About My Home Town: Development Guide

CSC478: Software Engineering Capstone

Audrey Yates, William Morgan, Steven Hogue
Group 11

April 29, 2024

Contents

1 Requirements Documentation

1.1 Introduction

1.1.1 Scope

We will construct a cross-platform web application that allows users to check on a variety of data points for their locale. This data may include, but is not limited to, gas prices, current and forecast weather, restaurant/food prices, public facilities, local businesses, and possibly crime information. The goal is to compile as much data from public or low-cost API's into a single interface, adding statistical analysis, and overlaying it into a mapping application (such as Google Maps). As a long-term goal we would like to interpret and nd news articles relating to an area of interest, and pin the location in which they took place on the map.

1.1.2 Definitions, Acronyms, and Abbreviations

- **The Application:** The web application we are building, which will be called "About My Home Town"
- **API:** Application Programming Interface
- **Authenticated User:** A user who has created an account and logged in
- **Bcrypt:** A password hashing function
- **HTTP:** Hypertext Transfer Protocol
- **HTTPS:** Hypertext Transfer Protocol Secure
- **JSON:** JavaScript Object Notation
- **Local Storage:** A method of storing data in the user's browser
- **Non-Authenticated User:** A user who has not logged in
- **RESTful API:** A type of API that follows the REST architectural style
- **Salt:** A random value that is used as an additional input to a one-way function that hashes data
- **Session Token:** A unique token generated when a user logs in, which is used to authenticate the user
- **User:** The end user of the application
- **Widget:** A small unit of data presentation, containing a sample of data we provide (ex. Weather Widget, Restaurant Widget, etc.)

1.2 General Description

1.2.1 Product Perspective

The application we are building will be useful for users who want to access a wide range of data points for their locale in a single interface. This comprehensive approach saves users time and effort by eliminating the need to visit multiple websites or applications to

gather this information. By building this application, we are addressing the need for a centralized platform that provides users with a wealth of localized information, ultimately making it easier for them to make informed decisions and navigate their surroundings.

1.2.2 Product Functions

This application will aggregate data from various public free and paid sources into a single interface. It will allow users to save their data by logging in and creating an account with their zip code, and then it will show them information regarding their local area, such as weather and restaurant information.

1.2.3 User Characteristics

Our prospective users are people who are interested in accessing a range of data points for a given area, such as users looking to move into a new area, users who are traveling, or users who are simply curious about their current surroundings and want to explore. The thing all of these users have in common is that they are looking for a centralized platform that provides them with a wealth of localized information.

1.2.4 General Constraints

This has to be built as a web application with constant network access, meaning that it necessarily *cannot* work offline. Due to this, the end user must access our application through a web browser that supports modern JavaScript features.

Another constraint of this is that the data we are pulling in is only as good as the data we are given. If a data source is inaccurate, our application will be inaccurate as well.

1.2.5 Assumptions and Dependencies

There are few assumptions from the end user, though as mentioned above, we will assume both that the end user is accessing from a modern web browser and that the data we are pulling in is accurate.

1.3 Specific Requirements

1.3.1 Data Sources (1)

Requirement Number	Description
1.0.0	The application must pull in the most recently available weather data whenever the end user loads the application
1.1.0	The application must pull in the most recently available restaurant data whenever the end user loads the application

1.3.2 User Data (2)

Requirement Number	Description
2.0.0	Any user must be allowed to create an account with **required fields**
2.0.1	Email Address, 45 character string
2.0.2	ZIP Code, either ZIP or ZIP+4 format stored as a string
2.0.3	Password, arbitrary length
2.1.0	Any user must be allowed to create an account with **optional fields**
2.1.1	First Name, 45 character string
2.1.2	Last Name, 45 character string
2.1.2	Bio Information, 256 character string

1.3.3 Authentication Model (3)

Requirement Number	Description
3.0.0	Email addresses must be globally unique
3.1.0	Users must be able to log in with their email and password
3.2.0	Passwords must be handled securely in all cases
3.2.1	Passwords must be at least 6 characters in length
3.2.2	Passwords must be sent to the server over HTTPS
3.2.3	Passwords must be encrypted using Bcrypt with a hash on the server side only
3.2.4	During user verification, passwords should hashed using the salt, and checked against the known hash
3.2.5	Passwords must **NEVER** be stored in plain-text, only during transit
3.3.0	When a user logs in, a session token should be generated
3.3.1	Session tokens must be stored along with the user's email in the database, and a last used date
3.3.2	Session tokens must simultaneously be stores in the user's browser's local storage
3.3.4	To authenticate a user's existing session, the local token must be one of the known session tokens
3.3.5	If a token is older than 24 hours, invalidate it and force log out the user
3.3.6	A valid session token must be required for all user update or modification requests
3.4.0	Users must be able to log out of their account, invalidating the session

1.3.4 User Interface (4)

Requirement Number	Description
4.0.0	The application must have an intuitive user interface
4.1.0	The application must have a home page
4.1.1	This page must be available to only authenticated users
4.1.2	This page must show an overview of the data we present in other locations on the site in widgets
4.2.0	The application must have an about page
4.2.1	This page must be available to all users
4.2.2	This page must show a description of the site and its technology
4.3.0	The application must have a sign up page
4.3.1	This page must be available to only *non* authenticated users
4.3.2	This page must allow the user to create an account based on the User Data and Authentication Model requirements
4.4.0	The application must have a sign in page
4.4.1	This page must be available to only *non* authenticated users
4.4.2	This page must allow the user to sign in based on the User Data and Authentication Model requirements
4.4.2	After a user has signed in, they are now considered authenticated
4.5.0	The application must have a sign out button
4.5.1	This button must be visible to only authenticated users
4.5.2	This page must allow the user to sign out, invalidating their session
4.5.2	After a user has signed out, they are again considered non-authenticated
4.6.0	The application must have a user modification page
4.6.1	This page must be available to only authenticated users
4.6.2	This page must allow the user to modify any of their user data (with exception to the password and email)
4.6.2	After submitting modifications, all data must update immediately and the page should refresh

1.3.5 User API (5)

Requirement Number	Description
5.0.0	The application must have a RESTful API that allows for the creation, reading, updating, and deletion of user data
5.1.0	This API must be available at the 'api' subdomain of the application site
5.2.0	All API specifications must return the User object as JSON on a succesful result, otherwise return an HTTP error code
5.3.0	The API must include an endpoint to allow user creation (sign up)
5.4.0	The API must include an endpoint to allow user authentication (sign in)
5.5.0	The API must include an endpoint to allow sesion token validation
5.6.0	The API must include an endpoint to allow the user modification (update user)

2 Design Documentation

The following design elements were composed prior to any work on the application, and should generally be followed throughout the development process.

2.1 Architecture Diagrams

This diagram shows the layout of the AWS architecture we are using for our application. The user interacts with the CloudFront Distribution, which then sends the request to the S3 bucket. The S3 bucket then returns the static template for the site to the end user.

From there, the user's browser interacts with the API Gateway, which then sends the request to the Lambda function. The Lambda function then interacts with the DynamoDB database to get or set the user's data. If no live data is available, the Lambda function will then interact with the external API to get the data.

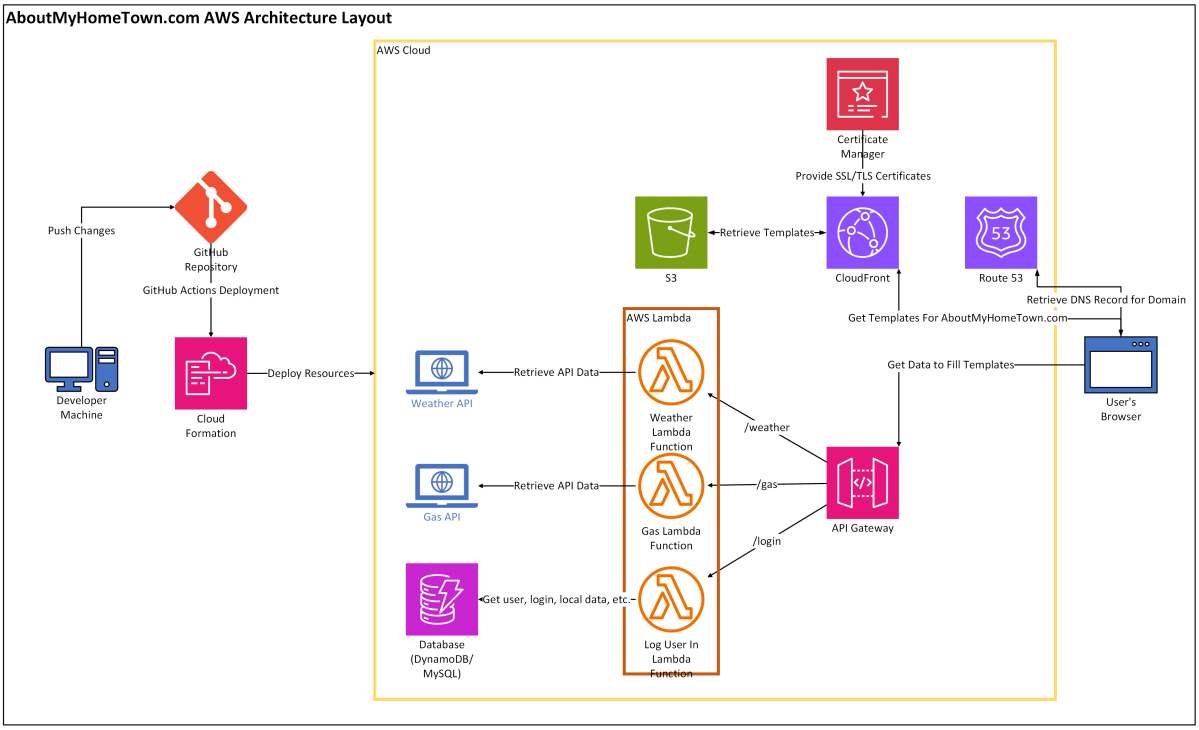
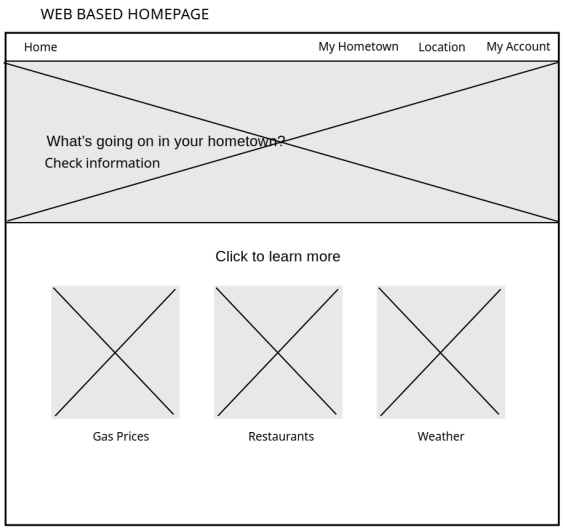


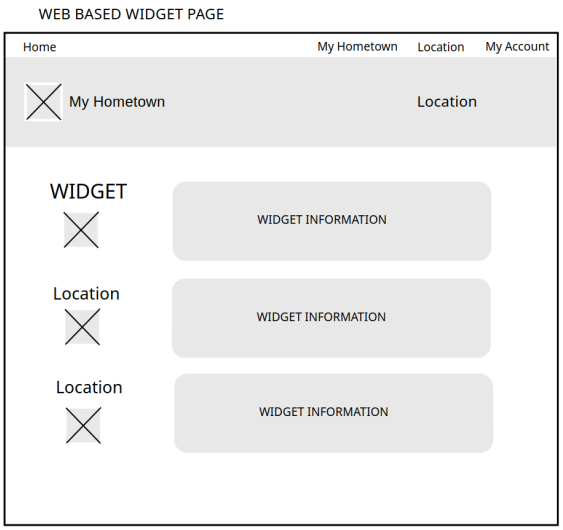
Figure 1: AWS Architecture Layout

2.2 Wireframes

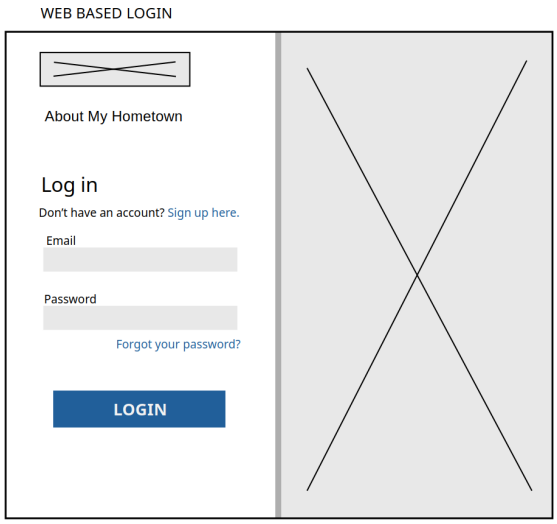
This figure shows the initial wireframed designs for our sign in page, homepage, and widgets page. We added to these designs, but the overall design philosophy remained the same.



(a) Home Page



(b) Widgets Page



(c) Another Page

3 Source Code Documentation

4 Testing Documentation

Note: for all testing, you need to completely set up your environment. This means running at least:

```
npm install
```

4.1 End to End Testing

This application supports full end to end testing of the application completely locally on the developer's machine. This is done in a few stages, with each stage representing a different part of the infrastructure.

Regardless of how you decide to test, if you follow either guide to set up the application listed below, you can access the application at `http://localhost:4200`. This allows the developer to see the application as the end user would see it, while having the ability to immediately test and see any changes they make.

4.1.1 Piece by Piece

- **Frontend:** The frontend is tested using the Angular CLI. The following will start the Angular CLI locally:

  ```
  ng serve --open --watch --port 4200
  ```

  Once the Angular CLI is running, you can access the application at `http://localhost:4200`.

- **Database:** The database is tested by spinning up a MySQL Docker container. The following will start the MySQL container:

  ```
  docker run --name my-local-mysql \\
      -e MYSQL_ROOT_PASSWORD=localpass \\
      -e MYSQL_DATABASE=mydb -p 3306:3306 -d mysql:latest \\
      --default-authentication-plugin=mysql_native_password
  ```

  This will start a *local and not-for-production* MySQL container with the root password set to 'localpass', the database name set to 'mydb', and the container will be accessible on port 3306.

- **Backend:** The backend is tested using the AWS SAM CLI. The following will start the SAM API locally:

  ```
  cd aboutmyhometown-lambda-stack
  sam local start-api -p 4201
  ```

  Once the SAM API is running, you can access the API at `http://localhost:4201`.

4.1.2 Helpers

While you can test the application piece by piece, we have set up a much simpler way to test. The following will start the entire application locally all at once, handling seting up and tearing down the database, and starting the frontend and backend:

```
npm run start-all
```

Or, for Linux users:

```
npm run start-all-linux
```

4.2 AWS CloudFormation Testing

The AWS infrastructure deployed from this repository (the API Gateway and Lambda functions) can be tested using the AWS SAM CLI as well. The following command can be used to validate that the templates are valid configurations:

```
sam validate -t aboutmyhometown-lambda-stack/template.yaml
```

4.3 Unit Testing

4.3.1 Backend API Unit Tests

The unit tests for the backend API are written in Python and can be found in the *aboutmyhometown-lambda-stack/tests* directory. These run automatically in the CI/CD pipeline for the lambda deployment.

Session Tests:

Test	Req.	Rationale	Input	Expected
1	3.3.x; 5.5.0	The session token parameter is required at the session token endpoint	Session token is not provided	HTTP 400 is returned
2	3.3.x; 5.5.0	An invalid session token is not accepted	An invalid token	HTTP 401 is returned
3	3.3.x; 5.5.0	A valid session token is returns user information	A valid session token for a user	HTTP 200 is returned
4	3.3.x; 5.5.0	An internal error returns invalid token	An unhandled internal data type	HTTP 401 is returned
5	3.3.x; 5.5.0	An invalid JSON object provided returns an error	An invalid JSON body	HTTP 400 is returned

Sign In Tests:

Test	Req.	Rationale	Input	Expected
6	3.1.0	Valid credentials return a valid user object	Valid user credentials	HTTP 200 and proper user object are returned
7	5.4.0	An invalid password fails to authenticate	An invalid password	HTTP 401 is returned
8	5.4.0	Missing email or password fails to authenticate	Missing username or password	HTTP 400 is returned
9	5.4.0	An invalid JSON object provided returns an error	An invalid JSON body	HTTP 400 is returned

Sign Up Tests:

Test	Req.	Rationale	Input	Expected
10	2.0.x; 5.3.0	All required parameters are really required for account creation	User creation request missing zip code or email	HTTP 400 is returned
11	3.0.0	All user's emails must be unique	A user who's email address already exists in database	HTTP 400 is returned
12	2.0.x	Given valid and enough information, a user can succesfully be created	Valid information for a new user	HTTP 200 and valid user object are returned
13	2.1.x	Given optional user information, a user can successfully be created	Valid information for a new user, including optional fields	HTTP 200 and valid user object are returned
14	5.3.0	An invalid JSON object provided returns an error	An invalid JSON body	HTTP 400 is returned

Modify User Tests:

Test	Req.	Rationale	Input	Expected
15	4.6.0	All required parameters are required for user modification	User modification request missing zip code or email	HTTP 400 is returned
16	3.3.6	Invalid session tokens deny user modifications	User modification request missing zip code or email	HTTP 400 is returned
17	3.3.6	Given new user info and a valid session token, a user can be updated	Valid user email and session token	HTTP 200 and updated user object are returned
18	4.6.0	Non-existant users are gracefully handled	Invalid user email provided to session	HTTP 400 is returned
19	4.6.0	An invalid JSON object provided returns an error	An invalid JSON body	HTTP 400 is returned

5 Known Bugs

- We were unable to get the HTTPS version of the redirect to 'www.' to work properly
  - Any of the following redirect properly to the main site:
    - `http://aboutmyhometown.com`
    - `http://www.aboutmyhometown.com`
    - `https://www.aboutmyhometown.com`
  - However, the following does not redirect properly:
    - `https://aboutmyhometown.com`
  - This seems to be a technology issue with the way S3 redirects, and it is unable to redirect HTTPS traffic without configuring a secondary route through the CloudFront Distribution