ID:** Controller
* **Type:** Python Scripts
* **Purpose:** The purpose of the UI controller is to handle communication and user input between the UI layer and the model (database / algorithm) layer.
* **Function:** Specifically, this layer handles database queries to retrieve user, quiz, or city data information along with any calls necessary to the algorithm module. In addition, it controls the displaying and updating of data to the UI. It does NOT process anything itself but merely facilitates communication, and queries, returning found objects and errors as necessary.
* **Subordinates:** The controller is comprised of two main files: controller.py and app.py. The controller class itself handles most of the functions listed in first portion (calculations) of the functions section while app.py deals with latter (web interfacing) portions of it.
* **Dependencies:** As stated previously, the User Interface itself (all modules of it**)** rely on this controller to interface with the model layer. Any user call from the UI that requires data pulled or calculations made will trigger Controller behavior and give an appropriate counter-response. In addition, the UI controller is dependent on the actual Algorithm module to generate calculations.
* **Resources:** CPU Execution time.
* **Processing:** Database querying.Algorithm calls, UI updating.
* **Data:** Utilizes the object: User Profile, Quiz Results, City Data

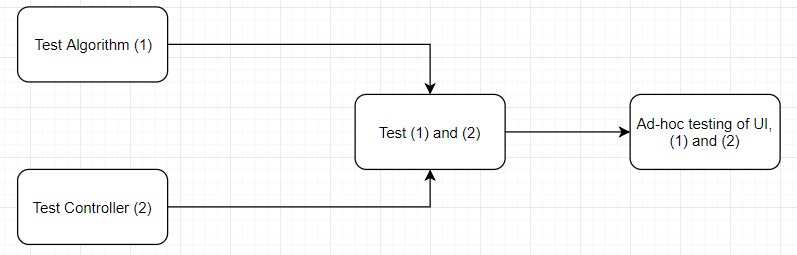
### 

### Interface Design:

Home Page:  
This page will serve as the main hub for the user to access the other pages on the site. It will also display basic information about how to use the service.  
  
Quiz Page:  
This page will serve as the main method for the user to not only fill out a quiz, but also modify the answers to a previously completed quiz. When modifying an existing quiz, the questions the user has already answered will appear with their answers already filled in.  
  
Results Page:  
This page will display the results of a selected completed quiz. The results will consist of the city that scored the highest according to the user’s answers, followed by a list of the 4 other cities receiving high scores.  
  
City Data Page:  
This page will serve as the main method for the user to view and interact with the data about each city in the YCS database. The user will be able to sort the data according to a chosen attribute. They will also be able to filter the data based on certain constraints they choose for certain attributes.  
  
User Profile Page:  
This page serves as the main method for a user to view and edit their profile info (username, password, and email).  
  
Sidebar Menu:  
The sidebar menu will serve as an easy way for the user to navigate to other pages of the site without having to return to the homepage. The menu will normally be collapsed to the side, however when the user click on the hamburger icon in the top corner, the menu will expand and allow them to select a page to navigate to.

# Test Document

Our testing plan followed the bottom up approach for integration testing and focused on three main aspects: Our controller, our algorithm, and our database. We had no formal process for testing the UI as mostly just called controller functions, owning to a lack of time to fully flesh out other features. As a result, we employed an ad-hoc approach of data entry and database verification for full integration testing.



## Algorithm Test Cases

|  |  |
| --- | --- |
| Test Case ID | 1 |
| Test Case Name | Test Calculation |
| Description | This test ensures that the Algorithm can successfully use the provided inputs to calculate a predetermined output |
| Prerequisites | N/A |
| Inputs | 1. Mock Quiz Results Object:    * Walkability: 1    * Sunshine: 10    * Tech Jobs: 5 2. Mock City Data Object List:    * [Los Angeles - Walkability: 3, Sunshine: 10, Tech Jobs: 6]    * [San Francisco - Walkability: 4, Sunshine: 7, Tech Jobs: 10]    * [Boston - Walkability: 10, Sunshine: 5, Tech Jobs: 6] |
| Expected Results | String: City / Scores for all 3 cities  “Los Angeles:133, San Francisco:124, Boston:90” |
| Associated Test Cases | N/A |

## Controller Test Cases

|  |  |
| --- | --- |
| Test Case ID | 2 |
| Test Case Name | Test Quiz Insert |
| Description | Test ensures that a Quiz Object can be successfully entered into the database |
| Prerequisites | N/A |
| Inputs | Mock Quiz Results Object:   * User\_id: 99999 * Attribute\_weights:   + Walkability: 1   + Bikeability: 2   + Transit: 3   + Traffic: 4   + Metro Pop: 5   + Pop Density: 6   + Prop Crime: 7   + Violent Crime: 8   + Air Pollution: 9   + Sunshine: 10 * City\_Scores: [“Testing String: 0”] |
| Expected Results | NONE: No error upon insertion |
| Associated Test Cases | N/A |

|  |  |
| --- | --- |
| Test Case ID | 3 |
| Test Case Name | Test Query for Specific City Data. |
| Description | Test ensures that the controller can query for and receive all information on a given city from the Database |
| Prerequisites | Database has “Los Angeles” entered in the data |
| Inputs | String: “Los Angeles” |
| Expected Results | Los Angeles City Data Object |
| Associated Test Cases | N/A |

|  |  |
| --- | --- |
| Test Case ID | 4 |
| Test Case Name | Test Query For All City Data |
| Description | Test ensures that it is possible to query for and secure every single city’s data from the database. |
| Prerequisites | Database has 25 cities in it |
| Inputs | NONE |
| Expected Results | List of City Data objects of length 25 |
| Associated Test Cases | Test Case 3 |

|  |  |
| --- | --- |
| Test Case ID | 5 |
| Test Case Name | Test Query for Specific User |
| Description | Test ensures that it is possible to query for and secure the data of a specific user |
| Prerequisites | Database has a user with user\_id 9999 |
| Inputs | Int: user\_id = 9999 |
| Expected Results | User Profile Object for user corresponding to user\_id 9999 |
| Associated Test Cases | N/A |

## Integration Testing

|  |  |
| --- | --- |
| Test Case ID | 6 |
| Test Case Name | Test Full Workflow |
| Description | This test ensures that the controller can work with the algorithm and database to produce a completed Quiz Result object, store it in the database and return a result string to the UI. |
| Prerequisites | Database Contains Standard set of City Data (see Appendix) |
| Inputs | 1. Dict containing [Attribute: Value] Key pairings    * Walkability: 1    * Bikeability: 2    * Transit: 3    * Traffic: 4    * Metro Pop: 5    * Pop Density: 6    * Prop Crime: 7    * Violent Crime: 8    * Air Pollution: 9    * Sunshine: 10 |
| Expected Results | 1. Dict containing [City Name: City Score] Key pairings:    * New York: 3921    * Boston: 3346    * San Diego: 3152    * Tampa: 3136    * Miami: 3074 2. List of Dicts containing [City Attribute Name: City Attribute Value] Key pairings:    * A tad too long to list. |
| Associated Test Cases | Test Cases 1, 2, 3 |

# User Manual / User Guide

Required Software:

1. Python 3.6
2. Python modules: flask, flask-mysql, pymysql
3. MySQL:<https://dev.mysql.com/doc/refman/8.0/en/windows-installation.html>

Quiz Instructions

To take a quiz, prospective yuppies should navigate to the quiz section of the YCS website.

1. Entering values

When a yuppie wishes to provide their preference on a particular city aspect, they can input a value from -10 to 10 in each of the boxes on the quiz form. If they wish to skip a particular entry, they may enter 0 to contribute no weighting to an aspect.

2.) Quiz score

When a yuppie wishes to receive their score, they may press the “View Results” button on the quiz form. This will run our algorithm and display a list of the top 5 most fitting cities. Each city will be displayed with a weighted score and its index values.

# Glossary

AWS - Amazon Web Services, cloud hosting platform

MySQL - A variation of the structured query language (SQL) of databases

YCS - Yuppie City Simulator, the technology we have birthed into this world

# References