SW Engineering - CSC648/848
Spring 2020

Unigator

Team 03
Milestone 4

Lionel Wong	lwong7@mail.sfsu.edu	Team Lead
Jorge Landaverde	jlandaverde@mail.sfsu.edu	Lead Backend Engineer
Gordon Lam	glam1@mail.sfsu.edu	Backend Engineer
Mitul Savani	msavani@mail.sfsu.edu	Lead Frontend Engineer/Git Master
Kevin Huang	khuang2@mail.sfsu.edu	Frontend Engineer
Jack Kower	jkower@mail.sfsu.edu	Frontend Engineer

Milestone	Date
Milestone04 Version02	5/20/2020
Milestone04 Version01	5/10/2020
Milestone03 Version02	5/8/2020
Milestone03 Version01	4/23/2020
Milestone02 Version02	4/15/2020
Milestone02 Version01	3/25/2020
MIlestone01 Version02	3/12/2020
Milestone01 Version01	3/5/2020

Table Of Contents

Product Summary

Unigator

1. San Francisco State University Students and Faculty are able to register for Unigator accounts using their school emails.
2. Users are only able to log in with their school associated email. The email needs to be verified before logging into Unigator.
3. Users on Unigator are able to search for available events by name.
4. Users are able to search through past events.
5. Users can also filter search results by Categories.
6. Users can view all information about an event.
7. Registered Users can log in or log out of their account.
8. Registered Users are able to save(Star) any event they are interested
9. Registered Users are able to RSVP to any event.
10. Registered Users are able to submit a “Create Event” request and that will be sent to an Administrator for approval. Information such as description, date, time, location and images are to be provided when creating an event.
11. Registered Users are able to become an Event Host.
12. Registered Users have their own unique profiles, which displays the user’s information.
13. Registered Users can edit their profiles.
14. Registered Users are able to view other user’s profiles.
15. Registered Users are able to view an Event Host’s email.
16. Registered Users can access the point shop and purchase customizable options by spending their accumulated points,
17. Registered Users can earn points by interacting with events on Unigator.
18. Administrators are able to view requests.
19. Administrators are able to approve or deny requests.

20. Administrators are able to ban a user.
21. Administrators are able to delete an event.
22. Administrators are able to send requests for changes to an Event Host.
23. Administrators have all options unlocked for the point shop.
24. Event Host shall be able to edit all of their event's information.
25. Event Host shall be able to delete their event.
26. Event Host are able to grant RSVPed users points.

Unigator aims to be a unique platform for anyone connected to San Francisco State University by uniting everyone together to enjoy and share their ideals through events. Any events concerning San Francisco State University can be found on Unigator and created to promote social engagement with the school and it's students. Unigator unites San Francisco State University peers through events and social engagements.

URL: http://13.52.231.107:3006/home

Usability Test Plan

Test Objective

The Unigator team would test for event search, event creation, applying purchased items on the point shop, saving an event and RSVPing to an event. Unigator aims to hold information on numerous events, thus the need for searching a specific event is crucial. Interaction and sociability is what makes up an event, making sure that the user has a way to create an event is vital in promoting socialization. Unigator boasts a unique system where users are able to customize their profile to their liking, with our point shop. Enabling/Disabling customizable options for a user's profile. RSVPing to an event gives the Event host a range of how many users might be showing up to their events.

Test Description

- System Setup
  - Unigator is deployed on Amazon Web Services
- Starting Point
  - Users should be on the Unigator home page
- Intended Users
  - SFSU students are the intended users.
- URL to be Tested
  - http://13.52.231.107:3006/home
- What is to be Measured
  - User Satisfaction

Usability Task Description

- Task 1: Searching for an event from the database by name
- Task 2: Creating an event and providing information
- Task 3: Applying changes to User's profile
- Task 4: Enabling/Disabling customizable options
- Task 5: RSVPing to an event

Test / Use Case	% Completed	Error	Comments
Search Event	100%	None	Searching for a specific event is working.
Create Event	100%	None	Currently duplicates of events can be created.
Purchasing customizable options for profile	100%	None	The functionality is there, now we just need it to be shown on the frontend
Enabling/Disabling customizable options	100%	Only one “type” of customizable option can be enabled.	Still trying to fix the errors at the moment
RSVPing to an Event	80%	None	Work in progress

Questionnaire

The process of searching for specific event(s) was simple and fast:

Strongly Agree Agree Neutral Disagree Strongly Disagree

The information pertaining to a specific event was very clear and easily identifiable:

Strongly Agree **Agree** Neutral Disagree Strongly Disagree

The process of creating an event(s) was fast and simple:

Strongly Agree Agree **Neutral** Disagree Strongly Disagree

The overall responsiveness of the website was fast and simple:

Strongly Agree **Agree** Neutral Disagree Strongly Disagree

The point shop was an intriguing way to customize the profile to one’s liking:

Strongly Agree Agree **Neutral** Disagree Strongly Disagree

QA Test Plan

Test Objective

As Unigator is a platform about events, the purpose of this test will primarily be focused on the events itself. The functionality that we will be testing are as follows: Event Creation, Event Editing, Event Deletion, Event Searching and Event Response (RSVP).

Hardware and Software Setup

Amazon Web Services instance is currently running Unigator. React.js for the frontend and Node.js on the backend with MySQL database. Google Chrome would be the default browser to be tested on.

Feature to be Tested

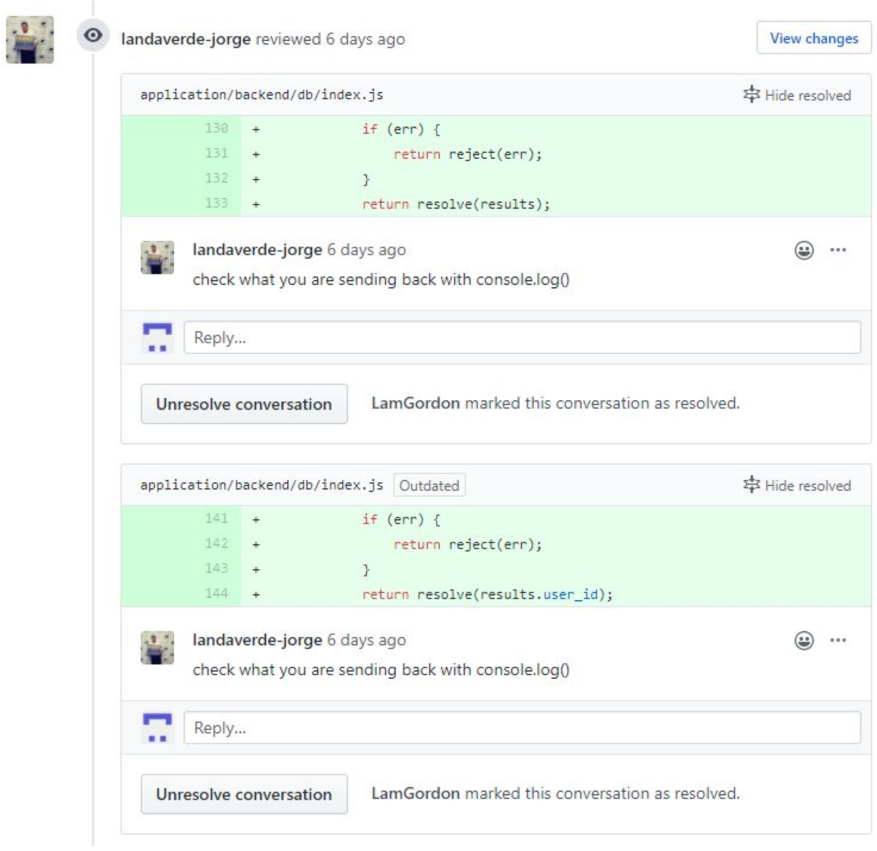
- Event Creation
- Event Editing
- Event Deletion
- Event Searching
- Event Response (RSVP)

Test#	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Event Creation	Registered User creates an event.	To create an event, the following info is needed: Name, Description, Location, Date, Time, and optionally an image.	Event Created and searchable from the search bar and is in the database.	**Pass**
2	Event Editing	Event Host editing information on their hosted event.	Goes to the event details page of the created event, and edit information on the event, such as the time and location. Then update it.	Able to change information pertaining to the event and saved.	**Fail**

3	Event Deletion	Event Host deleting their event.	Goes to the event details and deletes the event. Which would also delete it from the database.	The event is non-searchable anymore, since it’s deleted from the database.	**Half-Pass**
4	Event Searching	Registered User searches for an event.	Searches for “Mobile”, for any event which has “Mobile” in it on the search bar.	The specific event searched for popped up on the results, and clicking on it will lead to the Event details page.	**Pass**
5	Event Response (RSVP)	Registered User RSVPing to an event.	A registered user will RSVP to an event on the event details page.	The user that RSVPed to an event is visible under the RSVP list on the event details page.	**Half-Pass**

Code Review

We conducted our code review through GitHub comments in the pull requests. In this specific instance, Gordon Lam was having his code reviewed by Jorge Landaverde. Since each of us have our own coding styles, our code varies a lot, especially compared with front-end and back-end, the coding styles vary.



landaverde-jorge reviewed 6 days ago

application/backend/db/index.js

```
+            if (err) {
+                return reject(err);
+            }
+            return resolve(results);
```

landaverde-jorge 6 days ago
check what you are sending back with console.log()

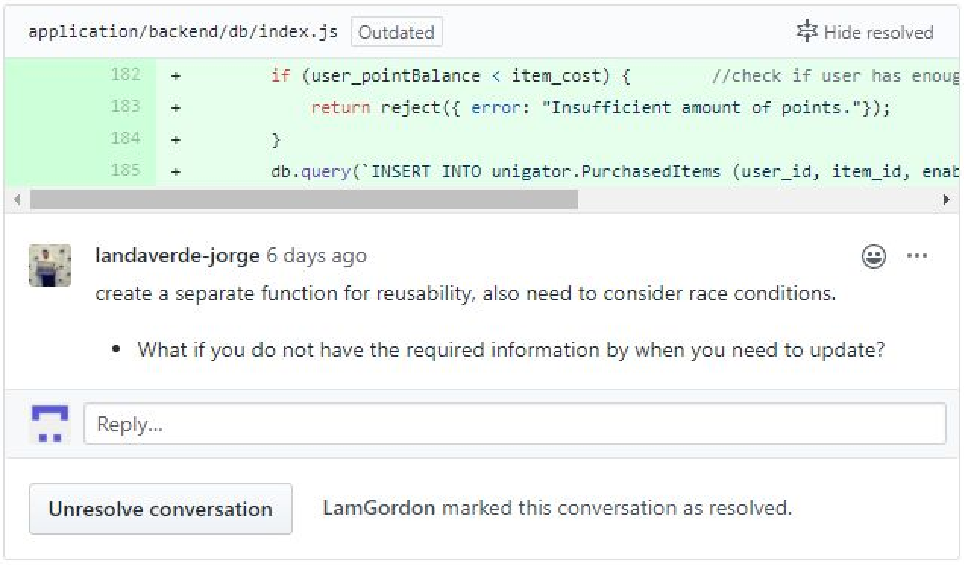
LamGordon marked this conversation as resolved.

application/backend/db/index.js Outdated

```
+            if (err) {
+                return reject(err);
+            }
+            return resolve(results.user_id);
```

landaverde-jorge 6 days ago
check what you are sending back with console.log()

LamGordon marked this conversation as resolved.



application/backend/db/index.js Outdated

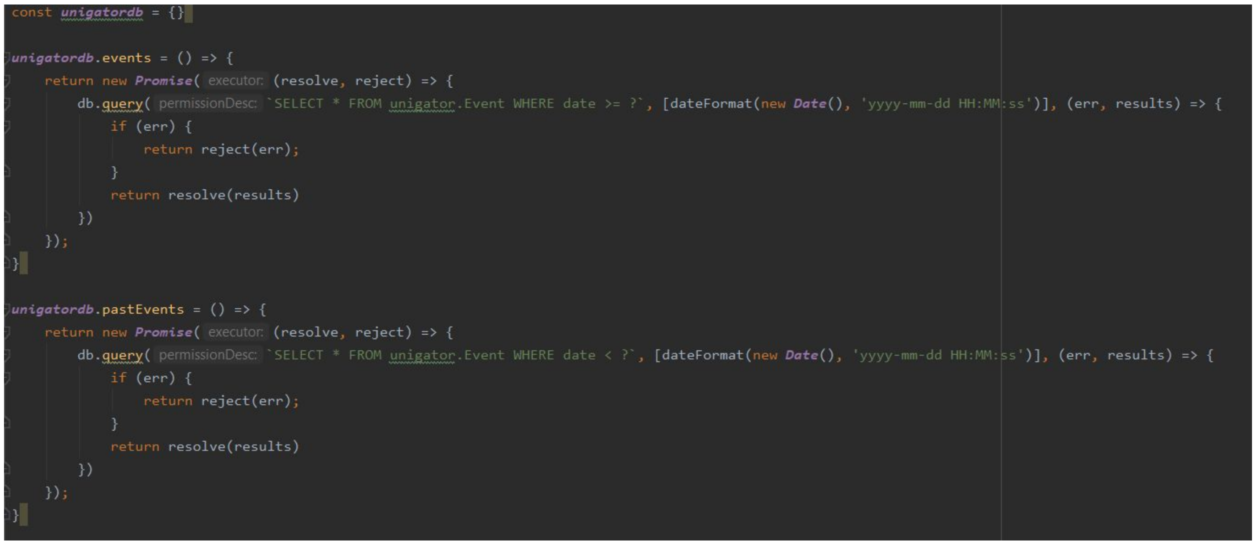
```
+         if (user_pointBalance < item_cost) {        //check if user has enoug
+             return reject({ error: "Insufficient amount of points."});
+         }
+         db.query(`INSERT INTO unigator.PurchasedItems (user_id, item_id, enab
```

landaverde-jorge 6 days ago
create a separate function for reusability, also need to consider race conditions.

- What if you do not have the required information by when you need to update?

LamGordon marked this conversation as resolved.

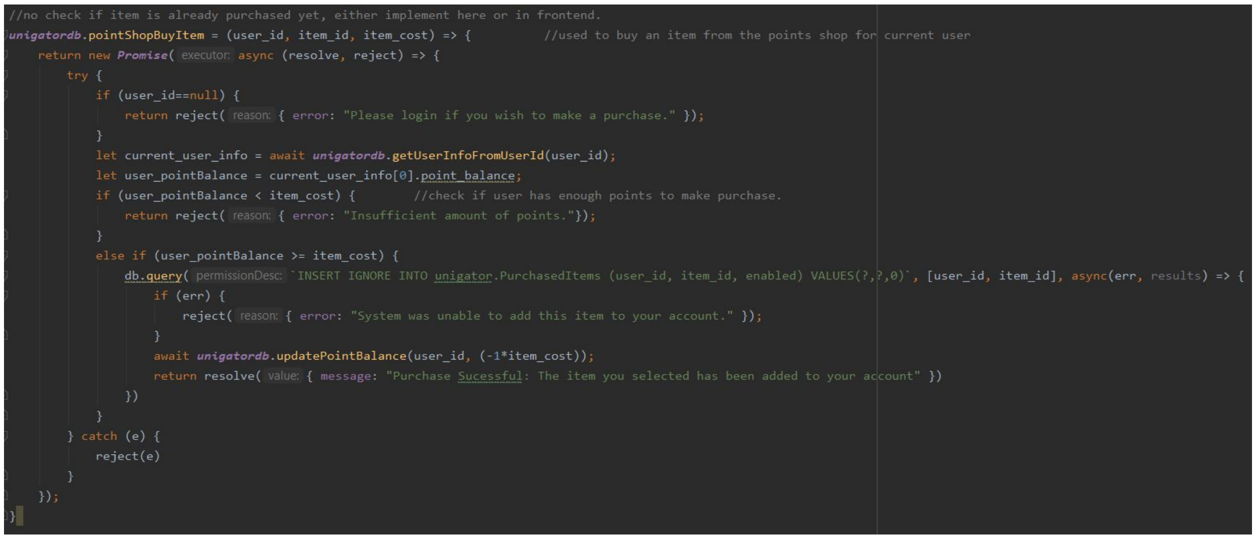
This Is just an example of our code, the code needs to further documented by having more comments.



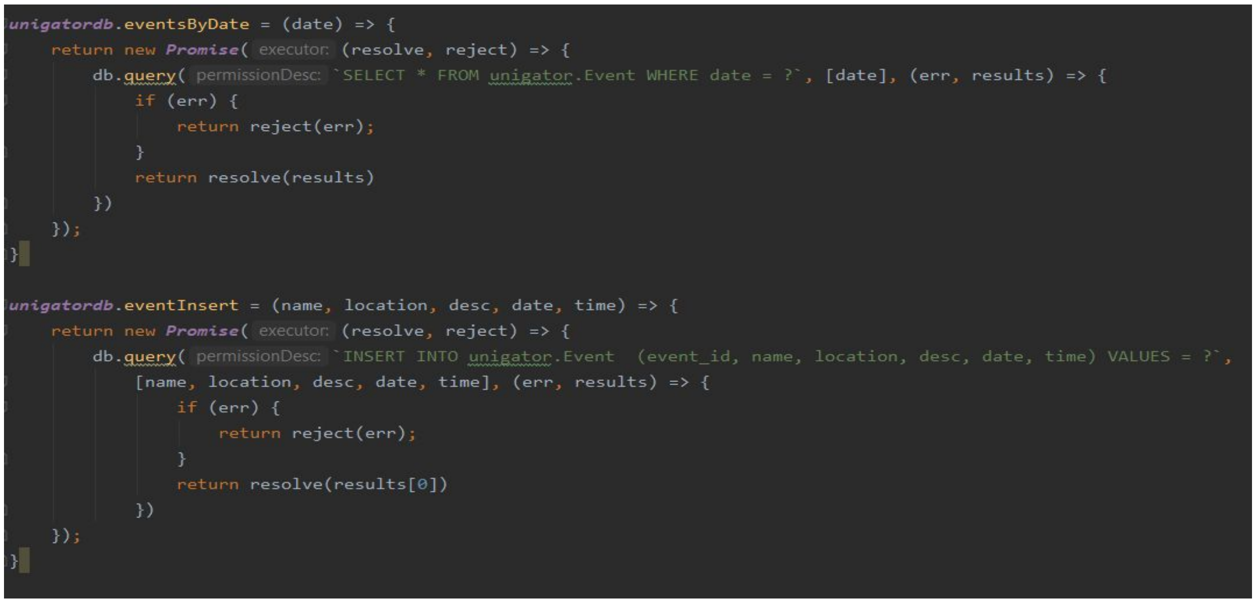
```
const unigatordb = {}

unigatordb.events = () => {
    return new Promise( executor: (resolve, reject) => {
        db.query( permissionDesc: `SELECT * FROM unigator.Event WHERE date >= ?`, [dateFormat(new Date(), 'yyyy-mm-dd HH:MM:ss')], (err, results) => {
            if (err) {
                return reject(err);
            }
            return resolve(results)
        })
    });
}

unigatordb.pastEvents = () => {
    return new Promise( executor: (resolve, reject) => {
        db.query( permissionDesc: `SELECT * FROM unigator.Event WHERE date < ?`, [dateFormat(new Date(), 'yyyy-mm-dd HH:MM:ss')], (err, results) => {
            if (err) {
                return reject(err);
            }
            return resolve(results)
        })
    });
}
```



```
//no check if item is already purchased yet, either implement here or in frontend.
unigatordb.pointShopBuyItem = (user_id, item_id, item_cost) => {          //used to buy an item from the points shop for current user
    return new Promise( executor: async (resolve, reject) => {
        try {
            if (user_id==null) {
                return reject( reason: { error: "Please login if you wish to make a purchase." });
            }
            let current_user_info = await unigatordb.getUserInfoFromUserId(user_id);
            let user_pointBalance = current_user_info[0].point_balance;
            if (user_pointBalance < item_cost) {        //check if user has enough points to make purchase.
                return reject( reason: { error: "Insufficient amount of points."});
            }
            else if (user_pointBalance >= item_cost) {
                db.query( permissionDesc: `INSERT IGNORE INTO unigator.PurchasedItems (user_id, item_id, enabled) VALUES(?,?,0)`, [user_id, item_id], async(err, results) => {
                    if (err) {
                        reject( reason: { error: "System was unable to add this item to your account." });
                    }
                    await unigatordb.updatePointBalance(user_id, (-1*item_cost));
                    return resolve( value: { message: "Purchase Sucessful: The item you selected has been added to your account" })
                })
            }
        } catch (e) {
            reject(e)
        }
    });
}
```

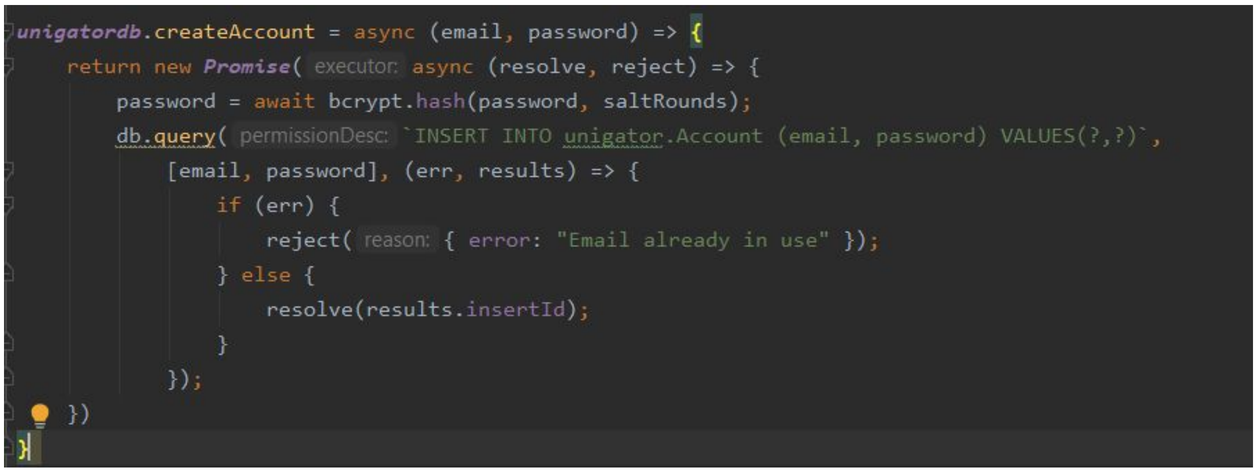


```
unigatordb.eventsByDate = (date) => {
    return new Promise( executor: (resolve, reject) => {
        db.query( permissionDesc: `SELECT * FROM unigator.Event WHERE date = ?`, [date], (err, results) => {
            if (err) {
                return reject(err);
            }
            return resolve(results)
        })
    });
}

unigatordb.eventInsert = (name, location, desc, date, time) => {
    return new Promise( executor: (resolve, reject) => {
        db.query( permissionDesc: `INSERT INTO unigator.Event  (event_id, name, location, desc, date, time) VALUES = ?`,
            [name, location, desc, date, time], (err, results) => {
                if (err) {
                    return reject(err);
                }
                return resolve(results[0])
            })
    });
}
```

Self-check on Best Practices for Security

The major assets that Unigator will be protecting are our user’s password, email addresses. Sensitive information such as those are kept in the database and has encryption. The bcrypt node package allows us to encrypt our user’s password, so any sensitive information would be kept confidential. We plan to only allow logged in users to view Event Host’s information.



```
unigatordb.createAccount = async (email, password) => {
    return new Promise( executor: async (resolve, reject) => {
        password = await bcrypt.hash(password, saltRounds);
        db.query( permissionDesc: `INSERT INTO unigator.Account (email, password) VALUES(?,?)`,
            [email, password], (err, results) => {
                if (err) {
                    reject( reason: { error: "Email already in use" });
                } else {
                    resolve(results.insertId);
                }
            });
    })
}
```

Self-check on Adherence to Original Non-Functional Specs

1) Security:
    a) Can only create an event page if you are logged into your account. (DONE)
    b) You can only create an account with a SFSU email.(DONE)
    c) Able to request change of password.(ON TRACK)
    d) Passwords will be encrypted.(DONE)
    e) Have password requirements for creating an account and resetting password.(ON TRACK
2) Audit:
    a) The user may request access to create an account without SFSU email.(ON TRACK)
    b) The administrator may delete any event from the event page.(ON TRACK)
    c) The administrator may view all RSVP accounts of any event.(DONE)
    d) The administrator may view and approve any event requests.(ON TRACK)
    e) New events must be approved by the administrator.(ON TRACK)
3) Response Time:
    a) The initial load up time on the site shall fall under 5 seconds.(DONE)
    b) Retrieving data from the database shall be fast and efficient.(DONE)
4) Capacity:
    a) The application shall be able to hold at least 50 events within the span of two weeks. (DONE)
    b) The application shall save a log of past events within 2 months.(ISSUE)
    c) The application shall be capable of handling accounts of at least all the SFSU students. (DONE)
    d) The application shall be made in a way that it’s scalable, in case of and further feature updates are planned.(DONE)
5) Reliability:
    a) Downtime maintenance shall be done within 5 hours.(DONE)
    b) Downtime maintenance shall restrict users from using the application to prevent any false information.(DONE)
    c) The application shall not crash under any circumstances.(DONE)
    d) Registered Users shall be informed of maintenance via an announcement on the main page, or through email.(ON TRACK)
6) Recovery:

   a) If any severe problems arise in the application, the application shall be down for repairs.(DONE)
   b) Application downtime shall not exceed 24 hours. (DONE)
   c) If the application is down, Registered User data and Events data shall be safe.(DONE)

7) Data Integrity:
   a) All Data shall be backed up every 72 hours. (ON TRACK)
   b) The users shall be prompted when they wish to delete or edit data.(ON TRACK)
   c) Any purchases in the point shop shall be saved with the Registered User's account.(DONE)
   d) Account recovery can be requested to the Administrator.(ON TRACK)
   e) Image sizes shall not exceed 4 megabyte.(DONE)
   f) Only image format of JPG, JPEG and PNG shall be accepted.(DONE)
8) Platform:
   a) The application shall be compatible with Chrome browser version 80.0.3987.132 and earlier.(DONE)
   b) The application shall be compatible with Firefox browser version 74.0 and earlier.(DONE)
   c) The application shall be compatible with Safari browser version 13.0 and earlier.(DONE)
   d) The application shall scale correctly with the screen size of the device its on. (ISSUE)
   e) The application shall be compatible with any client OS that supports Chrome, Firefox, or Safari browsers. (DONE)
   f) The application shall be able to account for any compatibility issues as a result of browser or OS updates. (ON TRACK)
9) Coding Standards:
   a) Comments should accompany the code in order to understand functionality.(DONE)
   b) Errors shall not disrupt the application in a way that is detrimental.(DONE)
   c) Code needs to be reviewed and tested by all members of the group before going into the master branch.(DONE)
   d) Internal and external error shall be recorded in a log for future reference.(ON TRACK)
   e) Any error or bugs that affect the user should be notified to them through their email.(ON TRACK)

f) Error debugging shall be done without interfering with the application's functionality.(ON TRACK)
g) All priority one requirements shall be implemented and working before launching the application.(ON TRACK)

10) Look and Feel Standards:
    a) The application should follow the aesthetics of San Francisco State University's site.(ON TRACK)
    b) Registered Users profiles are unique and customizable by the user.(DONE)
    c) The application shall appear simple in order to avoid any confusion to the users.(ON TRACK)
    d) Registered Users needs shall be prioritized over Unregistered Users.(DONE)
    e) The application should be automatically resized based on the screen size of the device.(ON TRACK)
    f) Have a loading toggle wheel.(ON TRACK)
    g) Have Links to Unigator social media in footer. (ON TRACK)
    h) Follow the color scheme of SFSU with purple and gold.(DONE)
11) Internalization / Localization:
    a) English shall be the default language of the application.(DONE)
    b) The application shall only allow SFSU students to register accounts.(ON TRACK)
    c) Non-SFSU students shall be able to make requests.(ON TRACK)
12) Website Policies:
    a) The application's policies shall be visible via a link on the homepage.(ON TRACK)
    b) The application shall not ask for any sensitive information.(DONE)
    c) Registered User info shall be kept confidential.(DONE)
    d) A verification of the email shall be done in the register section.(ON TRACK)
    e) Users shall understand and read through all the policies and abide by it when using the application.(ON TRACK)