WhatToEat

Iteartion 1 Report

github.com/melongbob/WhatToEat

Shawn An(301323174)

Rocky Ken(301247347)

SeungYun Kim(301325462)

Irfandi Riawan(301276690)

Oscar Zhen(301236327)

Abstract

WhatToEat is a web application that decides for the user what they should eat. The application decides for the user what they should eat based on their preferences, then factor in weather conditions and location to narrow down their choices. Our plan is to use the map API and weather API to identify the user’s environment, and then use food API’s to suggest a few choices of optimal meal options. Other potential features may provide the distance to location, customer reviews etc.

Requirements

User Stories for Iteration 1

A user that wants to use the application must sign up for a free account. User clicks on the signup button, which redirects them to a page where they enter their preferred username and password, and asks the user to confirm their password as well.

Once the information provided has been validated, a user account is created. The user logs in using the specified login ID to log into their account.

An administrator can make an admin account, which allows the administrator to oversee all users from their account. To make a new account as an admin, the administrator signs up using the same page used to sign up as a user. The administrator must follow several rules in creating an admin account. The administrator enters a username that follows several rules, and then enter their preferred password, as well as confirm their password.

Once the information provided has been validated, an admin account is created. Administrators log in from the login page using the specified admin login ID to log into their admin account. An admin account will have an additional button on their toolbar labeled "Users." This button allows the administrator to manage registered users.

Planning and Design

Getting the site up and running from Rails is the top priority. Once the site is up, the login feature will be implemented first. This login feature will be tested using an account premade into the code.

Once the login feature is ready, the sign up feature will be implemented next. The implementation of the sign up feature will implement the creation of a user account and an admin account.

Progress

As of June 26, the application has several features, most prominently a login functionality.

The sign up feature is partially implemented, which allows both users and administrators to make an account. An administrator account is designed to oversee users. Currently, there are no differences between an administrator account and a user account, except a placeholder button for the administrator labeled “Users” which, to be implemented in future iterations, allows administrators to manage registered users.

So far, a stricter input validation for signing up is not yet implemented. This allows user to sign up with a username that contains non-alphanumeric characters. Restrictions on username/password length is also yet to be implemented. The current version of the application also allows user to sign up even though there is a user signed in.

A user interface has also been implemented, albeit minimal. Due to current system’s design, a user can still sign up even though they have been signed in as a different user.

Testing

Current iteration’s functional testing focuses on the login and signup data entry.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Data | Description | Input | Output | Expected | Comment |
| Normal | Signup - signing up as user | id: user    pw: test    confirm: test | User account created. | A user account is created under the specified name. | Working as expected. |
| Signup - signing up as admin | id: admin    pw: admin    confirm: admin | Name has already been taken. | An admin account is created. | Admin account already present from previous testing (undocumented). Clearing database should solve this problem. |
| Login - logging in as user | id: user    pw: test | Logged in as user. | User logged in as specified user. | Working as expected. |
| Login - logging in as admin | id: admin    pw: admin | Logged in as admin. | Admin logged in as specified admin. | Working as expected. |
| Extreme | Signup - signing up with maximum length username | id: 01234567890123456789012345678901234567890124356789    pw: press | User account created. | A user account is created under the specified name. | Working as expected. |
| Abnormal | Login - logging in as user with wrong password | id: user    pw: pest | Invalid name/password combination. | User cannot login. A wrong password message is returned. | Working as expected. Maybe change error message to show wrong password only? |
| Login - logging in as admin with wrong password | id: admin    pw: nimda | Invalid name/password combination. | Admin cannot login. A wrong password message is returned. | Working as expected. Maybe change error message to show wrong password only? |
| Login - logging in as a nonexistent user | id: userTwo    pw: testPW    (userTwo does not exist) | Invalid name/password combination. | Cannot login. A nonexistent user error message is returned. | Working as expected. Maybe change error message to show wrong username only? |
| Login - logging in as a nonexistent admin | id: admin    pw: admin    (No admin account was created) | Logged in as admin. | Cannot login. A nonexistent admin error message is returned. | Admin account already present from previous testing (undocumented). Clearing database should solve this problem. |
| Signup - signing up with an existing username | id: user    pw: test    confirm: test    (“user” has signed up) | Name has already been taken. | Cannot sign up. A username taken error message is returned. | Working as intended. |
| Signup - signing up using non-alphanumeric characters | id: &$\*%()    pw: rand    confirm: rand | User account created. | Cannot sign up. Returns a message telling that only valid characters can be used. | Additional username validation required. |
| Signup - signing up with a very long username | id: abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz    pw: test    confirm: test | Name is too long (50 characters). | Cannot sign up. Returns a message telling that a username must be within a specified number of characters. | Working as intended. |
| Signup - signing up with a non-matching password and password confirmation | id: userFoo    pw: test  confirm: bash | Password confirmation does not match password. | Cannot sign up. Returns a message to enter same confirmation password as the password entered. | Working as intended. |
| Signup - signing up with no password | id: testUser    pw: [blank]    confirm: [blank] | Password cannot be blank / Password is too short (1 character). | Cannot sign up. Returns message to enter password. | Working as intended. |

User Interface

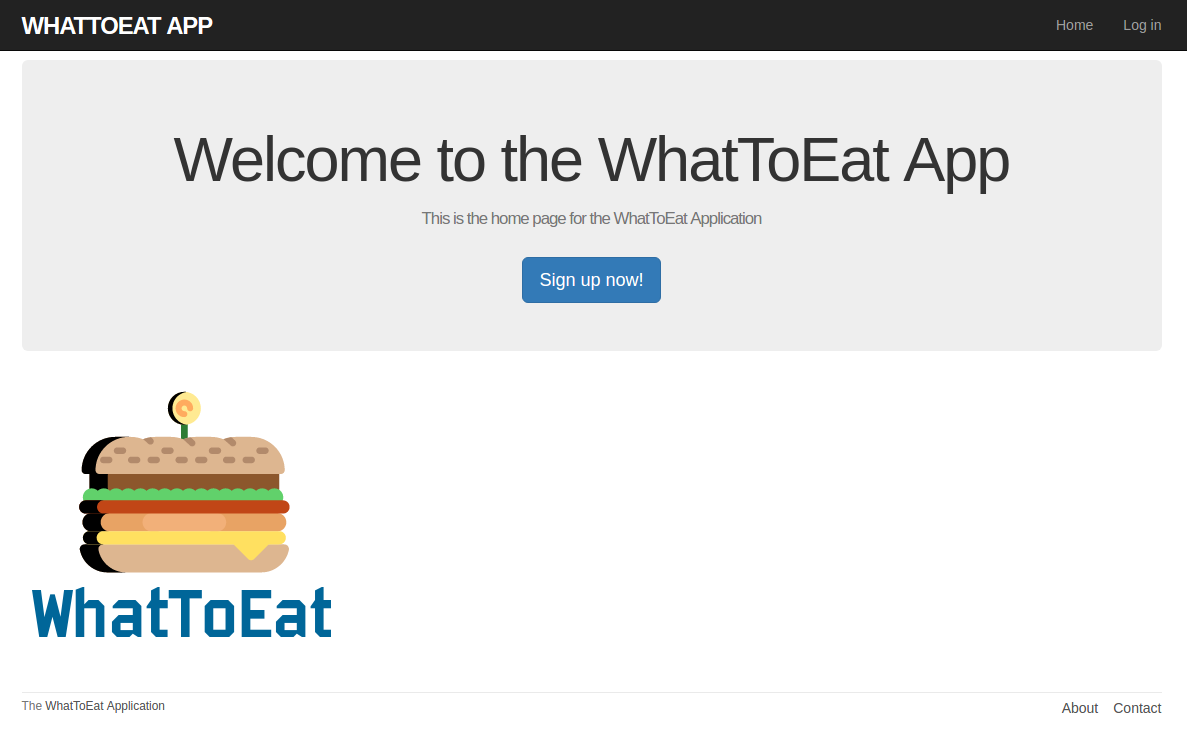


Fig.1.1. UI of the home page. Note that the design is subject to change in future iterations.

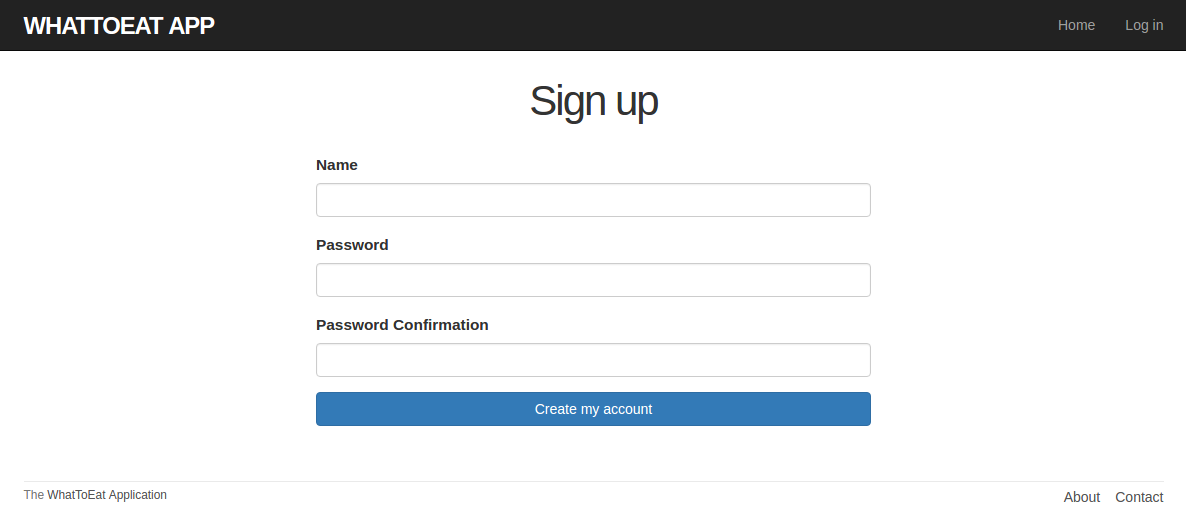


Fig.1.2. UI of the signup page.

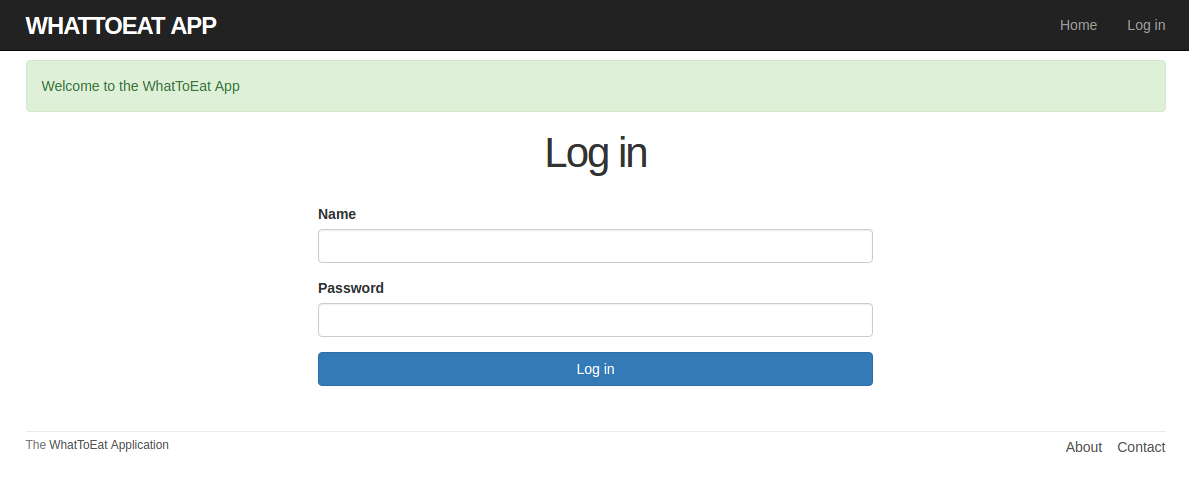


Fig.1.3. UI of the login page. After a user signs up, they will be redirected to this page.

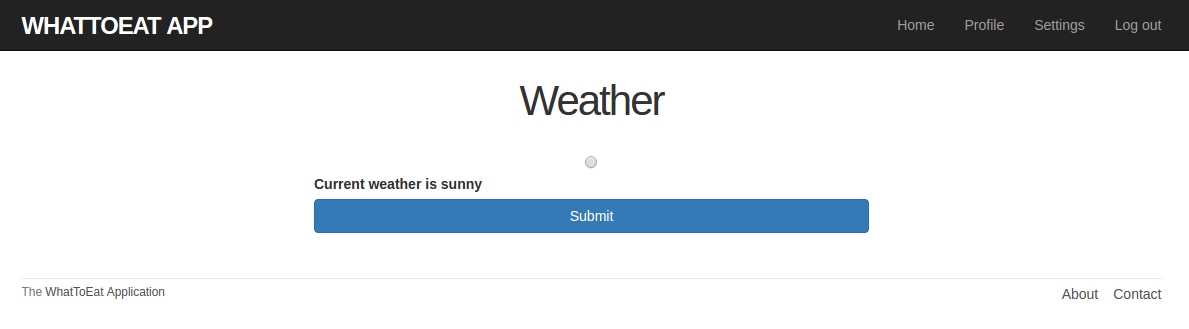


Fig.1.4. UI of the home page for users who are logged in. Note that the design is subject to change in future iterations.

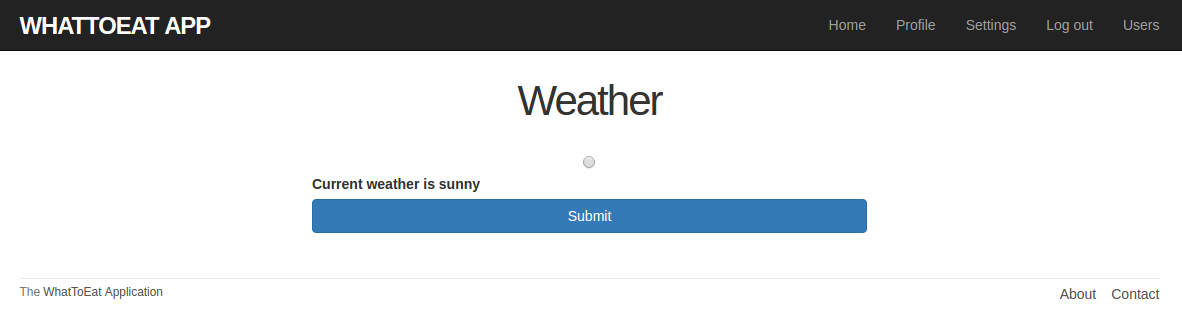


Fig.1.5. UI of the home page for admins who are logged in. Note the “Users” button on the top right corner of the UI. This button will be implemented in future iterations.

Future Iteration’s User Stories

* Once user logged in, bring to user homepage. The homepage displays weather.
* If user entered the wrong username or password, output messages that correctly represents the error.
* Once user logged in, user should not be able to sign up until the user signed out.
* The admin should be able to view and manage registered users using the “Users” button.
* The user can designate their location based on their city.