

Milestone Four

SW Engineering CSC648/848 Spring 2019 - Team 04
05/11/2020

SFStateEats

Team Members

Name	Email	Role
Rachit Joshi	rjoshi@mail.sfsu.edu	Team Lead
Samhita Brigeda	pandu.barigeda@gmail.com	Front End Lead
Pedro Souto	Pedro.Souto.SFSU@gmail.com	Back End Lead
John Pham	JohnPhamDeveloper@hotmail.com	Github Master
Vincent Tran	vtran6@mail.sfsu.edu	Database Lead
Khang Tran	ktran26@mail.sfsu.edu	Frontend Developer

Milestone 2 Version History

Milestone Version	Date
Milestone 4 Version 1	05/11/2020
Milestone 3 Version 2	04/30/2020
Milestone 3 Version 1	04/23/2020
Milestone 2 Version 2	04/09/2020
Milestone 2 Version 1	03/22/2020
Milestone 1 Version 2	03/10/2020
Milestone 1 Version 1	02/24/2020

Table of Contents

Section 1: Product Summary	3
Section 2: Usability Test Plan	6
Section 3: QA Test Plan	11
Section 4: Code Review	16
Section 5: Self-check: Best Practices for Security	20
Section 6: Self-check: Adherence to Original Non-functional Specs	24

Section 1: Product Summary

Name of Product

SFStateEats

Committed Functionalities

User

- New users shall be able to create an account.
- Users shall be able to login into their account.
- Users shall be able to rate restaurants on campus.
- Users shall be able to rate an event on campus.
- Users shall be able to write reviews.
- Users shall be able to view reviews.
- Users shall be able to see the menu for each restaurant.
- Users shall be able to see the price for each item on the menu.
- Users shall be able to see the address of the restaurant on the profile page.
- Users shall be able to see hours of business for each restaurant.
- Users shall be able to apply filters to their search.
- Users shall be able to flag a review as inappropriate for the application.
- Users shall be able to sort reviews by date.
- Users shall be able to sort reviews by popularity.
- Users shall be able to delete a review they have written.
- Users shall be able to see all reviews they have written.
- Users shall be able to see all reviews another user has written.
- Users will be able to see a visual representation of the ratings, which shall be displayed on a scale of 0-5 stars.
- Users shall be able to reply to other user's reviews. (Unique Feature)

Business

- New business owners shall be able to register a business account.
- Business owners shall be able to login into their account.
- Business owners shall be able to add their restaurant to the website.
- Business owners shall be able to create a menu for their restaurant.
- Business owners shall be able to upload photos of their restaurant.
- Business owners shall be able to write descriptions of their restaurant on the profile page.
- Business owners shall be able to change prices on the menu.

- Business owners shall be able to change hours of the restaurant.
- Business owners shall be able to flag a review as inappropriate for the website.
- Business owners shall be able to request removal of their restaurant from the website.
- Business owners shall be able to upload photos of each item on the menu.

Restaurant

- The restaurant profile page shall display photos.
- The restaurant profile page shall display ratings.
- The restaurant profile page shall display tags.
- The restaurant profile page shall display a description.
- The restaurant profile page shall display address.
- The restaurant profile page shall display a menu.
- The restaurant profile page shall display hours.
- The restaurant profile page shall display reviews.
- The restaurant profile page shall display the owner.
- The restaurant profile page shall display phone numbers.
- The restaurant profile page shall display the various payment methods the restaurant accepts.

System Admin

- System Administrators shall have the privilege to ban any users from the website for misuse.
- System Administrators shall have the privilege to delete restaurants from the platform.
- System Administrators shall have the privilege to change restaurant information.
- System Administrators shall have the privilege to delete reviews from the platform.

System

- The system shall show new restaurants on the main page.

Unique Features

The unique feature that our product offers is the ability to reply and leave comments to **reviews written by other users**. This will add an immense amount of user active time, since the review section of many restaurants will foster discussions. This is a great natural way to keep users on the website for longer.

This functionality also presents a benefit for the user. They are able to start discussions with someone who has left a review, and ask them to specify further what factors went into the decision of their review. For instance, if 'Alice' leaves a negative review of Taco Bell, 'Bob' is able to discuss this review with 'Alice' further. He may find that 'Alice' left a negative review due to the fact that her burger was cold. 'Bob' may then opt to go to the restaurants anyways, but not order a burger for himself.

URL to Product

<http://3.12.102.223:4000/>

Section 2: Usability Test Plan

Tasks

- 1) Write and view reviews
- 2) Reply to reviews
- 3) Apply filters to search
- 4) Add a restaurant
- 5) See hours of business for each restaurant

Test objectives:

In this test the users will test five main functions of the website. Users shall be able to write and view reviews, reply to reviews, apply filters to their search inquiry, add a new restaurant, and see hours of business for each restaurant. Writing and viewing reviews will be tested because that is one of the biggest functions in a website designated for food. Many users view reviews to see if they are interested in going to the restaurant or not. Next, we will test the function that will allow users to reply to other user's reviews. This will allow users to communicate about the restaurant and express their opinions. The third function that will be tested is the ability to apply filters to a search query because the users should be able to find the restaurants that match the specific needs of the user. Another function that we will be testing is to add a new restaurant. In this task the user will be able to add a new restaurant to the current list of restaurants, allowing other users to see information on this new restaurant. Finally, the last function to be tested is the hours of operation. Users should be able to clearly see the hours of operation by clicking the Hours tab on the profile page of each restaurant.

System Setup:

In the system setup, we clone the entire project from GitHub repository, set up the databases, and start the server. Next, we will provide each user with a laptop with the updated Chrome Browser to open the website and start navigating.

Starting Point:

The starting point for all users will be the homepage, where users will be able to search for restaurants, login into their account, or sign up for a new account. Then users will be able to navigate through the website to write reviews, view reviews, reply reviews, apply search filters, and see for hours of operation for a certain restaurant.

Intended Users:

The SfStateEats website's main target user groups are Sf state students and faculty. This website will help new incoming students to find affordable, delicious, and great restaurants on campus. Students and faculty members will be able to find restaurants based on health needs, payment options, and price.

User Satisfaction Evaluation:

To measure user satisfaction, we will use Likert tests to analyze exactly where we need to improve our website. Users will be able to choose from strongly disagree to strongly agree on each task that they are asked to perform.

Usability Task description:

<u>TASK</u>	<u>DESCRIPTION</u>
Task #1	View reviews of a restaurant and then write a review.
Machine State	User is logged in.
Successful Completion Criteria	Successfully writes a review and sees other reviews.
Benchmark	Completed in 1 min.

<u>TASK</u>	<u>DESCRIPTION</u>
Task #2	Reply to a review on a restaurant page.
Machine State	User is logged in.
Successful Completion Criteria	User clicks reply button and leaves a comment on another review
Benchmark	Completed in 1 min.

<u>TASK</u>	<u>DESCRIPTION</u>
Task #3	Apply filters to search results.
Machine State	User is not logged in.
Successful Completion Criteria	User successfully chooses all filter to their query
Benchmark	Completed in 30 sec.

<u>TASK</u>	<u>DESCRIPTION</u>
Task #4	Add a restaurant to the website.
Machine State	User logged into a business account.
Successful Completion Criteria	User successfully provides all details and presses the send button.
Benchmark	Completed in 30 sec.

<u>TASK</u>	<u>DESCRIPTION</u>
Task #5	View hours of operation of a restaurant.
Machine State	User is logged in.
Successful Completion Criteria	User successfully sees the hours of operation of restaurant
Benchmark	Completed in 45 sec.

- Questionnaire:

3 Likert scale questions were asked to the test subjects for evaluating user satisfaction.

S.No.	Task	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
1.	I am able to navigate through the website easily.			3	4	2
2.	I thought there was too much inconsistency in this system.		3	5	1	
3.	The process of writing a new review is efficient.		3	3	3	

Section 3: QA Test Plan

- **Test objectives:** Password shall contain at least 1 number, 1 special character, 1 uppercase letter, and 1 lowercase letter
- **HW and SW setup:** <http://3.12.102.223:4000/signup>
 - Hardware Setup:
 - macOS Catalina v.10.15.1
 - Macbook Pro (Retina, Mid 2012)
 - 2.3 GHz Quad-Core Intel Core i7
 - 8 GB 1600 MHz DDR3
 - NVIDIA GeForce GT 650M 1 GB
 - Begins when user enters every valid field except for password and then submits the form
- **Feature to be tested:** Signup
- **QA Test plan:**

Number	Description	Test Input	Expected Output	Pass/Fail
1	Password without special character	"Lake1"	error: "missing special character"	Fail
2	Password without upper case letter	"lake1@"	error: "missing upper case"	Fail
3	Password without number	"Lake@"	error: "missing number"	Fail
4	Password containing number, special character, uppercase letter, and lowercase letter	"Lake1@"	message: "user successfully registered"	Pass

- **Test objectives:** Images uploaded shall be in the format of jpg, jpeg, and png
- **HW and SW setup:** <http://3.12.102.223:4000/addrestaurant>
 - Hardware Setup:
 - macOS Cataline v.10.15.1
 - Macbook Pro (Retina, Mid 2012)
 - 2.3 GHz Quad-Core Intel Core i7
 - 8 GB 1600 MHz DDR3
 - NVIDIA GeForce GT 650M 1 GB
 - Begins when business owner enters the add restaurant page and attaches an image to their restaurant submission
- **Feature to be tested:** Image Upload
- **QA Test plan:**

Number	Description	Test Input	Expected Output	Pass/Fail
1	Uploading a GIF image	test.gif	error: "Image needs to be in jpg, jpeg, or png format"	Fail
2	Uploading a TIFF image	test.tiff	error: "Image needs to be in jpg, jpeg, or png format"	Fail
3	Uploading a png image	test.png	message: "image successfully uploaded"	Pass

- **Test objectives:** Username shall be stored as alphanumeric characters
- **HW and SW setup:** <http://3.12.102.223:4000/signup>
 - Hardware Setup:
 - macOS Cataline v.10.15.1
 - Macbook Pro (Retina, Mid 2012)
 - 2.3 GHz Quad-Core Intel Core i7
 - 8 GB 1600 MHz DDR3
 - NVIDIA GeForce GT 650M 1 GB
 - Begins when a user needs an account and navigates to the signup link to fill out the username field
- **Feature to be tested:** Signup
- **QA Test plan:**

Number	Description	Test Input	Expected Output	Pass/Fail
1	Storing username with an "@" symbol	"Bob@"	error: "username can only be stored as alphanumeric characters"	Fail
2	Storing username with an "*" symbol	"B*ob"	error: "username can only be stored as alphanumeric characters"	Fail
3	Storing username with only alphanumeric characters	"Billy123"	message: "user successfully registered"	Pass

- **Test objectives:** Only authorized users with bearer tokens shall be able to leave reviews
- **HW and SW setup:** <http://3.12.102.223:4000/review>
 - Hardware Setup:
 - macOS Cataline v.10.15.1
 - Macbook Pro (Retina, Mid 2012)
 - 2.3 GHz Quad-Core Intel Core i7
 - 8 GB 1600 MHz DDR3
 - NVIDIA GeForce GT 650M 1 GB
 - Begins when a user navigates to a restaurant page to create a review and submits the review for posting
- **Feature to be tested:** Review
- **QA Test plan:**

Number	Description	Test Input	Expected Output	Pass/Fail
1	Not logged in user trying to post review	{bearer: null}	error: "user is not logged in"	Fail
2	User trying to post review with an invalid bearer token (modified)	{bearer: "NotValidToken123"}	error: "user has an invalid bearer token"	Fail
3	User posting review with a valid bearer token	{bearer: "ValidToken123"}	message: "user authorized for this route"	Pass

- **Test objectives:** Password shall be stored as an encrypted value
- **HW and SW setup:** <http://3.12.102.223:4000/signup>
 - Hardware Setup:
 - macOS Cataline v.10.15.1
 - Macbook Pro (Retina, Mid 2012)
 - 2.3 GHz Quad-Core Intel Core i7
 - 8 GB 1600 MHz DDR3
 - NVIDIA GeForce GT 650M 1 GB
 - Begins when a user successfully signs up their account and information is send to backend to be encrypted
- **Feature to be tested:** Signup
- **QA Test plan:**

Number	Description	Test Input	Expected Output	Pass/Fail
1	Storing password as encrypted value	"lake"	"\$2y\$12\$j3G.cl13WHTKRjyt5MH7Ne8zsHQjqKI0HL7NEvOkNJmoEUXyYv./i"	Pass
2	Attempting to store password as original password	"lake"	"lake"	Fail
3	Storing password as encrypted value	"Lake!2@a"	"\$2y\$12\$um0OUbEFZYX5donYdI3GfeRZMIDgR9uziSbKnV3IXOKSnf6OPkTRW"	Pass

Section 4: Code Review

Coding Style

Bracket Indentation Convention

For our application, we choose to go with the "the one true brace style" (abbreviated as OTBS). This style places the first curly bracket in the same line as the function name, and the last curly bracket on the line after the last line of code. Some advantages of this style are that the starting brace needs no extra line alone; and the ending brace lines up with the statement it conceptually belongs to. (Source:

[https://en.wikipedia.org/wiki/Indentation_style#Variant:_1TBS_\(OTBS\)](https://en.wikipedia.org/wiki/Indentation_style#Variant:_1TBS_(OTBS)))

Variable Naming Convention

For our application, we choose to go with an underscore (`_`) delimiter for all variables. This means that when a variable consists of multiple words, each word is separated by an underscore. For instance, a variable that represents an account id would be called **`account_id`**.

Code Review Information

- Code Written By: Pedro Souto
- Code Reviewed By: John Pham
- Code Review Method: Email
- Code Description: This piece of code is written inside of a javascript file named 'account.controller.js'. This controller is responsible for all interactions with the account table inside our database. It implements functions which create, delete, and modify accounts.

This particular snippet, which is pasted below, was used as the basis for the code review.

Code Review Key

- **** Positive Comment****
- **** Negative Comment****

Code Reviewed Snippet

```
const bcrypt = require('bcrypt');
var jwt = require('jsonwebtoken');
const saltRounds = 10;
const {addUser, deleteUserByID} = require("./user.controller");

  **I like this, good idea!**
/**
 * These are the STRING representation for the column names in our database to
 * help prevent misspelling
 */
const TableColumnKey = {
  TABLE_NAME: "account",
  ACCCOUNT_ID: "account_id",
  USER_ID: "users_id",
  USERNAME: "username",
  PASSWORD: "password",
};

  **Good Header**

// ** REGISTER AN ACCOUNT **
// Request URL: http://localhost:3000/account/register
// Required key-value parameters in POST body: username, password, name, dob
const registerAccount = async (req, res) => {
  ** Extraneous Comment, Consider Removing**

  // Check to make sure all required parameters passed in
```

```

    if (req.body.username && req.body.password && req.body.name && req.body.dob)
    {
        // Before we can create an account, verify that the username is free
        if (await usernameIsFree(req) == 0) {
            console.log("Failed to register new account. Username taken.");
            res.status(400).send({ error: "Failed to register new account.
Username taken."});
            return;
        }
        **Good Comment, Explains Database Relationships**

        // An account needs a user as a FK. User stores the personal information
such as name, dob, and photo
        // First create a user entry in the user table, for this specific
account
        **Variable Name Follows Convention**

        var user_id = await addUser(req);

        // Check to make sure user was added successfully
        if (user_id != -1) {
            hashed_password = bcrypt.hashSync(req.body.password, saltRounds);
            try {
                ** Extraneous Comment, Consider Removing?**

                // Attempt to insert account
                const response = await req.DATABASE_CLIENT.query(`
                    INSERT INTO ${TableColumnKey.TABLE_NAME}
                    (${TableColumnKey.USER_ID}, ${TableColumnKey.USERNAME},
                    ${TableColumnKey.PASSWORD})
                    VALUES (${user_id}, '${req.body.username}',
                    '${hashed_password}')
                `);
                console.log("Successfully registered new account.");
                res.send({ message: "Successfully registered new account." });
                return;
            } catch (e) {
                // If account creation was unsuccessful, we should delete the
user we just made as well
                req.body.id = user_id;
                await deleteUserByID(req);
            }
        }
    }
}

```

```

        console.log(e);
        console.log("Failed to register new account.");
        res.status(500).send({ error: "Failed to register new
account."});
        return;
    }
} else {
    console.log("Failed to add new user before account.");
    res.status(500).send({ error: "Failed to add new user before
account."});
    return;
}
} else {
    console.log("Failed to register new account. Missing Parameters.");
    res.status(400).send({ error: "Failed to register new account. Missing
Parameters."});
    return;
}
};

```

Transcript of Email - Summary of Comments

Positives

- All variable names follow the agreed upon naming convention.
- All brackets follow the agreed upon indentation convention.
- Database object at the top of the file is very smart. It allows for quick change of names, without changing every instance of that table row inside all queries.
- Plenty of comments, especially in the confusing parts of the code.
- Functions have headers, which discusses both what the function does and what the function requires (parameters).

Negatives

- Although comments are great, there are a few comments that seem redundant and extra. Not every line needs a comment, only those sections or lines of code in which another programmer might be confused with.

Section 5: Self-check: Best Practices for Security

List of Major Assets being Protected

- User Password
 - User passwords are being encrypted using the Bcrypt blowfish hashing algorithm. This will ensure that even after a security breach, hackers would not be able to login to any accounts.
- Uploaded Images
 - All uploaded images will be stored in an AWS service called S3. S3 is a bucket to store anything from images, to videos, to files. We will use it to store all media on our website. This is being protected via permissions, which are set up inside the AWS console. Only whitelisted IP addresses are able to access these images.
- Database Information
 - The database is being stored in an AWS service called RDS. RDS is a Relational Database System which stores a variety of SQL databases. We are using this service to store our PostgreSQL instance. By hosting this instance on AWS, we are able to leverage their experience and security. Via the AWS console, we are able to set permissions and whitelist specific IP addresses. By doing this, we ensure that even during a security breach, the only entity able to access these records is our server, thus reducing the potential damage.

Confirmation of Password Encryption

All user passwords are being encrypted using the open source node library called '**bcrypt**'. Bcrypt uses a variant of the Blowfish encryption algorithm's keying schedule, and introduces a work factor, which allows you to determine how expensive the hash function will be. Because of this, bcrypt can keep up with Moore's law. As computers get faster you can increase the work factor and the hash will get slower (Source: <https://codahale.com/how-to-safely-store-a-password>)

Encrypt

Encrypt some text. The result shown will be a Bcrypt encrypted hash.

Rounds

`$2y$12$LSfB04WkHWDVMK6mRwMVE.h7mxNE4lxjL7VqZbkv/qBzzaQHVi3.G`

Above is an example of a password '**ThisIsMyPassword!**' which has been hashed using 12 salt rounds. A salt round determines how much work is needed to decrypt the password, and a common standard today is 12. Over time however, this number will need to increase due to computing capabilities increasing. The hash for this particular password is: '**\$2y\$12\$LSfB04WkHWDVMK6mRwMVE.h7mxNE4lxjL7VqZbkv/qBzzaQHVi3.G**'.

```
hashed_password = bcrypt.hashSync(req.body.password, saltRounds);
```

Above is the actual hashing implementation in our code. This particular piece of code is used in the register function. This line hashes the password, and uses the **hashed password** to create an account for the user which is stored in the database.

```
var validPassword = bcrypt.compareSync(req.body.password,  
    response.rows[0].password);
```

Above is the actual decryption implementation in our code. This particular piece of code is used in the login function. This line compares the password the user is entering, with the hashed password stored in the database. If the hash of the given password is equal to the stored hash in the database, then the password is valid. When the password is valid, this function returns a boolean of 'True', which allows for an easy comparison without **ever** storing the unhashed password of the user.

Confirmation of Input Validation

All inputs are first validated before being sent to an endpoint. This is done via a middleware. A middleware is a function that goes between an HTTP(S) endpoint, and the function behind that endpoint. This ensures that before any functions are called, all information being sent to that endpoint is first validated. This is beneficial in 2 ways. First, if a request is not valid due to improper input, the server does not need to waste resources executing commands, only to find out later that there was an error with the input. Secondly, it allows each endpoint to assume that all data is valid, which reduces copied code.

Account Details Validation

```
app.use((req, res, next) => {
  if (req.cookies['Token']) {
    var decoded = jwt.verify(req.cookies['Token'],
    'csc648CookiesSecret');
    req.account_id = decoded.account_id;
    console.log("Call made by user: ", req.account_id);
  }
  next();
});
```

The above is the middleware function responsible for validating tokens (which is a user input that corresponds to identification). Before **any** endpoint is called, this middleware first ensures that the cookie is valid. If a cookie is not valid, it stops execution of that endpoint, thus not wasting any resources. If a cookie is valid however, this endpoint attaches the `account_id` to the request. This allows **every** endpoint to assume that `account_id` is valid!

Endpoint Parameters Validation

```
const registerAccount = async (req, res) => {
  // Check to make sure all required parameters passed in
  if (req.body.username && req.body.password && req.body.name &&
  req.body.dob) {
    // CODE GOES HERE
  }
}
```










The above is a sample endpoint. Every endpoint implements similar checks, to ensure that all pieces of data it needs were sent in the HTTP call. If a user makes a request to an endpoint, but does not send the correct information, then that request is simply ignored and does not cause a waste of resources on the server.

Section 6: Self-check: Adherence to Original Non-functional Specs





Progress Key

-  DONE
-  ISSUE
-  ON TRACK





Security

-  Login shall be required to leave reviews.
-  Login shall be required to create a business listing.
-  A gateway service will redirect traffic to other services to prevent the discovery of the end URL.
-  Email shall not be from a temporary email host provider.
-  Passwords shall contain at least 1 number, 1 special character, 1 uppercase letter, and 1 lowercase letter.
-  Users shall be required to change password every 12 months.
-  Change of password shall require email verification.
-  Users shall have 5 login attempts before account lockout.
-  Account unlock shall require email verification before unlocking.

Audit

-  New business listings shall be approved by the system administrator.
-  Change in business owner shall be approved by the system administrator.
-  Flagged reviews shall be reviewed by system administrators.
-  Inappropriate reviews shall be removed by the system administrator.

Performance

-  Each service should be hosted in its own server to prevent overwhelming one server.
-  The web application should restart if it is abruptly shut down to prevent downtime.
-  System shall respond visually within 5 seconds.
-  Animations shall be kept to a minimum to accommodate for slow performing devices.

Data Integrity

- Images shall only be in the format of jpg, jpeg, and png.
- Images shall be saved on the server.
- Images uploaded shall be at most 2mb.
- Images shall be saved in the original size.
- Images shall be resized and displayed via CSS formatting, thus avoiding displaying and resizing after.
- Reviews shall not consist of special characters or emojis.
- Display name shall not consist of special characters or emojis.
- Email shall not consist of emojis.
- Passwords shall not consist of emojis.
- Databases shall be backed up every 24 hours.
- Databases shall be able to be backed up on command by a system administrator.

Compatibility

- The site shall be compatible for all mobile screen sizes.
- The site shall be compatible for all monitor screen sizes.
- The site shall be able to scale to high resolution (2K, 4K).
- The site shall be compatible for Safari on version 12.1.2.
- The site shall be compatible for Firefox on version 73.0.
- The site shall be compatible for Chrome on version 80.0.3987.106.


Conformance with Coding Standards

- The whole production cycle of the site shall be finished in at least 5 days before the delivery date.
- Development console logging shall be disabled for production.
- The DOM tree shall have meaningful semantics for every element.
- Components shall be created to allow for reusable code.

Look and Feel Standards

- The site shall have an interface that is intuitive to navigate at first glance.
- The site shall use clean fonts and colors.
- The site shall meet modern design standards.

Internalization / Localization Requirements

-  The site shall be internationalized in the English language

Web Site Policies

- The site shall not allow illegal content.
- The site shall not allow harassment of other users.
- The site shall not allow nudity.
- The site shall not allow business owners to review their own site.
- The site shall not allow self-promotion or malicious links.
- Reviews and replies are subject to the website's community guidelines.