|  |
| --- |
| **UMGC CITY TEAM 1** |
| **SOFTWARE DESIGN DOCUMENT** |
|  |
| **Christy Gilliland**  **Krystina Poling**  **Daniel Abresch**  **Ziad Elharaoui**  **Jack Amnuaysirikul**  **Melanie Meek**  **Tarig Abasit**  **Patience Okereke**  **3-20-2020** |

Contents

[1. INTRODUCTION 4](#_Toc35645613)

[1.1 Purpose 4](#_Toc35645614)

[1.2 Scope 4](#_Toc35645615)

[1.3 Overview 5](#_Toc35645616)

[1.4 Reference Material 6](#_Toc35645617)

[1.5 Definitions and Acronyms 6](#_Toc35645618)

[2. SYSTEM OVERVIEW 7](#_Toc35645619)

[3. SYSTEM ARCHITECTURE 8](#_Toc35645620)

[3.1 Architectural Design 8](#_Toc35645621)

[3.2 Decomposition Description 9](#_Toc35645622)

[3.3 Design Rationale 10](#_Toc35645623)

[4. DATA DESIGN 11](#_Toc35645624)

[4.1 Data Description 11](#_Toc35645625)

[4.2 Data Dictionary 14](#_Toc35645626)

[5. COMPONENT DESIGN 15](#_Toc35645627)

[5.1 Presentation Layer 15](#_Toc35645628)

[5.2 Service Layer 17](#_Toc35645629)

[5.3 REST Controller 17](#_Toc35645630)

[5.4 Service Component 18](#_Toc35645631)

[5.5 DATA ACCESS LAYER 19](#_Toc35645632)

[6. HUMAN INTERFACE DESIGN 20](#_Toc35645633)

[6.1 Overview of User Interface 20](#_Toc35645634)

[6.1.1 Home Page Component 21](#_Toc35645635)

[6.1.2 Login Form Component 22](#_Toc35645636)

[6.1.3 Sign-Up Form Component 23](#_Toc35645637)

[6.1.4 Credential Recovery Help Component 24](#_Toc35645638)

[6.1.5 Landing Portal Component 25](#_Toc35645639)

[6.1.6 Manual Use Case Component 26](#_Toc35645640)

[6.1.7 Auto Upload Use Cases Component 27](#_Toc35645641)

[6.1.8 Manage Existing Use Cases Component 28](#_Toc35645642)

[6.1.9 Edit Use Case Component 29](#_Toc35645643)

[6.1.10 Delete Use Case Component 30](#_Toc35645644)

[6.1.11 Export Project Component 31](#_Toc35645645)

[6.1.12 Generate Output Component 32](#_Toc35645646)

[6.1.13 Display Preview Component 33](#_Toc35645647)

[6.1.14 Download Project Component 34](#_Toc35645648)

[6.1.15 Interactive Map Component 35](#_Toc35645649)

[6.1.16 Quick Reference Page 36](#_Toc35645650)

[7. REQUIREMENT MATRIX 37](#_Toc35645651)

[Appendix A 44](#_Toc35645652)

Table of Figures

[Figure 1 - High Level Architectural Overview of the UMGC City Application 9](#_Toc35645653)

[Figure 2 -Entity Relationship Diagram for the UMGC City Database 13](#_Toc35645654)

[Figure 3 - Component Process Flow 16](#_Toc35645655)

[Figure 4 - Data Flow Diagram: UMGC City Zoning Application (Management User) 19](#_Toc35645656)

[Figure 5 - Data Flow Diagram: UMGC City Zoning Application (Resident User) 20](#_Toc35645657)

[Figure 6 - Wireframe: Home Page Component 22](#_Toc35645658)

[Figure 7 - Wireframe: Login Form Component 23](#_Toc35645659)

[Figure 8 - Wireframe: Sign-Up Form Component 24](#_Toc35645660)

[Figure 9 - Wireframe: Credential Recovery Help Component 25](#_Toc35645661)

[Figure 10 - Wireframe: Landing Page Component 26](#_Toc35645662)

[Figure 11 - Wireframe: Manual Use Case Component 27](#_Toc35645663)

[Figure 12 - Wireframe: Auto Upload Use Cases Component 28](#_Toc35645664)

[Figure 13 - Wireframe: Mange Existing Use Case Component 29](#_Toc35645665)

[Figure 14 - Wireframe: Edit Use Case Component 30](#_Toc35645666)

[Figure 15 - Wireframe: Delete Use Case Component 31](#_Toc35645667)

[Figure 16 - Wireframe: Export Project Component 32](#_Toc35645668)

[Figure 17 - Wireframe: Generate Output Component 33](#_Toc35645669)

[Figure 18 - Wireframe: Display Preview Wireframe 34](#_Toc35645670)

[Figure 19 - Wireframe: Download Project Component 35](#_Toc35645671)

[Figure 20 - Interactive Map Component 36](#_Toc35645672)

[Figure 21 - Quick Reference Page 36](#_Toc35645673)

Table of Tables

[Table 1 - Definitions and Acronyms 5](#_Toc35645674)

[Table 2 - Data Dictionary 13](#_Toc35645675)

[Table 3 - Requirements Matrix 37](#_Toc35645676)

# INTRODUCTION

The following subsections provide a general description of this document, formally known as the Software Design Document (SDD). These subsections present a top-level architecture and design approach of the UMGC City Application. Some of the information here was excerpted from the project plan approved by the stakeholders on February 22, 2020.

## Purpose

This SDD describes the architecture and system design of the UMGC City Application (Version 1.0). Its main goal is to adequately describe “how” the UMGC City Application shall be built by explaining the core infrastructure along with all its system components.

## Scope

The scope of the UMGC City project is to build a free, open-source application that can be used by city officials to improve the usability of existing city web portals. The objective of the application is to enhance the user experience by helping them easily and intuitively locate applicable city ordinances for a predetermined list of frequently requested user inquiries (as determined by the city officials). In order to accomplish this goal, the application shall present the city officials with an interface to accept input of various use cases along with all the pertinent information that will benefit their intended audience. The application shall process that data into appropriate tables that comprise a database. When the users visit the city web portal to search for specific ordinances, the database shall be called upon to produce appropriate output that is tailored to them in a user-friendly format. This generic, build-to-suit database infrastructure allows the application to be reused by any city.

The UMGC City teams, along with the DevOps team, will work closely with the customer, Israel Del Toro, to design the database. In addition, the development teams shall tailor the application to deliver new pages and functionalities that are customized for the website of the City of Pasadena, California. Specifically, the UMGC City project teams will be delivering two interfaces: an interactive map/web-based interface (Team 1) and a ChatBot interface (Team 2). This SDD document presents all system requirements applicable to the database setup and the map/web-based interface, whose tasks fall under UMGC City Team 1. UMGC City Team 2 will use the same database designed by Team 1 to ensure the same data is used for both interfaces.

## Overview

This SDD is organized into six main sections as listed below. They constitute the overall software design approach for the map/web-based modules of UMGC City Application:

**Section 1. Introduction**

Describes and defines the overall purpose, scope, and approach of the Software Design Document.

**Section 2. System Overview**

Provides a general description of UMGC City Application’s functionality, context, and design choices.

**Section 3. System Architecture**

Explores the modular program structure within UMGC City Application, breaks down the structure into its decompositions, and discusses the rationale behind the selected architectural design.

**Section 4. Data Design**

Explains how information used in the system is transformed into appropriate data structures, including the main database. In addition, the system’s data entities shall be listed with their name, attributes, and each attribute’s type.

**Section 5. Component Design**

Examines how each component contributes to the system in order to serve up the functionalities of UMGC City Application.

**Section 6. Human Interface Design**

Displays how each user interface (UI) functionality of the system is designed to deliver the content that is useful to the user.

**Section 7. Requirements Matrix**

Links system components and data structures to the functional requirements provided in the Software Requirements Specification (SRS).

## Reference Material

* SDD Template: <https://sovannarith.files.wordpress.com/2012/07/sdd_template.pdf>
* Assignment Document: https://learn.umuc.edu/d2l/le/content/444089/viewContent/16973490/View

## Definitions and Acronyms

Table 1 - Definitions and Acronyms

|  |  |
| --- | --- |
| **ChatBot** | a computer program designed to simulate conversation with human users |
| **CRUD** | “Create, Read, Update and Delete” |
| **DevOps** | Development and Operations Team |
| **DFD** | Data flow Diagram |
| **ERD** | Entity Relationship Diagram |
| **GUI** | Graphical User Interface |
| **HTTP Request** | The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server. |
| **PostgreSQL** | A general-purpose and object-relational database management system. |
| **REST Controller** | This is used to create RESTful web services using Spring MVC. It takes care of mapping request data to the defined request handler method |
| **REST API** | an application program interface ([API](https://searchapparchitecture.techtarget.com/definition/application-program-interface-API)) that uses HTTP requests to get, put, post and delete data |
| **SDD** | Software Design Description |
| **Spring Data JPA** | Part of the larger Spring Data family. This makes it easy to implement JPA based repositories |
| **UMGC** | University of Maryland Global Campus |
| **PaaS** | Platform as a Service - a cloud computing model in which a third-party provider supplies the software and hardware infrastructure required to develop, run, and manage an application. |

# SYSTEM OVERVIEW

The UMGC City Application is a collection of interfaces designed to assist city officials in creating a database and a customer-facing HTML interface. The customer-facing HTML interface is created from a pre-configured HTML template and data collected from city

management users which is stored in the application’s database. The result of the application’s deliverable is a static HTML page in an easy-to-navigate format for use by city residents. The UMGC City Application is intended to integrate with current websites employed by the cities. However, it does not imply suitability to wholly replace current system. This SDD shall serve as a technical design blueprint for the development of the UMGC City Application.

# SYSTEM ARCHITECTURE

The following subsections give the descriptions of how the UMGC City Application shall be built, including a high-level overview of the whole architectural design, its subsystems, and the rationale behind the chosen components.

## Architectural Design

The UMGC City Application is built and supported by a microservices architecture. The entire application is composed of small independent services that communicate using web APIs. The UMGC City’s microservices architecture is completely hosted within the Heroku (PaaS) cloud platform using isolated, virtualized Linux containers named “dynos”. In addition, the SQL database that the UMGC City microservices rely on is also hosted on the Heroku cloud platform, which is called Heroku Postgres.

The overall high-level structure of the UMGC City Application can be broken down into three primary modules spread out across the system’s two microservices. The three main modules of the system include a presentation layer, supplied by the micro-frontend Angular based service, a service layer, and a data access layer, both of which are supplied by the micro-backend Java based service. See Figure 1 below for a visual representation of the UMGC Architectural design at a high-level.

A screenshot of a cell phone

Description automatically generated

Figure 1 - High Level Architectural Overview of the UMGC City Application

## Decomposition Description

Each of system’s three identified modules from section 3.1 (the presentation layer, the service layer, and the data access layer) has a specific role to play to ensure that the UMGC City Application functions properly and meets the requirements as agreed to be all stakeholders. Each system module and its associated role is as follows:

* Presentation Layer: Provides the users of the system with a graphical user interface that is accessible through a web browser and displays content and information in a way that makes it useful to the user.
* Service Layer: Contains the functional business logic which drives the applications core capabilities such as authorization and authentication, data validation and processing, and all business transactions that come in or go out of the system.
* Data Access Layer: Maps the relational database to and from the system in-memory Java objects.

## Design Rationale

The UMGC City Application is designed with a microservices architecture, instead of a monolithic architecture so that each process is not tightly coupled to another process. In a monolithic architecture, all processes are tightly coupled and run as a single service which adds risk to the application’s availability. In a system where all the processes are dependent on one another, the chances of the system crashing due to a single process failure increases greatly. In addition, having tightly coupled processes means that when one process experiences an increase in traffic, the entire architecture must be allocated more resources, and adding any additional features or improvements to the system increases the complexity of the implementation.

The UMGC City Application’s microservice architecture allows each component of the application to be developed, deployed, operated, and scaled without affecting the functioning of the entire system. The microservice architecture that the UMGC City Application was designed with increases the system’s resilience, allows each microservice to be independently scaled to meet increased system demands, and enables continuous integration and delivery. The numerous advantages provided to the UMUC City Application by choosing this architecture is largely contributed to the fact that the entire system is decomposed into small modular components. This permits each functional component to be implemented independently without depending on the work of another team member. Furthermore, decomposing the system into modules aids in empowering each team member to work more independently, faster, and feel comfortable to take ownership over an individual module, seeing as the tasks become smaller and their context becomes better understood.

# DATA DESIGN

The following subsections explain how the information is organized into data structures within the UMGC City Application. Entities along with their descriptions and attributes shall be provided to show the design of system data.

## Data Description

The UMGC City Application’s data will be stored in a PostgreSQL database server stored on the Heroku cloud platform. The data model consists of 8 entities. The entities and their attributes are explained below:

* City\_User: This entity stores the details of the city official that manages the city. It includes the last name, first name, email address, and password attributes. The email address and password are required entities that will be used to authenticate users that log in to the system’s city administration portal provided by the front-end Angular UI.
* City: This entity stores the details of the city that will be included in the zoning application, i.e. the city’s name and the state where the city is located. This entity references the City\_User entity.
* Zone\_Land\_Use: This entity stores all land uses applicable for a city by their description. This entity references the city entity.
* Zone: This entity stores all available zone types within a city by their zoning symbol and naming description. This entity references the city entity.
* Allowed\_Land\_Use: This entity links the Zone\_Land\_Use entity to its associated Zone entity. It acts as a bridge entity for the many-to-many relationship between Zone\_Land\_Use and Zone and stores a record for each of the combinations of these other two entities.
* Development\_Standards: This entity stores the URL for each list of development standards that must be followed in each zone for a city. It references the Zone entity.
* Permit\_Type: This entity includes the permit types that must be obtained for an allowed land use in each individual zone belonging to a city. It references the Allowed\_Land\_Use entity.
* Application: This entity stores the link to each application that must be submitted for each individual permit type to be obtained. It references the Permit\_Type entity.

Each entity will include an id attribute that will be the entity’s primary key. The type of the primary key used in each entity is a Universally Unique Identifier (UUID), also known as GUID (global unique identifiers) to most software companies and used by software developers to refer to Component Object Module objects. For the UMGC City Application the PostgreSQL uuid\_generate\_v4() function will be used to generate each entity record’s UUID primary key value. The ERD diagram provided below in Figure 2 provides the architectural layout for the system’s PostgreSQL relational database management system and illustrates the relationship between each entity.

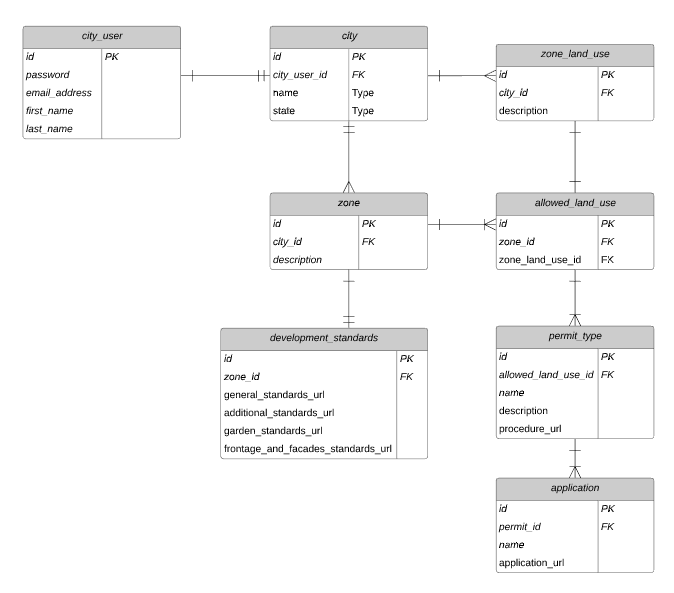


Figure 2 -Entity Relationship Diagram for the UMGC City Database

## Data Dictionary

Table 2 - Data Dictionary

| Entity | Field | Type | NULL | Default |
| --- | --- | --- | --- | --- |
| City\_User | Id | Primary Key: uuid | No | uuid |
|  | last\_name | varchar(30) | Yes | uuid |
|  | first\_name | varchar(50) | Yes |  |
|  | email\_address | varchar(50) | No |  |
|  | password | varchar(100) | No |  |
|  |  |  |  |  |
| City | Id | Primary Key: uuid | No | uuid |
|  | city\_user\_id | Foreign Key: uuid | No | uuid |
|  | name | varchar(50) | No |  |
|  | state | varchar(50) | No |  |
|  |  |  |  |  |
| Zone\_Land\_Use | Id | Primary Key: uuid | No | uuid |
|  | city\_id | Foreign Key: uuid | No | uuid |
|  | description | varchar(1000) | No |  |
|  |  |  |  |  |
| Zone | Id | Primary Key: uuid | No | uuid |
|  | city\_id | Foreign Key: uuid | No | uuid |
|  | description | varchar(1000) | No |  |
|  | zone\_symbol | varchar(5) | No |  |
|  |  |  |  |  |
| Development\_Standards | Id | Primary Key: uuid | No | uuid |
|  | zone\_id | Foreign Key: uuid | No | uuid |
|  | additional\_standard\_url | varchar(2000) | No |  |
|  | garden\_standard\_url | varchar(2000) | No |  |
|  | frontage\_and\_facades\_standards\_url | varchar(2000) | No |  |
|  | general\_standard\_url | varchar(2000) | No |  |
|  |  |  |  |  |
| Allowed\_Land\_Use | id | Primary Key: uuid | No | uuid |
|  | zone\_id | Foreign Key: uuid | No | uuid |
|  | zone\_land\_use\_id | Foreign Key: uuid | No | uuid |
|  |  |  |  |  |
| Permit\_Type | id | Primary Key: uuid | No | uuid |
|  | allowed\_land\_use\_id | Foreign Key: uuid | No | uuid |
|  | name | varchar(100) | No |  |
|  | description | varchar(1000) | No |  |
|  | procedure\_url | varchar(2000) | No |  |
|  |  |  |  |  |
| Application | id | Primary Key: uuid | No | uuid |
|  | permit\_id | Foreign Key: uuid | No | uuid |
|  | application\_url | varchar(2000) | No |  |
|  | name | varchar(50) | No |  |

# COMPONENT DESIGN

## Presentation Layer

The presentation layer of the UMGC City Application is handled by the front-end Angular microservice and is built around a view specific model that only handles the presentation logic, but no business logic. The Angular microservice is designed to be MVC-capable and therefore is broken down into three separate layers:

* The View Layer: There are a total of fifteen main components contained within the view layer. Each component has its own HTML template and Typescript file (also known as the component’s metadata). The metadata file is highly important to each component because it Angular where to the information it needs to create and present the component and its view.
* The Controller Layer: Each of the fifteen components contained within the view layer will also have one or more controllers attached to them acting as an object constructor. Controllers are JavaScript object that are used within the application to respond to user input and perform interactions on the data model objects which are used to manage the data passing through the front-end service.
* The Services Layer: The front-end service will have one service class that interacts with the controllers attached to each of the view layers components to enable data from being passed from the front-end microservice to the back-end service and vice versa.

Each component contained within the UMGC City Application’s presentation layer is illustrated in Figure 3 below, and the process flow for which each user will interact with each component, and how they interact with each other is also depicted.

A picture containing text, map

Description automatically generated

Figure 3 - Component Process Flow

## Service Layer

The service layer of the UMGC City Application resides within the back-end Java microservice and is designed with a Model-View-Controller (MVC) architecture, typically used for building RESTful CRUD web applications in Java. The service module is designed with one REST API controller class, one business logic service class, and eight data model classes.

## REST Controller

The system’s ZoningProjectContoller class is used to define which service entry point corresponds to a given HTTP URL, and how parameters are to be read from the HTTP request. The REST Controller is designed with the following API endpoints:

| **Method** | **Endpoint** | **Definition** |
| --- | --- | --- |
| POST | /city-zoning-project-management/users | Creates new user account |
| POST | / city-zoning-project-management/usecases | Creates new use case |
| POST | / city-zoning-project-management/usecases/upload | Creates new use cases from an upload csv file |
| DELETE | / city-zoning-project-management/usecases/{id } | Deletes use case |
| PUT | / city-zoning-project-management/usecases/{id} | Edits use case |
| GET | / city-zoning-project-management/cities/{id}/usecases | Retrieves all use cases belonging to a city |
| GET | / city-zoning-project-management/cities/{id}/usecases/{id} | Retrieves a single use case belonging to a city |
| GET | / city-zoning-project-management/cities/{id}/zones | Retrieves all zones belonging to a city |
| GET | / city-zoning-project-management/cities/{id}/landuses | Retrieves all available land uses for a city |
| GET | / city-zoning-project-management/cities/{id}/zone/{id}/landuses/{id} | Retrieves all allowed land use for a city zone |
| GET | / city-zoning-project-management/cities/{id}/zones/{id}/landuses/{id}/development | Retrieves the development standards link for an allowed land use in a city zone |
| GET | / city-zoning-project-management/cities/{id}/zones/{id}/landuses/{id}/permit | Retrieves the permit regulations link for an allowed land use in a city zone |
| GET | / city-zoning-project-management/cities/{id}/permits/{id}/application | Retrieves the application link for a city’s land use zoning permit |
| GET | / city-zoning-project-management/users | Requests authorization into the system |
| GET | / city-zoning-project-management/users/{name} | Retrieves a user’s email address |
| GET | / city-zoning-project-management/users/{email} | Requests to send forgotten password to user email address |

## Service Component

The service class containing all the business logic for the UMGC City Application and is implemented by using an interface class called ProjectZoningService and then implemented using the public class called ProjectZoningServiceImp. The service class is designed to use a Project Service API to store, retrieve, update and delete the data from the system’s database.

The service class processes incoming data request models, more specifically JSON object being sent over HTTP from the front-end and converts them into data objects that can be persisted into the system’s database. The eight persistent data models used by the ProjectZoningService are structured to match each of the databases entities.

## DATA ACCESS LAYER

The data access layer of the UMGC City application resides within the back-end Java microservice right under the system’s service layer. The data access layer is designed using Spring Data JPA and by using one of the JPA Repository interfaces made available through the Spring Data JPA library, the data access object (DAO) has all the basic CRUD methods ( and queries) defined and implemented that the UMGC system will requires. The following diagrams provided in Figure 4 and 5 show precisely how each piece of data flows through the system passing from end user, to functional requirement, and into the data access objects that are utilized by the data access layer.

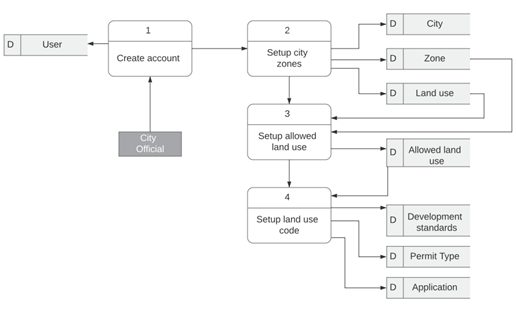


Figure 4 - Data Flow Diagram: UMGC City Zoning Application (Management User)

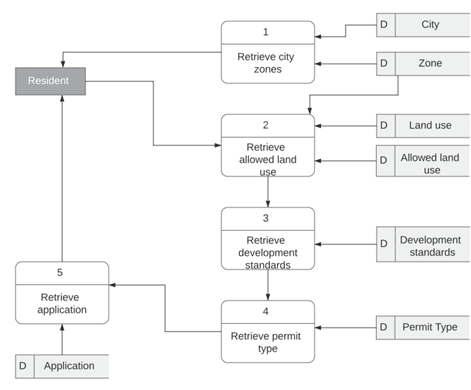


Figure 5 - Data Flow Diagram: UMGC City Zoning Application (Resident User)

# HUMAN INTERFACE DESIGN

The following subsections describe the functionalities of UMGC City Application from the user’s perspective. It shall give the detailed account of how the user will be able to use the system to complete all the expected features based on the requirements provided in the SRS. Mock wireframes are used to facilitate the understanding of the system’s UI.

## Overview of User Interface

The UMGC City Application’s user interface shall follow four basic design principles to ensure the system’s UI is intuitive, efficient, and easy-to-navigate for a wide range of users. The four design principles that will be followed are simplicity, structure, consistency, and tolerance. This means the UI design will be kept modern but minimal to ensure the UI does not become confusing to users or over-crowd the user’s screen with unnecessary content. The UI will follow a basic and logical data gathering process using structured data input forms. In addition, the entire design of the UI meaning color schema, behavior, and content placement shall be kept consistent throughout the entire web application. Lastly, the UI will aid the user in preventing mistakes by confirming all input decisions before acceptance into the system is permitted and provide backward error recovery whenever applicable. The system’s UI navigational design for move users from one UI component to the next are provided in the following sub-sections and a wireframe design for each component is provided.

## Home Page Component

When the user first accesses the UMGC City Admin Portal UI a Home Page Screen shall be displayed that provides a general overview and purpose of the application and prompts the user to either sign up or sign into the web application.

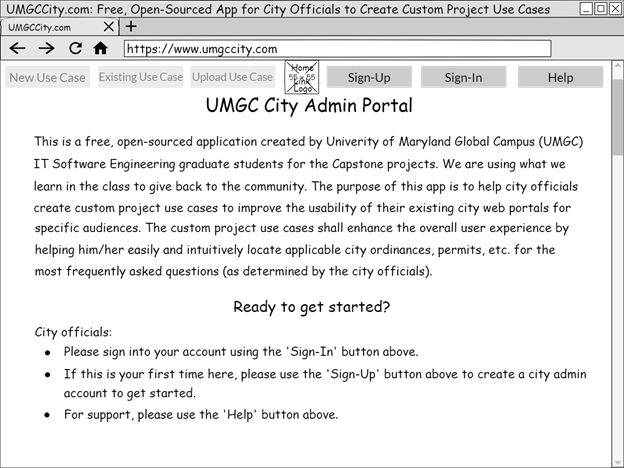


Figure 6 - Wireframe: Home Page Component

## Login Form Component

The UMGC City Admin Portal UI shall display a form prompting the user to input their login credentials, i.e. username and password, after the user selects the log in option from the UI’s home page.

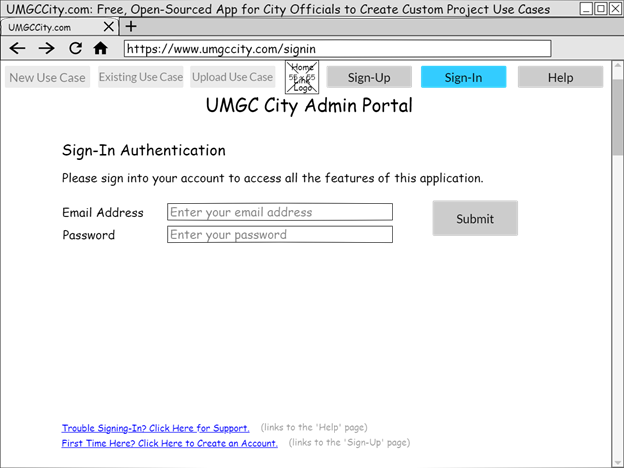


Figure 7 - Wireframe: Login Form Component

## Sign-Up Form Component

The UMGC City Admin Portal UI shall display a sign-up form to the user, after the user selects the sign-up option from the UI’s home page. The sign-up form shall prompt the user to input required data (city, state, email address, password) and optional data (first name and last name), and then submit the form into the system so a new user account will be created for the user.

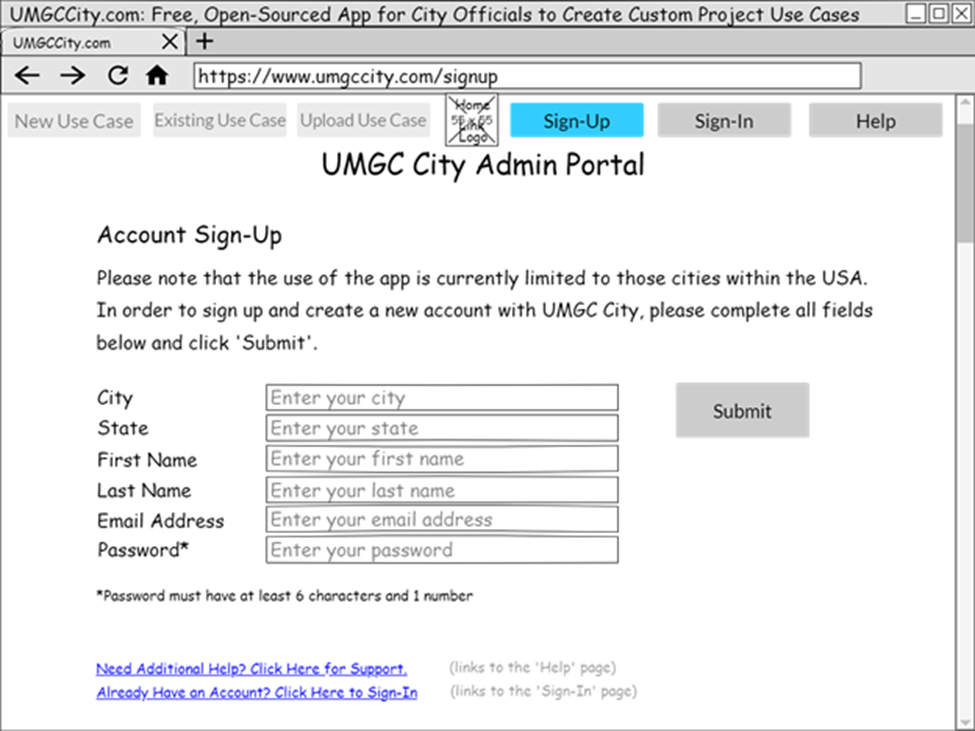


Figure 8 - Wireframe: Sign-Up Form Component

## Credential Recovery Help Component

The UMGC City Admin Portal UI shall display a component consisting of two separate forms after the user selects the help option from the home page component. The purpose of the first form shall be to assist the user in retrieving an email address associated with a registered user account. The second form shall be to retrieve a user’s password associated with a specific email address registered to the system. After the user has completed one of the two UI forms the user will be prompted to submit the form by selecting the submit button next to the form that was completed.

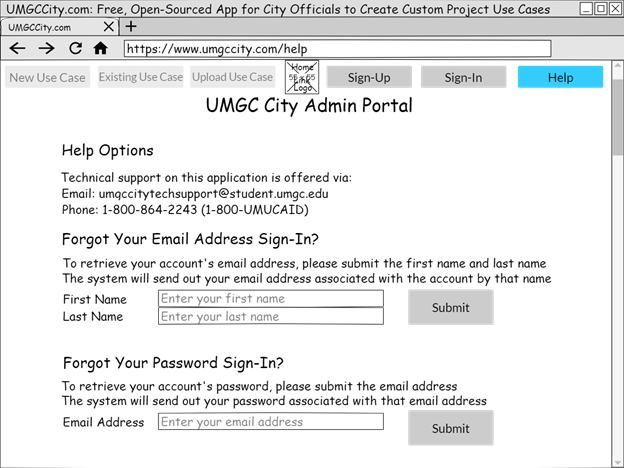


Figure 9 - Wireframe: Credential Recovery Help Component

## Landing Portal Component

The UMGC City Admin Portal UI shall display the Landing Page Screen to the user after the user successfully inputs valid login credentials into the sign-in component. The Landing Page shall display the following options to the user: create a new use case, edit existing use cases, or upload use case file.

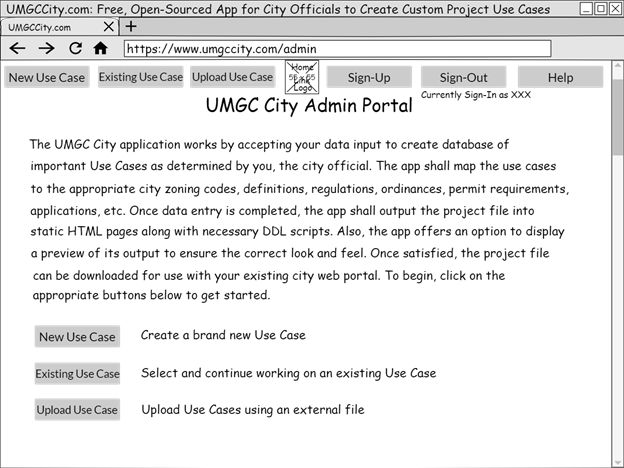


Figure 10 - Wireframe: Landing Page Component

## Manual Use Case Component

The UMGC City Admin Portal UI shall display a form prompting the user to manually input a new use into the system after the user selects the new use case option from the landing page component. The form will require the user to input the following data before allowing it to be submitted for processing: use case name, zoning code use case belong to, definition of zoning code, development standards links and permit link associated with the use case and its zoning code, and an application link to obtain the required permit.

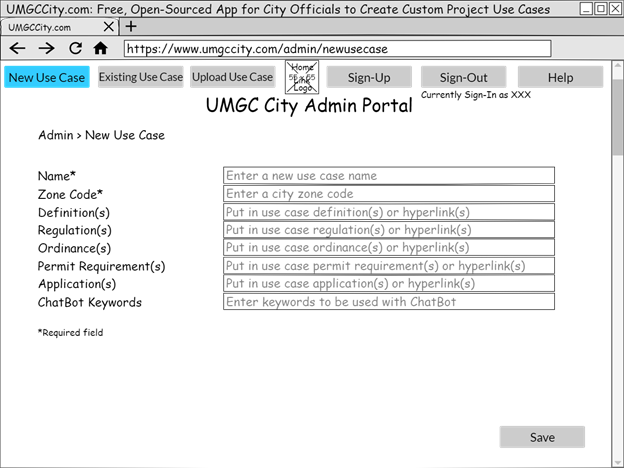


Figure 11 - Wireframe: Manual Use Case Component

## Auto Upload Use Cases Component

The UMGC City Admin Portal UI shall display a file upload form to user after the user selects the upload use cases file from the UI’s land page component. The user will be prompted the user to choose a csv text file from their local filing system, once selected the system will validate the files structure and format before processing it into the system.

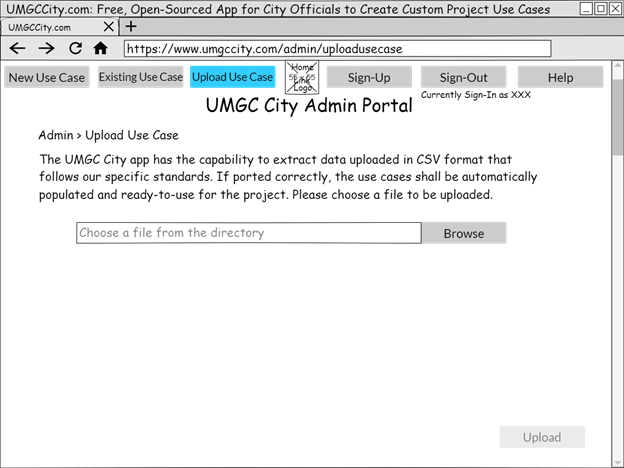


Figure 12 - Wireframe: Auto Upload Use Cases Component

## Manage Existing Use Cases Component

The UMGC City Admin Portal UI shall display the manage existing use case component the user after the user selects the “Manage Existing Use Case” option from the UI’s landing page component. The Manage Existing Use Case Screen shall display all the user’s current use cases saved within the system and prompt the user to choose one use to either edit or delete. The Manage Existing Use Case Screen shall also provide the user with the option to export the project once the user is done inputting and editing all their use case data.

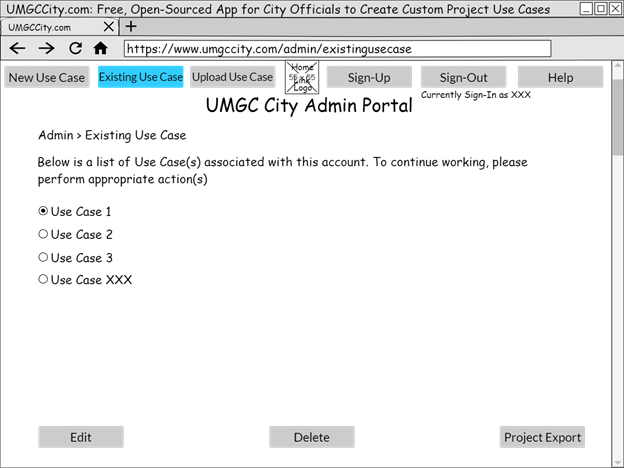


Figure 13 - Wireframe: Mange Existing Use Case Component

## Edit Use Case Component

The UMGC City Admin Portal UI shall display the edit use case component after the user makes a use case selection from the manage use cases component and submits their edit choice to the system by selecting the edit button. The edit use case component shall display a prepopulated use case form, like the new use case component, but with all the data from the previously selected use case filled into the form. The user shall be able to make any data edits directly into the prepopulated form and once done submit their newly edited data to the system for reprocessing.

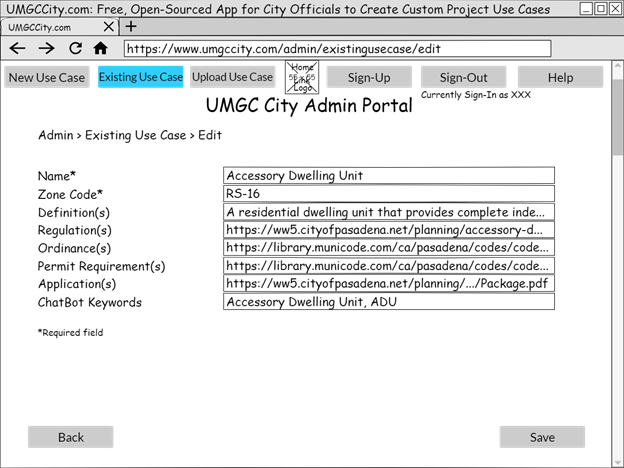


Figure 14 - Wireframe: Edit Use Case Component

## Delete Use Case Component

The UMGC City Admin Portal UI shall display the delete use case component to the user after the user selects a use case from the list of existing uses displayed to them by the manage existing use case component and then selects the “Delete Use Case” option on the UI. The delete use case component shall prompt the user to confirm their deletion choice performing submitting the user’s request into the system for processing.

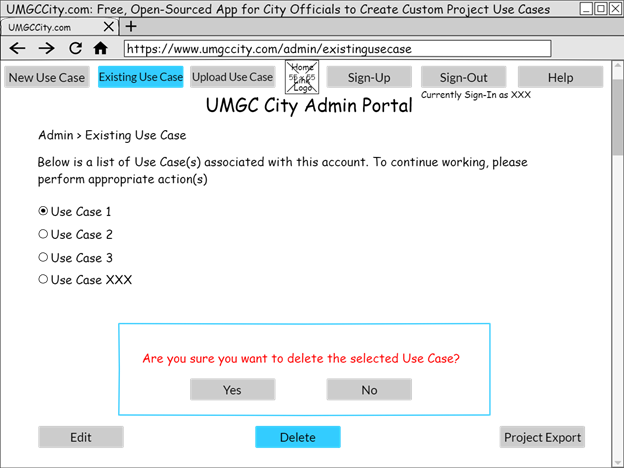


Figure 15 - Wireframe: Delete Use Case Component

## Export Project Component

The UMGC City Admin Portal UI shall display the export project component after the user selects the “Project Export” option from the manage existing use case component The export project component shall display a brief overview of the content that will be generated by the system and prompt the user for one of the following inputs: generate project, preview project, download project, and return to Manage Existing Use Case Screen.

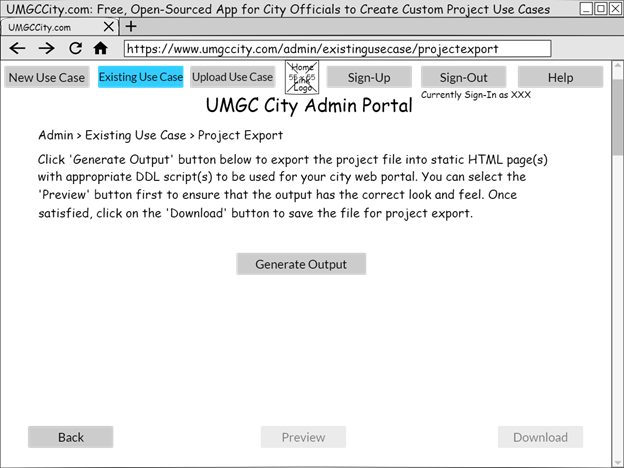


Figure 16 - Wireframe: Export Project Component

## Generate Output Component

The UMGC City Admin Portal UI shall display the generate output component to the user after the “Generated Output” selection is made from the export project component. The generate output component shall display the system’s progress status while converting the user’s project data into a static html page. Once the system’s project generator process is completed the UI will display an alert message to the user notifying the user to their completed project.

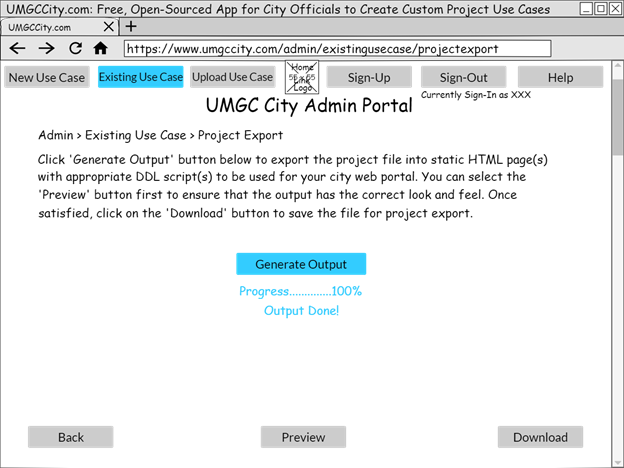


Figure 17 - Wireframe: Generate Output Component

## Display Preview Component

The UMGC City Admin Portal UI shall display the display preview component to the user after the user selects the “Preview” choice from the export project component, but only after the project generator process has been completed through the generate project component. The display preview component shall display a mock screen sample of the user’s static html page.

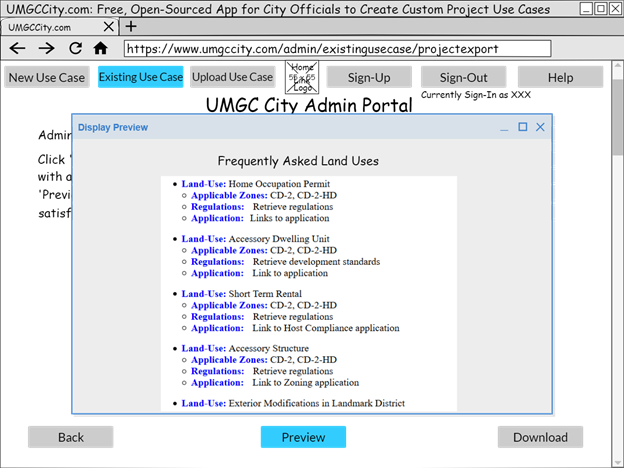


Figure 18 - Wireframe: Display Preview Wireframe

## Download Project Component

The UMGC City Admin Portal UI shall display the download project component to the user after the “Download” selection is chosen from the export project component, but only after the project generator process has been completed through the generate project component. The download project component shall prompt the user to select a storage location within their local filing system, then after the user makes their storage choice the UI shall kick off the download project process.

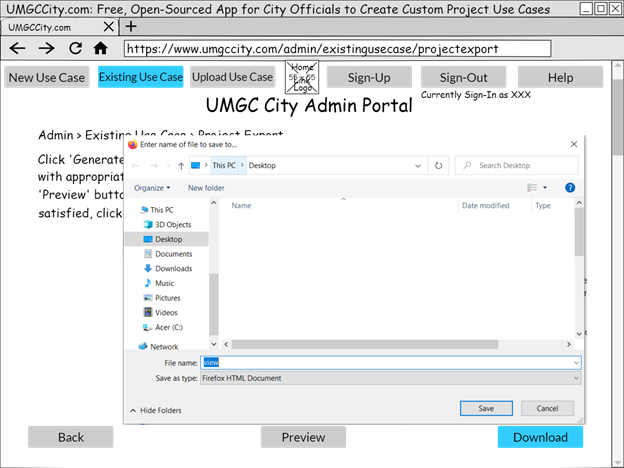


Figure 19 - Wireframe: Download Project Component

## Interactive Map Component

The interactive map component shall only be displayed to public users visiting the City of Pasadena, California’s zoning web page. The interactive map component will be made available to public users through a hyperlink located on the City of Pasadena’s zoning web page. The map shall display specific zoning information associated to the zone map the map that the user clicks on.



Figure 20 - Interactive Map Component

## Quick Reference Page

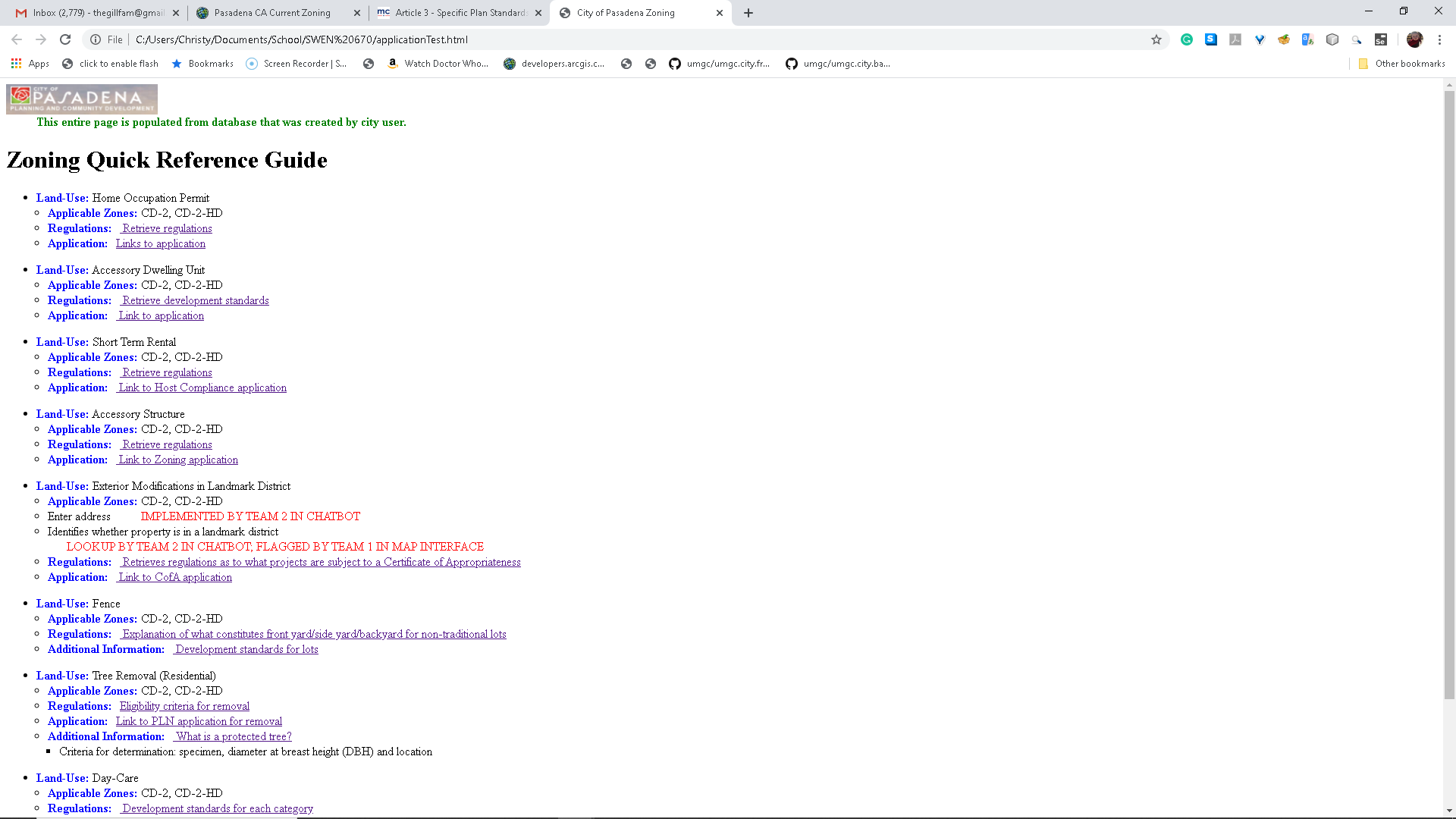


Figure 21 - Quick Reference Page

# REQUIREMENT MATRIX

The table supplied below is a requirements matrix for the UMGC City Application. It’s a cross-reference table that links the software design components to the SRS requirements for easy understanding of the system.

Table 3 - Requirements Matrix

| **ID** | **Requirement ID** | **Requirement Description** | **System Design Component** | **Section Reference** |
| --- | --- | --- | --- | --- |
| **1** | OE-1.1 | For increased accessibility, the application shall operate on a web server compatible with a common, modern browser. | Web Database Application | 6.1.1  6.1.15 |
| **2** | OE-2.1 | The application shall authenticate users using a secure validation process and Hypertext Transfer Protocol Secure (HTTPS) requests. | Web Database Application | 5.3  5.4  5.5  6.1.2  6.1.3  6.1.4 |
| **3** | OE-3.1 | The system shall be able to build, deploy and maintain a relational database to store city information | Web Database Application | 4  5.4  5.5  6.1.1 |
| **4** | REQ-1.1 | The home page of UMGC City Admin Portal system shall be accessible to the user via the following web URL: https://www.umgccity.com. | Web Database Application | 6.1.1  6.1.15 |
| **5** | REQ-1.2 | There shall be a consistent main navigation bar along the top of web pages to allow the user to quickly navigate within the application. | Web Database Application | 6.1 |
| **6** | REQ-1.3 | The UMGC City Admin Portal system shall provide a new user with the ability to sign up for an account by clicking on the “Sign-Up” button. | Web Database Application | 6.1.3 |
| **7** | REQ-1.4 | The UMGC City Admin Portal system shall allow a user to sign into his/her existing account by clicking on the “Sign-In” button. | Web Database Application | 6.1.2 |
| **8** | REQ-1.5 | The UMGC City Admin Portal system shall provide a user with a web page dedicated to providing help and support by clicking on the “Help” button. | Web Database Application | 6.1.4 |
| **9** | REQ-1.6 | The user shall be brought to the landing page once successfully logged into the system.  The system features of creating a new use case, editing existing use cases, and uploading a use case are now available. | Web Database Application | 6.1.5 |
| **10** | REQ-1.7 | The UMGC City Admin Portal system works by accepting data inputted by the user.  The user creates a use case for his/her city. The application shall map the use case to the appropriate city data. | Web Database Application | 6.1.6 |
| **11** | REQ-1.8 | The UMGC City Admin Portal system shall allow a user to manually add a new use case, edit existing use cases, and delete a use case from a list within the account.  In addition, the system shall allow the user to upload use cases from an external file using the “Upload Use Case” button. | Web Database Application | 6.1.6  6.1.7  6.1.8  6.1.9  6.1.10 |
| **12** | REQ-1.9 | The UMGC City Admin Portal system shall provide a user with a certain amount of memory storage space that is linked to his/her account.  The user can save the use cases and come back to them later. | Web Database Application | 5  6.1.2  6.1.8 |
| **13** | REQ-1.10 | The UMGC City Admin Portal system shall process the project data inputted by the user and generate proper output of static HTML page with appropriate DDL scripts using the “Generate Output” button. Refer to Appendix A for more info. | Web Database Application | 6.1.11  6.1.12  6.1.13  6.1.14  6.1.16 |
| **14** | REQ-2.1 | When the user drags either scrollbar, the map image is moved within the containing window.  Scrolling is set to default Chrome settings. | Interactive Map | 6.1.15 |
| **15** | REQ-2.2 | When the user clicks on a zone within the map, a modal opens on top of the map to display specific information to the selected zone. The modal is designed using in-line JavaScript and custom CSS. | Interactive Map | 6.1.15 |
| **16** | REQ-2.3 | When the user clicks on the “Zone Information” link, the browser navigates to an external page on the Municode website that contains information that is relevant to the selected zone. | Database Query | 6.1.15 |
| **17** | REQ-2.4 | When the user clicks on any “Application” link, the browser navigates to an external page on the Municode website that contains the applications that are relevant to the selected zone. | Database Query | 6.1.15 |
| **18** | REQ-2.5 | When the user clicks on the “Quick Reference Page” link, the browser navigates to an internal page that contains information designed by the city officials.  The information on this page is retrieved from the database. | Database Query | 6.1.16 |
| **19** | REQ-2.6 | When the user clicks on the “Need Help?” icon, the ChatBot application opens on top of the map interface.  The ChatBot application is designed by UMGC City Team 2. | ChatBot | 6.1.15 |
| **20** | REQ-3.1 | The system shall direct the user to the “Quick Reference Page” | Database Query | 6.1.16 |
| **21** | REQ-4.1 | The system shall direct the user to the “Home Occupation” regulations page after the user clicks the regulation link. | Database Query | 6.1.16 |
| **22** | REQ-4.2 | The system shall direct the user to the “Home Occupation” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **23** | REQ-5.1 | The system shall direct the user to the “Accessory Dwelling Unit” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **24** | REQ-5.2 | The system shall direct the user to the “Accessory Dwelling Unit” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **25** | REQ-6.1 | The system shall direct the user to the “Short Term Rental” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **26** | REQ-6.2 | The system shall direct the user to the “Short Term Rental” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **27** | REQ-7.1 | The system shall direct the user to the “Accessory Structure” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **28** | REQ-7.2 | The system shall direct the user to the “Accessory Structure” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **29** | REQ-8.1 | The system shall direct the user to the “Exterior Modifications in Landmark District” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **30** | REQ-8.2 | The system shall direct the user to the “Exterior Modifications in Landmark District” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **31** | REQ-9.1 | The system shall direct the user to the “Fence” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **32** | REQ-9.2 | The system shall direct the user to the “Fence” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **33** | REQ-10.1 | The system shall direct the user to the “Tree Removal (Residential)” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **34** | REQ-10.2 | The system shall direct the user to the “Tree Removal (Residential)” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **35** | REQ-11.1 | The system shall direct the user to the “Day-Care” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **36** | REQ-11.2 | The system shall direct the user to the “Day-Care” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **37** | REQ-12.1 | The system shall direct the user to the “Sober Living Facility” regulations page after the user clicks the regulations link. | Database Query | 6.1.16 |
| **38** | REQ-12.2 | The system shall direct the user to the “Sober Living Facility” application page after the user clicks the application link. | Database Query | 6.1.16 |
| **39** | SI-1.1 | The system shall interact with PostgreSQL RDBMS for all data transactions. | Web Database Application/Database Query | 4  5 |
| **40** | SI-2.1 | The system's server-side Java EE application shall interact with the system's client-side Angular web application. | Web Database Application/Database Query | 5.1 |
| **41** | SI-3.1 | The system’s client-side Angular application shall interact with the system’s server-side Java EE application. | Web Database Application/Database Query | 5.1  5.2  5.5 |
| **42** | SI-4.1 | The system’s client-side Angular application shall interact with the file storage system of the user’s device. | Web Database Application/Data Processing | 5.2  5.5 |
| **43** | SI-5.1 | The system’s server-side Java EE application shall interact with a Python script to parse CSV data files. | Web Database Application/Data Processing | 5.1  5.2  5.5  6.1.7 |
| **44** | SI-6.1 | The system shall interact with a Chatbot Java web application developed by UMGC City Team 2. | ChatBot | N/A |
| **45** | SI-7.1 | The system shall be able to build, deploy and maintain a relational database to store city information. | Web Database Application | 4 |
| **46** | UI-1.1 | The UMGC City Admin Portal UI shall display a Home Page Screen. | Web Database Application | 6.1.1 |
| **47** | UI-1.2 | The UMGC City Admin Portal UI shall display a Sign-in Screen. | Web Database Application | 6.1.2 |
| **48** | UI-1.3 | The UMGC City Admin Portal UI shall display a Sign-Up for new users. | Web Database Application | 6.1.3 |
| **49** | UI-1.4 | The UMGC City Admin Portal UI shall display a Help Screen. | Web Database Application | 6.1.4 |
| **50** | UI-1.5 | The UMGC City Admin Portal UI shall display a Landing Page Screen to the user when the user inputs valid login credentials into Sign-in Screen. | Web Database Application | 6.1.5 |
| **51** | UI-1.6 | The UMGC City Admin Portal UI shall display the New Use Case Screen to the user when the user selects the “New Use Case” option. | Web Database Application | 6.1.6 |
| **52** | UI-1.7 | The UMGC City Admin Portal UI shall display the File Upload Screen to the user when the user selects the “File Upload” option. | Web Database Application | 6.1.7 |
| **53** | UI-1.8 | The UMGC City Admin Portal UI shall display the Manage Existing Use Case Screen to the user when the user selects the “Manage Existing Use Case” option.6.1.4 | Web Database Application | 6.1.8 |
| **54** | UI-1.9 | The UMGC City Admin Portal UI shall display the Delete Use Case Screen to the user when the user selects a use case and selects the “Delete Use Case” option. | Web Database Application | 6.1.8  6.1.10 |
| **55** | UI-1.10 | The UMGC City Admin Portal UI shall display the Export Project Screen to the user when the “Export Project” option is selected on the Manage Existing Use Case Screen. | Web Database Application | 6.1.11 |
| **56** | UI-1.11 | The UMGC City Admin Portal UI shall display the Generate Output Screen to the user when the “Generate output” option is selected on the Export Project Screen. | Web Database Application | 6.1.12 |
| **57** | UI-1.12 | The UMGC City Admin Portal UI shall display the Display Preview Screen to the user when the “Display Preview” option is selected on the Export Project Screen. | Web Database Application | 6.1.13 |
| **58** | UI-1.13 | The UMGC City Admin Portal UI shall display the Download Project Screen to the user when the “Download” option is selected on the Export Project Screen. | Web Database Application | 6.1.14 |

# Appendix A

**DDL Script for UMGC City**

**Web Application Database**

DROP DATABASE "UMGCity";

CREATE DATABASE "UMGCity"

WITH

OWNER = postgres

TABLESPACE = pg\_default

CONNECTION LIMIT = 500;

DROP SCHEMA IF EXISTS public CASCADE;

DROP TABLE IF EXISTS city\_user CASCADE;

DROP TABLE IF EXISTS city CASCADE;

DROP TABLE IF EXISTS zone\_land\_use CASCADE;

DROP TABLE IF EXISTS zone CASCADE;

DROP TABLE IF EXISTS development\_standards CASCADE;

DROP TABLE IF EXISTS allowed\_land\_use CASCADE;

DROP TABLE IF EXISTS permit\_type CASCADE;

DROP TABLE IF EXISTS application CASCADE;

CREATE SCHEMA public

AUTHORIZATION postgres;

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

COMMENT ON EXTENSION "uuid-ossp" IS 'generate universally unique identifiers (UUIDs)';

CREATE TABLE public.city\_user

(

id uuid DEFAULT uuid\_generate\_v4 (),

password character varying(30) NOT NULL,

last\_name character varying(50),

first\_name character varying(50),

email\_address character varying(100) NOT NULL,

CONSTRAINT city\_user\_id PRIMARY KEY (id)

);

ALTER TABLE public.city\_user

OWNER to postgres;

CREATE TABLE public.city

(

id uuid DEFAULT uuid\_generate\_v4 (),

city\_user\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES city\_user(id) ON DELETE CASCADE ON UPDATE CASCADE,

name character varying(50) NOT NULL,

state character varying(50) NOT NULL,

CONSTRAINT city\_id PRIMARY KEY (id)

);

ALTER TABLE public.city

OWNER to postgres;

CREATE TABLE public.zone\_land\_use

(

id uuid DEFAULT uuid\_generate\_v4 (),

city\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES city(id) ON DELETE CASCADE ON UPDATE CASCADE,

description character varying(100) NOT NULL,

CONSTRAINT zone\_land\_use\_id PRIMARY KEY (id)

);

ALTER TABLE public.zone\_land\_use

OWNER to postgres;

CREATE TABLE public.zone

(

id uuid DEFAULT uuid\_generate\_v4(),

city\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES city(id) ON DELETE CASCADE ON UPDATE CASCADE,

zone\_symbol character varying(5) NOT NULL,

description character varying(100) NOT NULL,

CONSTRAINT zone\_id PRIMARY KEY (id)

);

ALTER TABLE public.zone

OWNER to postgres;

CREATE TABLE public.allowed\_land\_use

(

id uuid DEFAULT uuid\_generate\_v4(),

zone\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES zone(id),

zone\_land\_use\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES zone\_land\_use(id) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT allowed\_land\_use\_id PRIMARY KEY (id)

);

ALTER TABLE public.allowed\_land\_use

OWNER to postgres;

CREATE TABLE public.development\_standards

(

id uuid NOT NULL DEFAULT uuid\_generate\_v4(),

zone\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES zone(id) ON DELETE CASCADE ON UPDATE CASCADE,

general\_standard\_url character varying(100) NOT NULL,

additional\_standard\_url character varying(100) NOT NULL,

garden\_standard\_url character varying(100) NOT NULL,

frontage\_and\_facades\_standards\_url character varying(100) NOT NULL,

CONSTRAINT development\_standards\_id PRIMARY KEY (id)

);

ALTER TABLE public.development\_standards

OWNER to postgres;

CREATE TABLE public.permit\_type

(

id uuid NOT NULL DEFAULT uuid\_generate\_v4(),

allowed\_land\_use\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES allowed\_land\_use(id) ON DELETE CASCADE ON UPDATE CASCADE,

description character varying(100) NOT NULL,

procedure\_url character varying(100) NOT NULL,

name character varying(100),

CONSTRAINT permit\_type\_id PRIMARY KEY (id)

);

ALTER TABLE public.permit\_type

OWNER to postgres;

CREATE TABLE public.application

(

id uuid NOT NULL DEFAULT uuid\_generate\_v4(),

permit\_type\_id uuid NOT NULL DEFAULT uuid\_generate\_v4() REFERENCES permit\_type(id) ON DELETE CASCADE ON UPDATE CASCADE,

application\_url character varying(100) NOT NULL,

name character varying(50) NOT NULL,

CONSTRAINT application\_id PRIMARY KEY (id)

);

ALTER TABLE public.application

OWNER to postgres;