SW Engineering CSC648-848-05 Spring2024

GATOR CONNECT

TEAM 03

Student	Full Name	SFSU Email	Role
#1	Hoang-Anh Tran	htran31@sfsu.edu	Team-lead
#2	Ralph Quiambao	rquiambao@sfsu.edu	Frontend-lead
#3	Karma Gyalpo	kgyalpo@sfsu.edu	Backend-lead
#4	Dustin Meza	dmeza2@sfsu.edu	Database-admin
#5	Fabian Weiland	fweiland@sfsu.edu	Github-master
#6	Jeawan Jang	jjang3@sfsu.edu	Docs-editor

MILESTONE 2

Date: 03/04/2024

History Table

Milestone	Version	Date Submitted
Milestone 1	M1V1	3/1/2024
	M1V2	4/4/2024
Milestone 2	M2V1	4/4/2024
	M2V2	N/A

TABLE OF CONTENTS

MILESTONE 2 ..... **1**
History Table 1
TABLE OF CONTENTS **2**
I. DATA DEFINITIONS 3
II. PRIORITIZED FUNCTIONAL REQUIREMENTS 5
Priority 1 (Critical) 5
Priority 2 (Important) 7
Priority 3 (Opportunistic) 8
III. UI MOCKUPS AND STORYBOARDS (high level only) 9
UI Mockups 9
Storyboards 12
IV. HIGH LEVEL DB ARCHITECTURE & ORGANIZATION 20
Database Requirements 20
Define the DBMS 22
Database Organization 22
a. The Entities, Attributes, Relationship, and Domains at the high level 22
b. An Entity Relationship Diagram (ERD) 25
c. An Entity Establishment Relationship Diagram (EER) 26
d. Forward engineering the database model 27
Media Storage 35
V. HIGH LEVEL APIs & MAIN ALGORITHMS 36
High Level APIs 36
Non-Trivial Algorithms 37
New changed SW tools and frameworks 37
VI. SYSTEM DESIGN 38
System Diagram 38
Summary of the components of your system architecture 39
UML Class diagrams 40
Summary 41
VII. HIGH LEVEL APP NETWORK & DEPLOYMENT DESIGN 42
Application Networks Diagram 42
High Level Deployment Diagram 43
VIII. IDENTIFY ACTUAL KEY RISKS 44
IX. PROJECT MANAGEMENT 45
X. DETAILED LIST OF CONTRIBUTIONS 46

I. DATA DEFINITIONS

1. User
    - A user can be categorized into two types:
        - Registered users: Users can create an account using their SFSU email.
        - Non-registered users: Users can only see the register and login pages.
    - Registered users can be:
        - Student: Users who are enrolled in a course at the university. Students can access all functionalities of the application.
        - Professor: Users who teach at the university.
2. Account
    - Each user can only create one account, using a unique school email address and username.

3. Profile
    - Avatar: Size will at least be restricted to a 128x128 thumbnail.
    - Biography: A brief description that the user will add containing information pertaining to their role at SFSU.
    - Number of Friends
    - Profile Feed: Only posts that are created by the user will be displayed on their page.

4. Message
    - Noteworthy Attributes:
        - Message Type → either public or private
        - Message Status → sent, received or read
    - Private messages are between two users only.
    - Public messages are posted on channels that many users can view and interact with.

5. Like
    - Users can react using a like on a user's post or message.

6. Comment
    - Users can comment on posts or messages inside the channels.

7. Channel:
    - A designated space where many users can post messages.
    - A specialized chat room that is focused on a specific topic.

8. Notification
    - Notification shall be sent for new messages, likes, friend requests and comments through the use

9. Friend
    - A friend can be contacted by users through a friend request.
    - A friend can have a status of either accepted, pending or denied.

10. Post:
    - A post will contain text content by the author, a timestamp, and number of likes and comments.
    - The author's username and avatar will be displayed on the top left of the post.
    - Attributes: (post_id), post_description, post_time, num_likes, num_comments")

11. Rate My Professor (RMP): [PRIORITY 2]
    - A section where users can provide reviews and ratings for a professor/lecturer.

12. Resource:
    - This provides campus-related information and services for students to use.
    - Ratings and reviews will be available for each resource listed.

13. Transportation
    - Transportation is provided under the resource section, including helpful information on buslines that will take you to and from campus.

14. Food Vendor
    - Food Vendor is a helpful resource that contains the many options a user has to choose from on, or nearby campus.

15. Events:
    - Under the resource section, a user is able to organize an event that many can take part in and enjoy.
    - Displays activities, gatherings, or occasions that are hosted within the university.
    - Includes the event time, location and description.

16. Groups: [PRIORITY 2]
    - An organized community that users can create and join. These are typically centered around a specific interest or affiliation.

17. Report: [PRIORITY 2]
    - This is a way for users to flag abusive or inappropriate behavior within the application.

II. PRIORITIZED FUNCTIONAL REQUIREMENTS

Priority 1 (Critical)

1. User
    - A user shall have an account
    - A user is a student, professor or friend
    - A user shall be able to send messages
    - A user shall be able to make posts
    - A user shall be able to push likes
    - A user shall be able to write comments
    - A user shall be able to send friend requests to have friends
    - A user shall be able to receive notifications
    - A user shall be able to access resources

2. Account
    - An account shall belong to a user
    - An account shall have a profile
    - An account shall be registered with a SFSU email

3. Profile
    - A profile shall be associated with an account
    - A profile shall be able to set an avatar (profile picture)
    - A profile shall be able to show user's full name
    - A profile shall show the number of friends
    - A profile shall contain user's personal posts

4. Student
    - A student is a type of user
    - A student shall set detailed information like major
    - A student shall be able to plan events

5. Professor
    - A professor is a type of user
    - A professor shall set detailed information like department

6. Message
    - A message shall be sent by a user
    - A message is either private or public

7. Private message
    - A private message is a message
    - A private message shall consists of one-to-one like user-to-user

8. Public message
    - A public message is a message
    - A public chat room shall have channels

9. Channel
    - A channel shall be associated with public message
    - A channel shall be study help, find a roommate, general, campus club, or campus event

10. Notification
    - A notification shall be sent for new friend requests to the user

11. Friend
    - A friend shall be able to accept or decline a friend request
    - A friend shall be contacted by a user through a friend request

12. Post
    - A post shall be created by a user
    - A post shall have comments
    - A post shall have likes
    - A post shall have username and user's avatar
    - A post shall have content in text format

13. Like
    - A like shall belong to a post
    - A like shall be canceled

14. Comment
    - A comment shall be created by a user
    - A comment shall belong to a post

15. Resource
    - A resource shall have a search bar
    - A resource shall contain options such as food vendors, campus events, and transportation

16. Food Vendor
    - A food vendor shall be belong to resource
    - A food vendor shall have the menu (attribute)
    - A food vendor shall have ratings
    - A food vendor shall have reviews
    - A food vendor shall have description
    - A food vendor shall have a PDF map (attribute)

17. Transportation
    - Transportation shall be belong to resource
    - Transportation shall be regarding various buses, munies and shuttles in SFSU
    - Transportation shall have a PDF map (attribute)

18. Event
    - An event shall belong to a resource
    - An event shall be promoted by students.
    - An event shall have specific date, time, and location.
    - An event shall have description
    - An event shall have a PDF map (attribute)

Priority 2 (Important)

1. Student
    - A student shall set detailed information with minor
    - A student shall be able to rate and comment on Rate My Professor (RMP)

2. Message
    - A message shall be deleted
    - A message shall be liked by a user

3. Channel
    - A channel shall be sports interest or study group

4. Notification
    - A notification shall be able to announce who liked user's post
    - A notification shall be able to announce who commented user's post
    - A notification shall be able to announce new messages
    - A notification shall be sent for upcoming booked appointments or reservations

5. Post
    - A post shall be shared
    - A post shall be deleted
    - A post shall have content in image format

6. Resource
    - A resource shall contain Rate My Professor (RMP) option

7. Transportation
    - Transportation shall show the detailed schedules of diverse transportations
    - Transportation shall contain pricing information

8. Rate My Professor (RMP)
    - A RMP shall be embedded in resource
    - A RMP shall have professor's name
    - A RMP shall have ratings

Priority 3 (Opportunistic)

1. Channel
    - A channel shall be created by a user

2. Notification
    - A notification shall be turned on/off by user

3. Resource
    - A resource shall contain room reservation option
    - A resource shall contain find roommate option

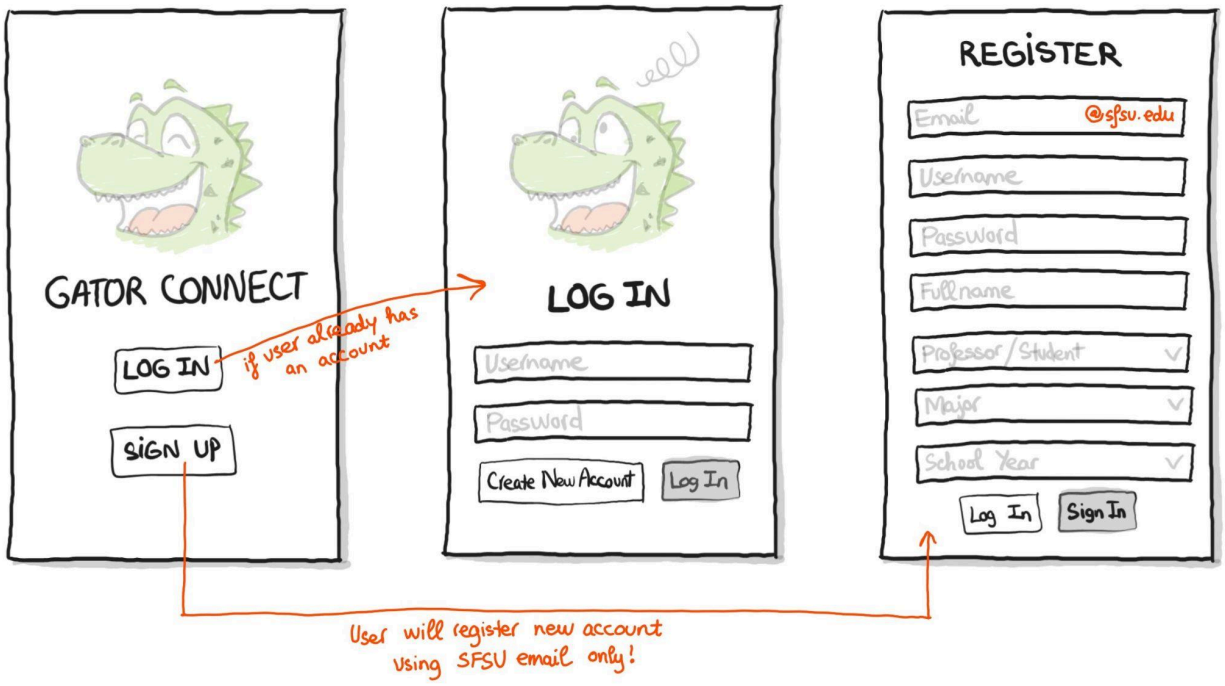
4. Room Reservation
    - A room reservation shall be belong to a resource
    - A room reservation shall be for a study room
    - A room reservation shall be for a gym/workout room

5. Find roommate
    - Find roommate shall be belong to a resource
    - Find roommate shall have posts
    - Find roommate shall only be accessed by students

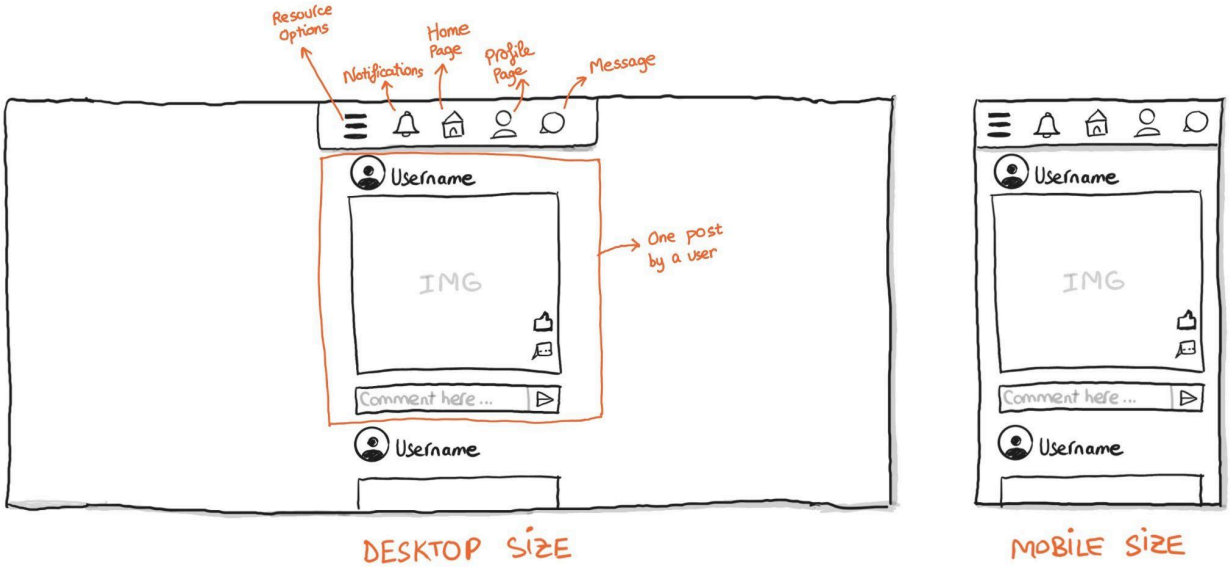
III. UI MOCKUPS AND STORYBOARDS (high level only)

UI Mockups

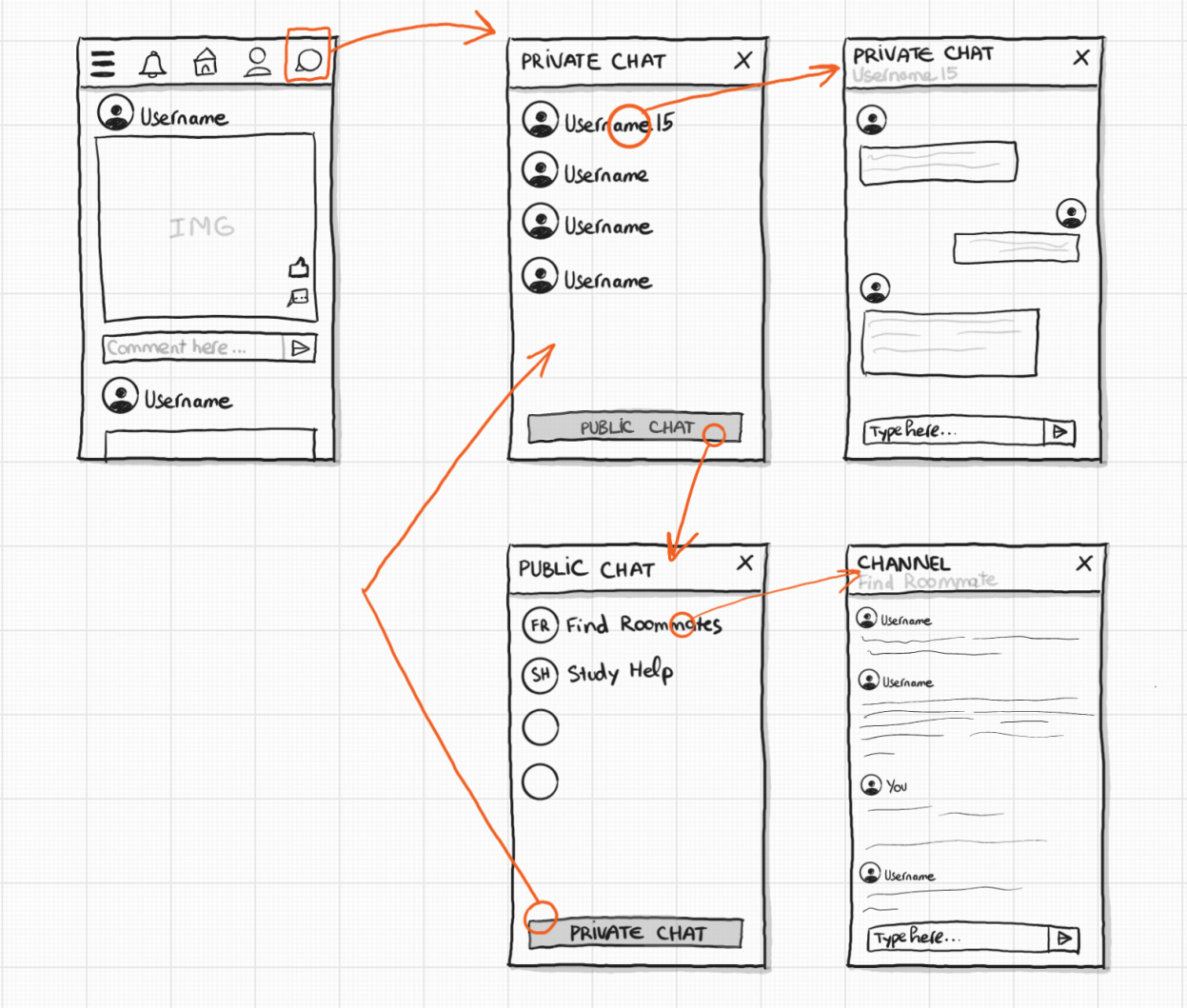
1. Authentication Pages



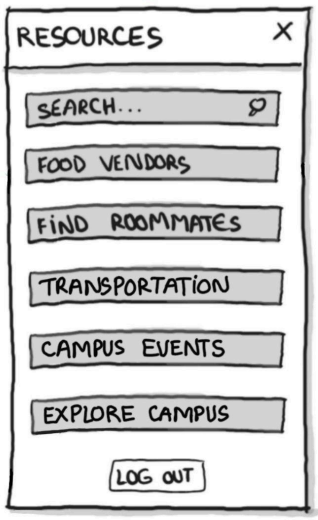
2. Home Page

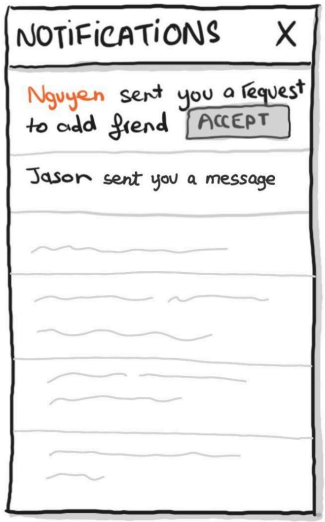


3. Chat Pages

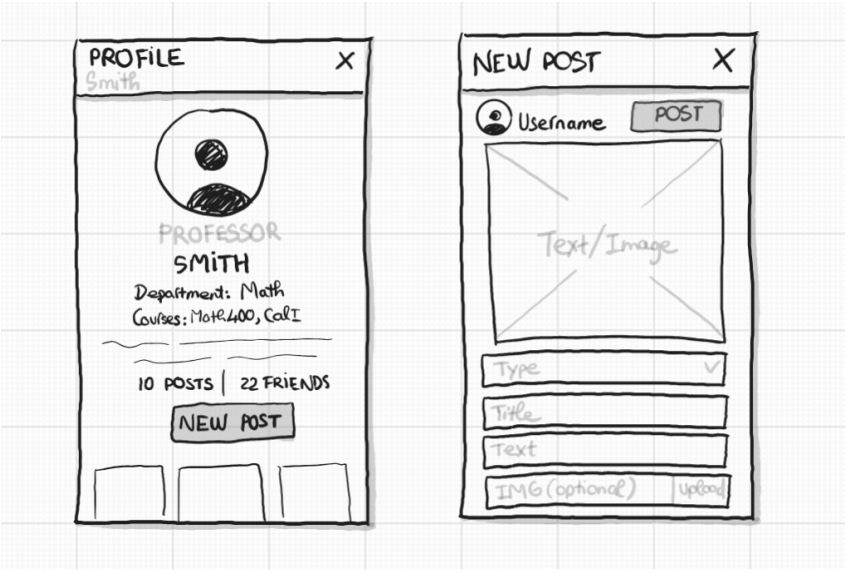


4. Resources & Notifications Pages



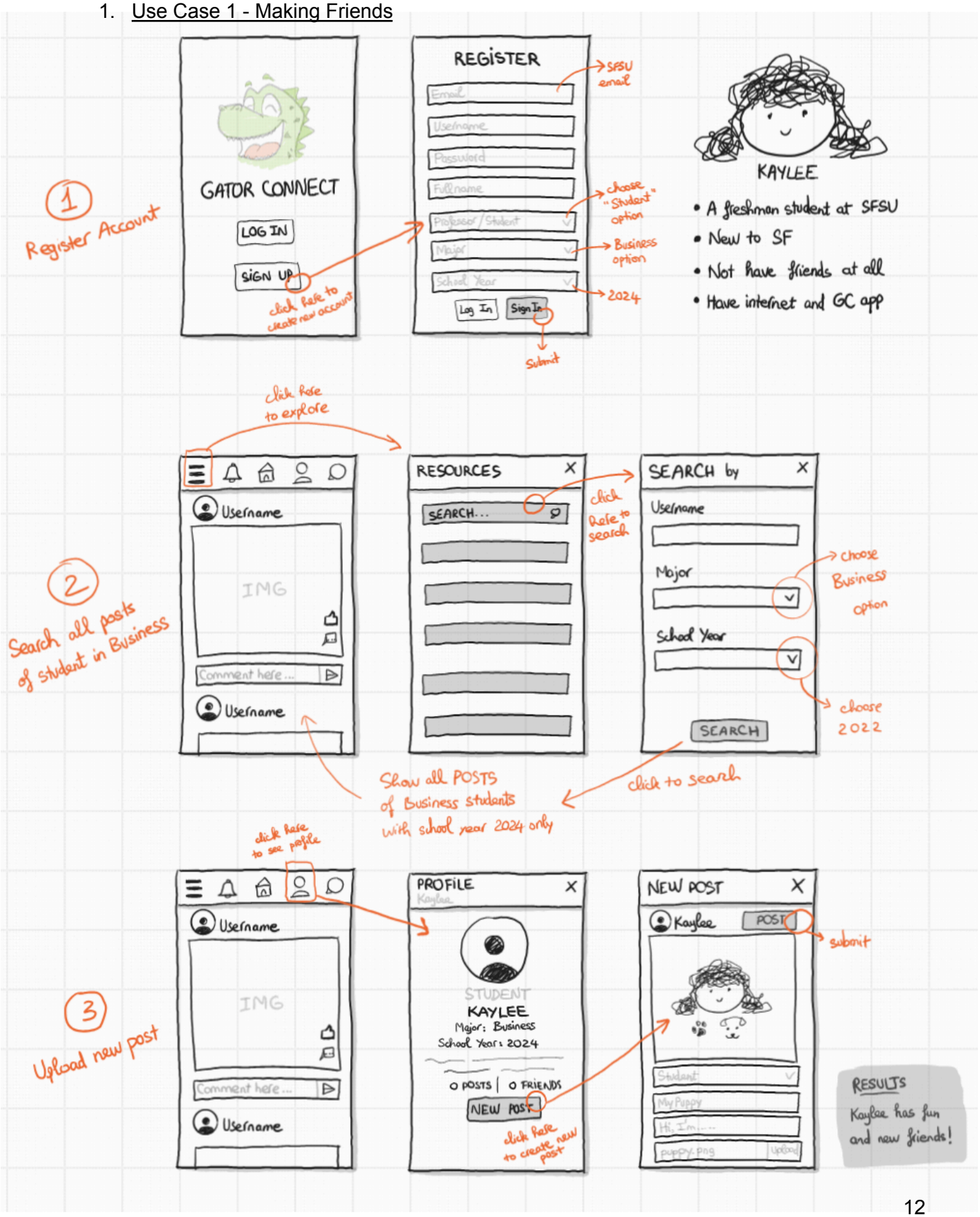


5. Profile & Add-New-Post Pages

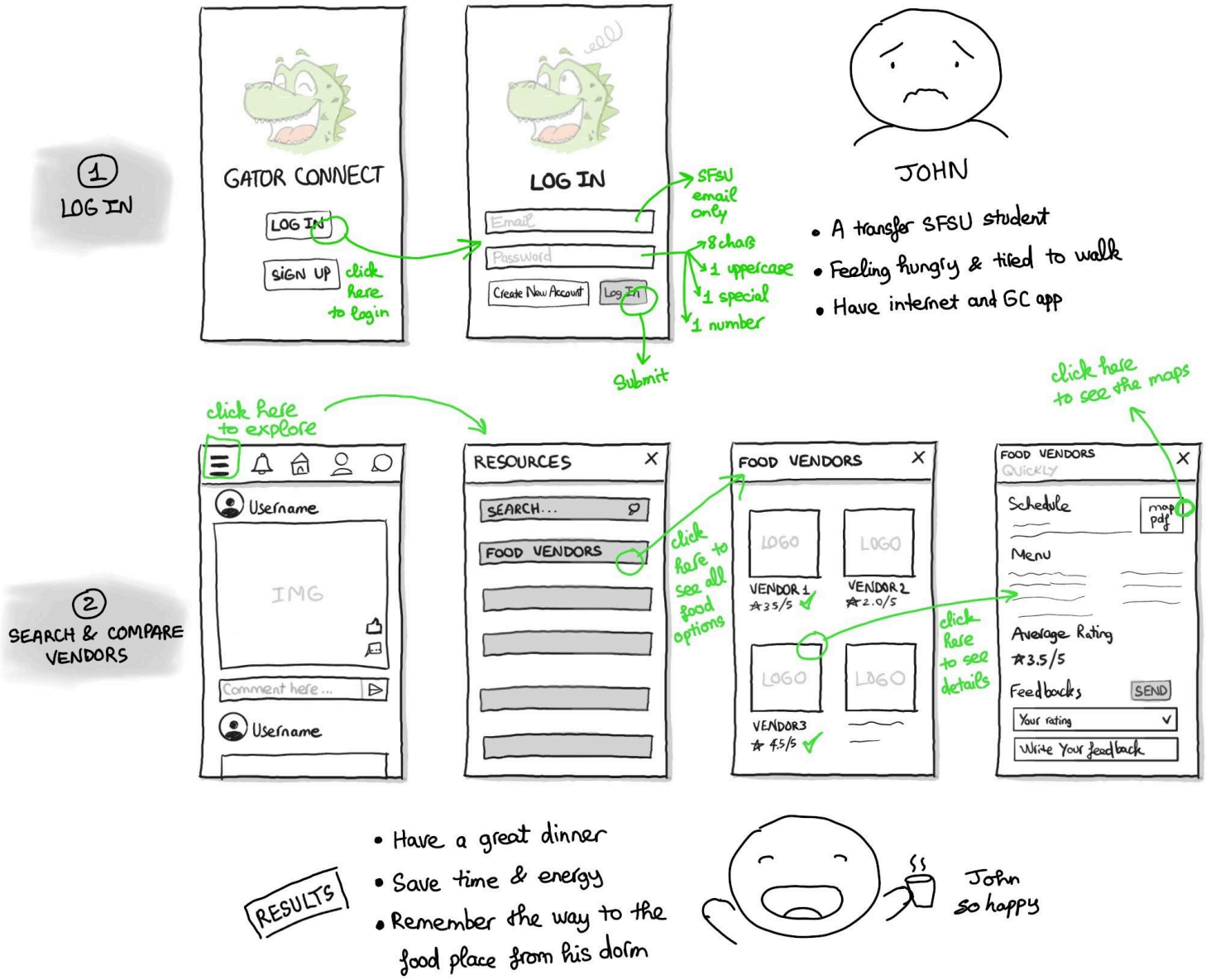


Storyboards

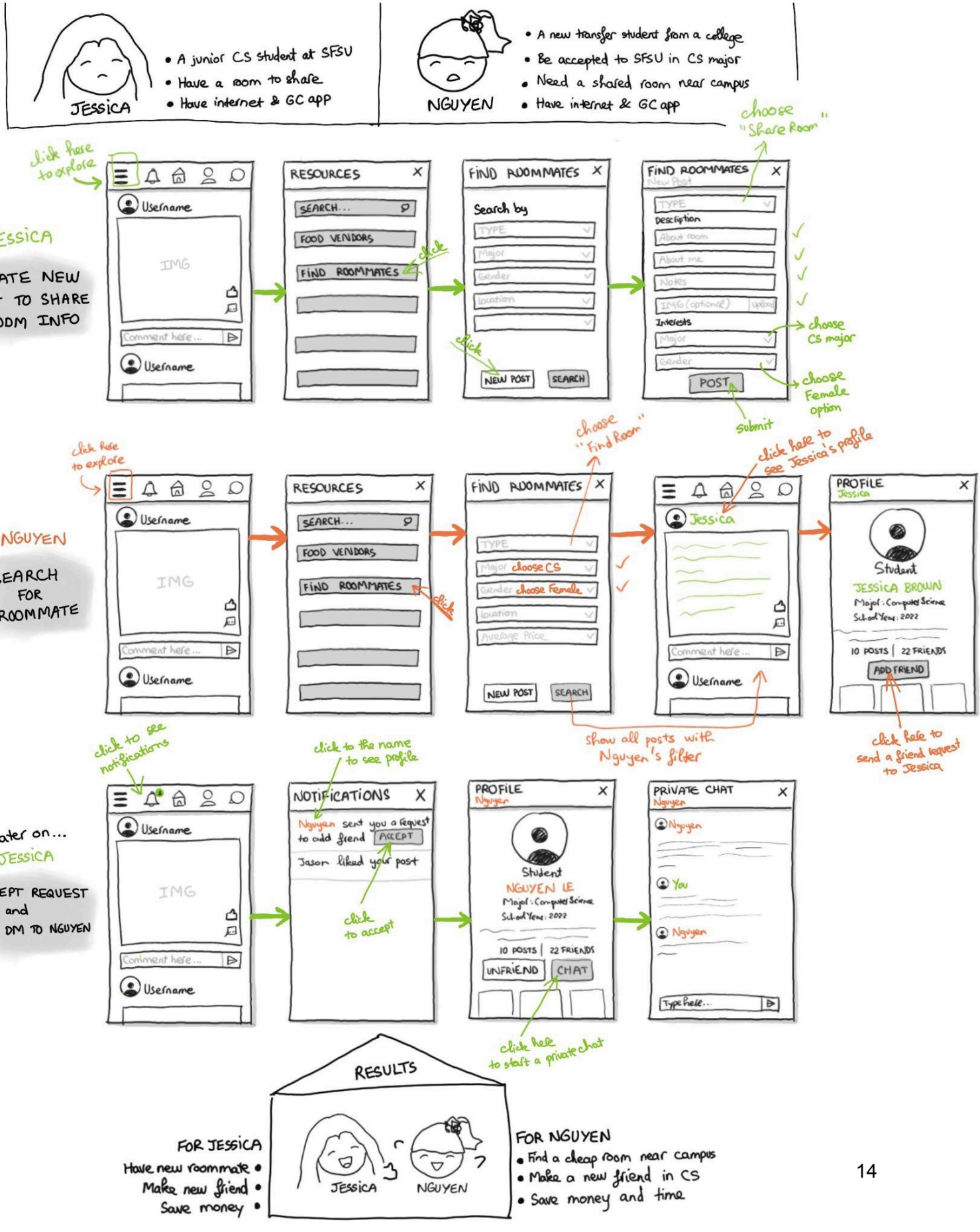
1. Use Case 1 - Making Friends



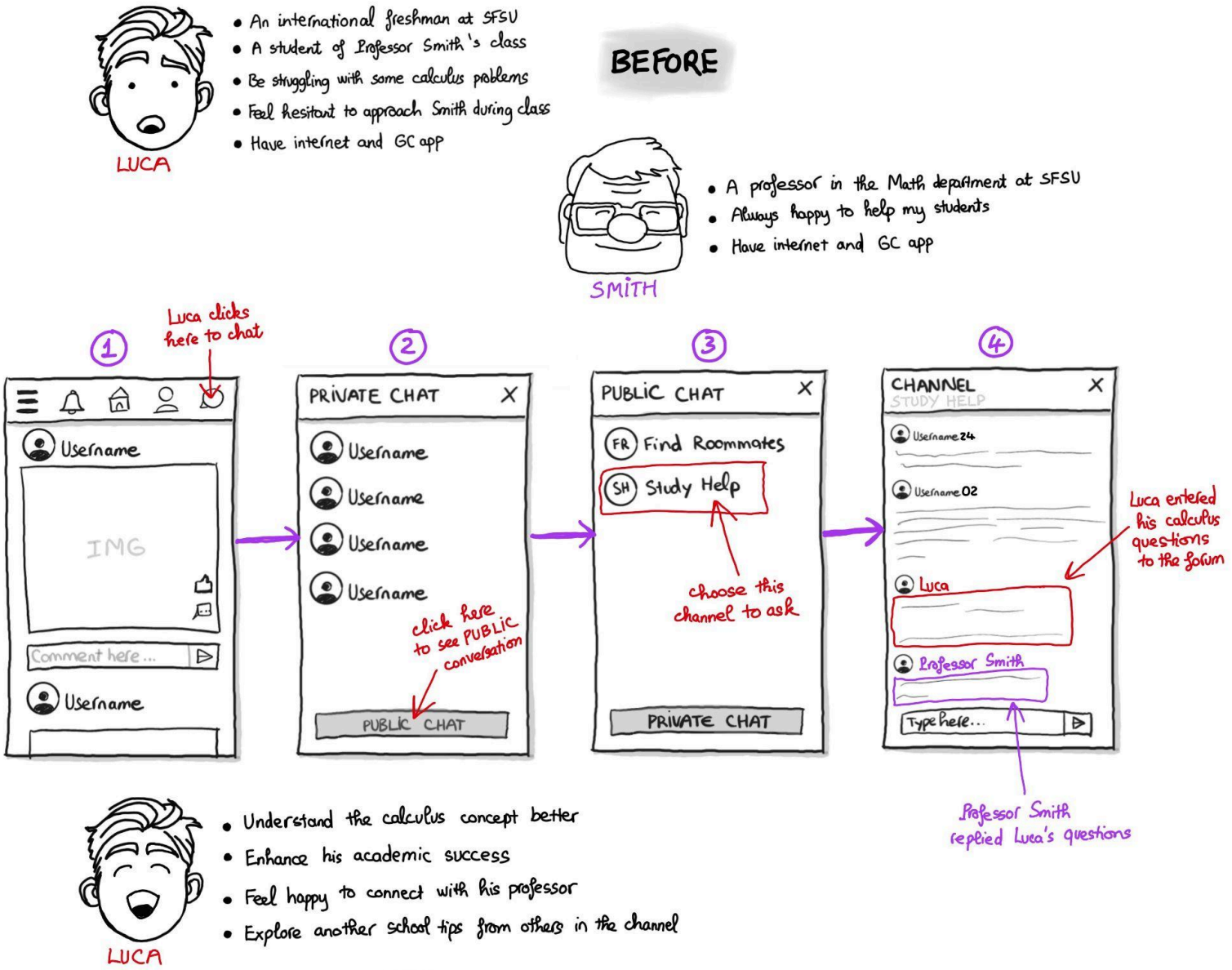
2. Use Case 2 - Cafeteria Search



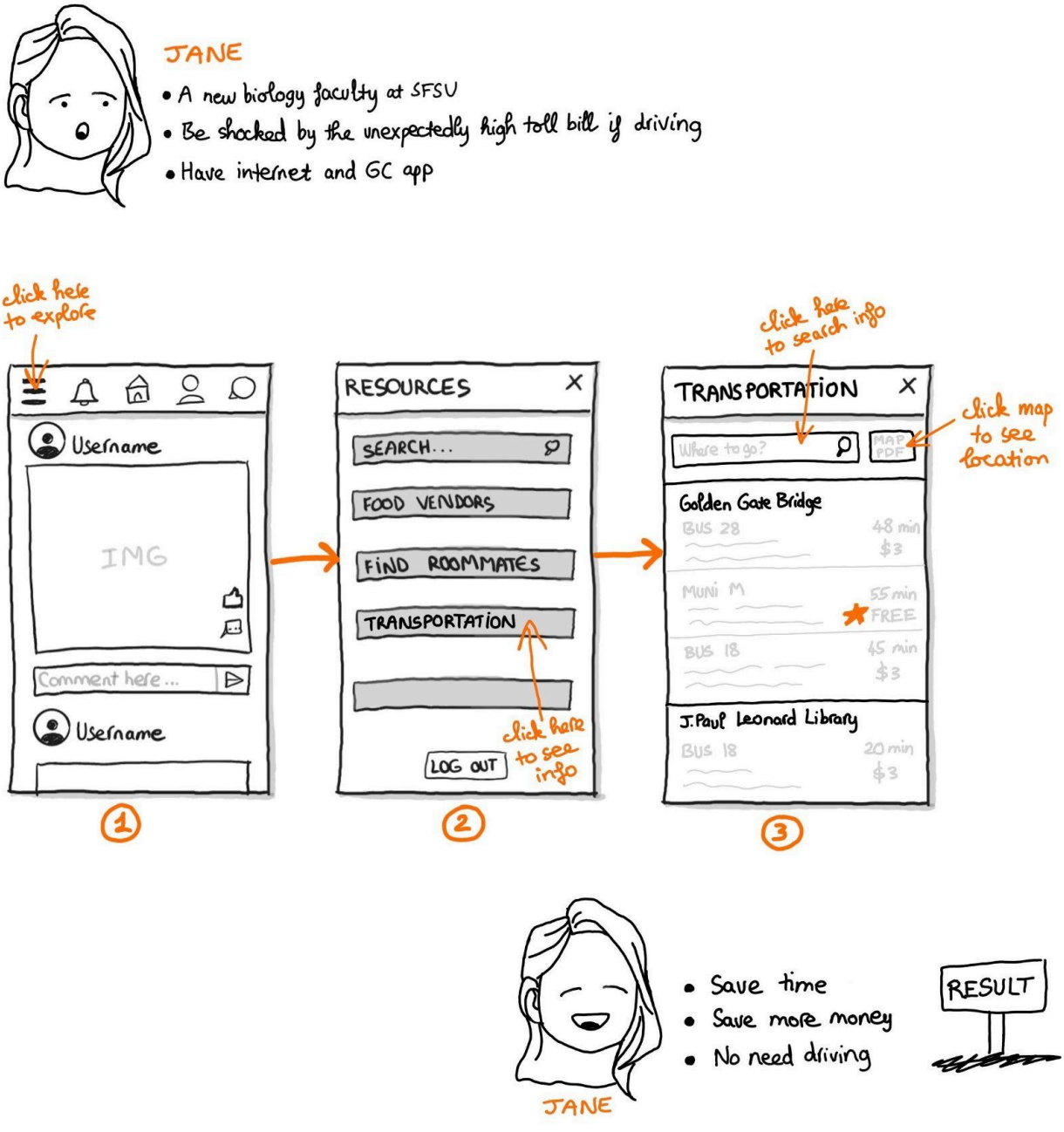
3. Use Case 3 - Finding Roommates (Priority 3)



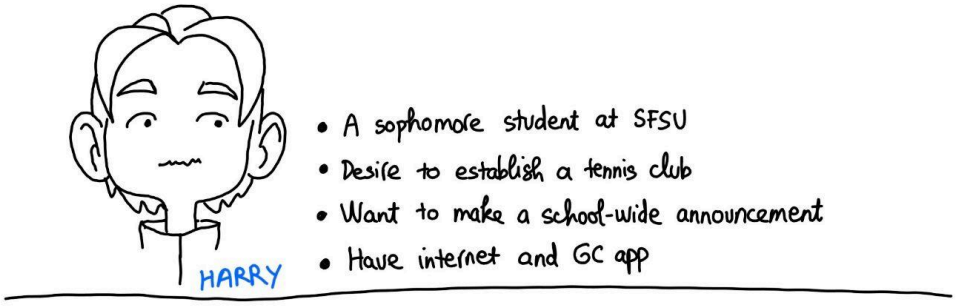
4. Use Case 4 - School Tips

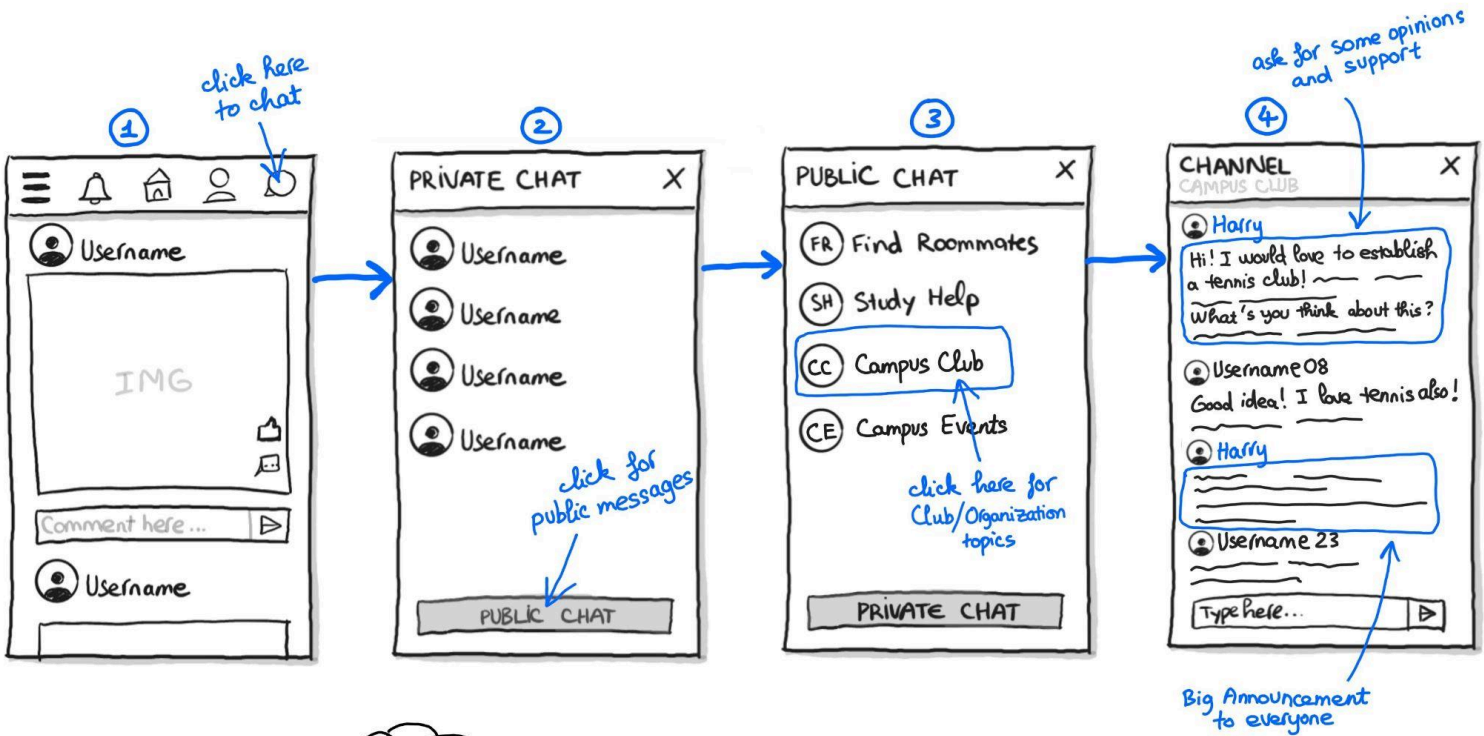


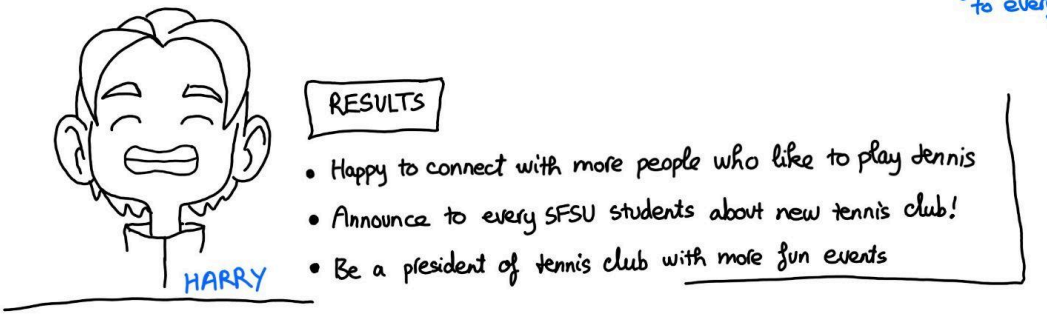
5. Use Case 5 - Bus Routes and Shuttles



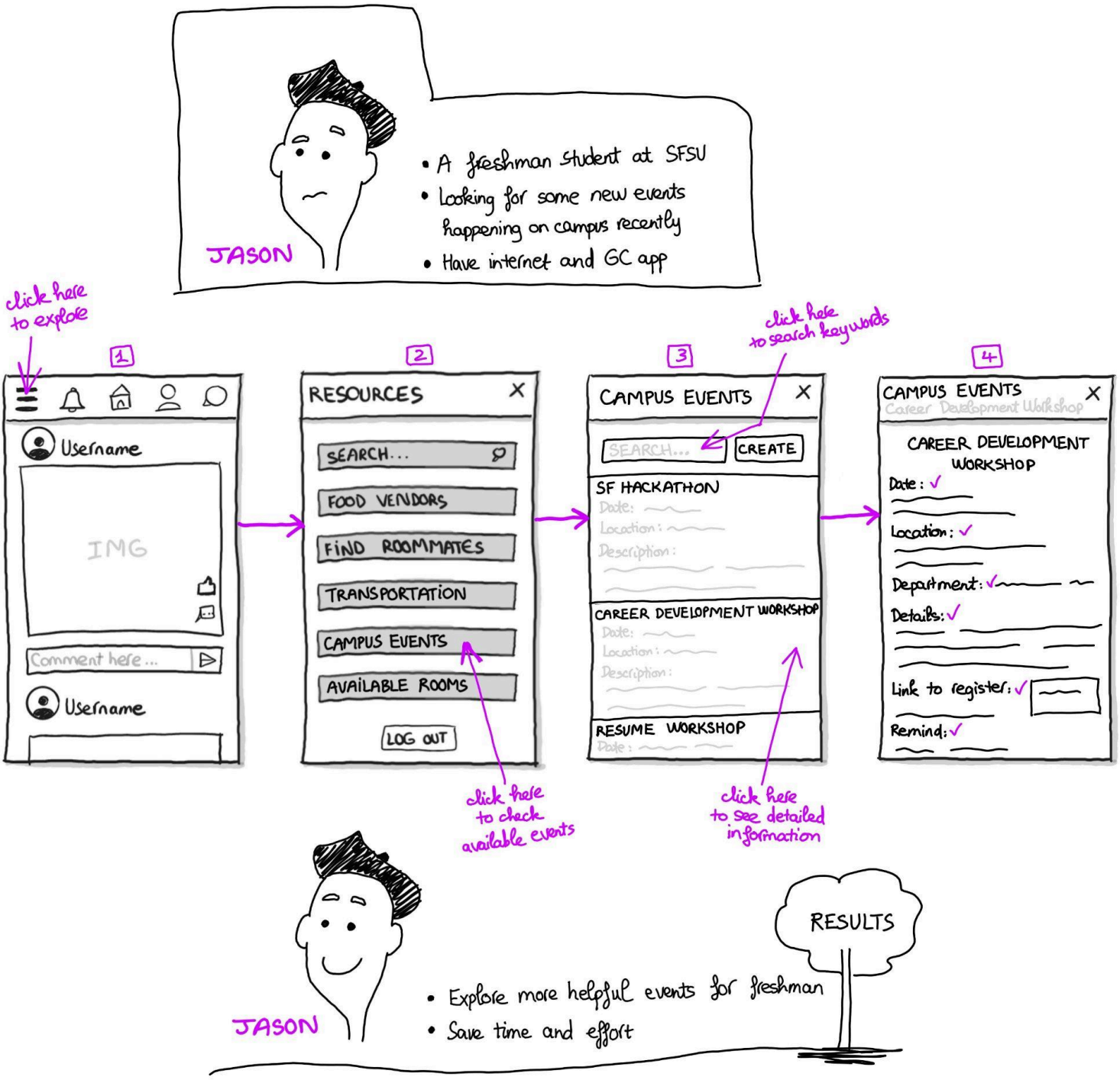
6. Use Case 6 - Big Announcement



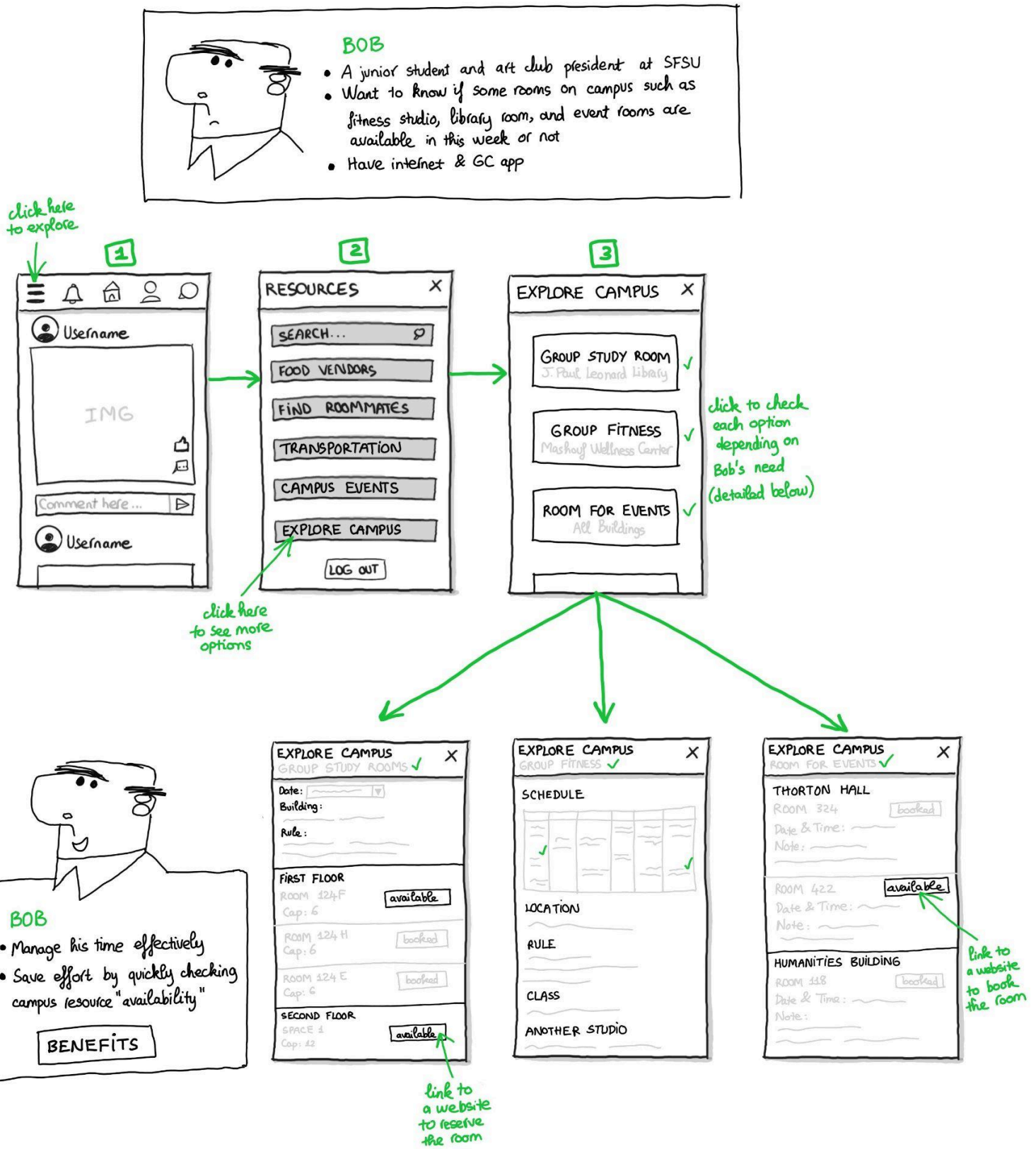




7. Use Case 7 - Campus Events



8. Use Case 8 - Exploring Campus Room (Priority 3)



IV. HIGH LEVEL DB ARCHITECTURE & ORGANIZATION

Database Requirements

Priority One Functional Requirements with Cardinality

1. User (Strong)
    - A user shall create at most one account
    - A user shall send many messages
    - A user is a student, professor or friend
    - A user shall be able to write many comments
    - A user shall be able to push many likes
    - A user shall be able to create many posts
    - A user shall access many resources
    - A user shall make many friends (recursive via friend)
    - A user shall receive many notifications through friend request processes
    - A user's homepage feed consists of likes, comments and posts

2. Account (Weak)
    - An account shall belong to one and only one user
    - An account shall have one and only one profile

3. Profile (Weak)
    - A profile shall be associated with one and only one account
    - A profile's profile feed shall consist of likes, comments and posts

4. Student (Weak)
    - A student is a user
    - A student shall create many events

5. Professor (Weak)
    - A professor is a user

6. Message (Weak)
    - A message shall be sent by one user
    - A message is either public or private

7. Private message (Weak)
    - A private message is a type of message

8. Public message (Weak)
    - A public message is a type of message
    - A public message shall be associated with one channel

9. Channel (Strong)
    - A channel shall be contain many public messages

10. Notification (Weak)
    - A notification shall be alerted to one user
    - A notification shall be triggered by at least one friend requests (recursive)

11. Friend (Weak)
    - A friend is a type of user
    - A friend shall be referred in making many friend requests

12. Post (Strong)
    - A post shall have many comments
    - A post shall have many likes
    - A post shall be written by at most one user

13. Like (Weak)
    - A like shall belong to one post
    - A like shall be pushed by exactly one user

14. Comment (Weak)
    - A comment shall belong to one post
    - A comment shall be written by exactly one user

15. Resource (Strong)
    - A resource shall contain many food vendors
    - A resource shall contain many transportation informations
    - A resource shall contain many events
    - A resource shall be accessed by many users

16. Food Vendor (Weak)
    - Food Vendor shall be in one section of Resource

17. Transportation (Weak)
    - Transportation shall be in one section of Resource

18. Event (Strong)
    - Event shall be in one section of Resource
    - An Event is started by many students

Define the DBMS

We decided to use MySQL as the database management system (DBMS) for this project due to its lightweight nature, simplicity of setup, and seamless integration into various programming languages and frameworks, which align well with the project's requirements for a compact, easy-to-use database solution.

Database Organization

a. The Entities, Attributes, Relationship, and Domains at the high level

1. User (Strong)
    - user_id: key, numeric
    - first_name: alphabetic
    - last_name: alphabetic
    - sfsu_email: unique, alphanumeric (unique)

2. Account (Weak)
    - account_id: key, numeric
    - password: composite, alphanumeric
    - user_id: foreign key, alphanumeric
    - timestamp: composite, timestamp
    - Username: composite, alphanumeric (unique)

3. Profile (Weak)
    - profile_id: primary key, numeric
    - biography: composite, alphanumeric
    - avatar: image
    - account_id: foreign key, alphanumeric
    - Friend_count: composite, numeric

4. Student (Weak)
    - user_id: foreign key, numeric
    - Student_id: primary key, numeric
    - major: alphabetic
    - minor: alphabetic

5. Professor (Weak)
    - user_id: foreign key, numeric
    - Professor_id: key, numeric
    - department: alphabetic

6. Notification (Weak)
    - notification_id: key, numeric
    - timestamp: composite, date
    - user_id: foreign key, alphanumeric

7. Message (Weak)
    - message_id: key, numeric
    - message_content: alphanumeric
    - message_time: composite, date
    - message_type: alphabetic
    - Message_status: composite, alphanumeric

8. Comment (Weak)
    - comment_id: key, numeric
    - user_id: foreign key, numeric
    - post_id: foreign key, numeric
    - comment_content: alphanumeric
    - comment_time: composite, date

9. Like (Weak)
    - like_id: key, numeric
    - user_id: foreign key, numeric
    - post_id: foreign key, numeric
    - timestamp: composite, date

10. Public message (Weak)
    - message_id: foreign key, numeric
    - Private_id: key, numeric

11. Private message (Weak)
    - message_id: foreign key, numeric
    - public_id: key, numeric

12. Channel (Strong)
    - channel_id: key, numeric
    - channel_name: alphabetic

13. Post (weak)
    - post_id: key, numeric
    - post_description: alphanumeric
    - user_id: foreign key, alphanumeric
    - post_time: composite, date
    - num_likes: numeric
    - num_comments: numeric

14. Resource (strong)
    - resource_id: key, numeric

15. food vendor (Weak)
    - food_vendor_id: key, numeric
    - resource_id: foreign key, numeric
    - review: alphanumeric
    - rating: int (0, 1, 2, 3, 4, 5)

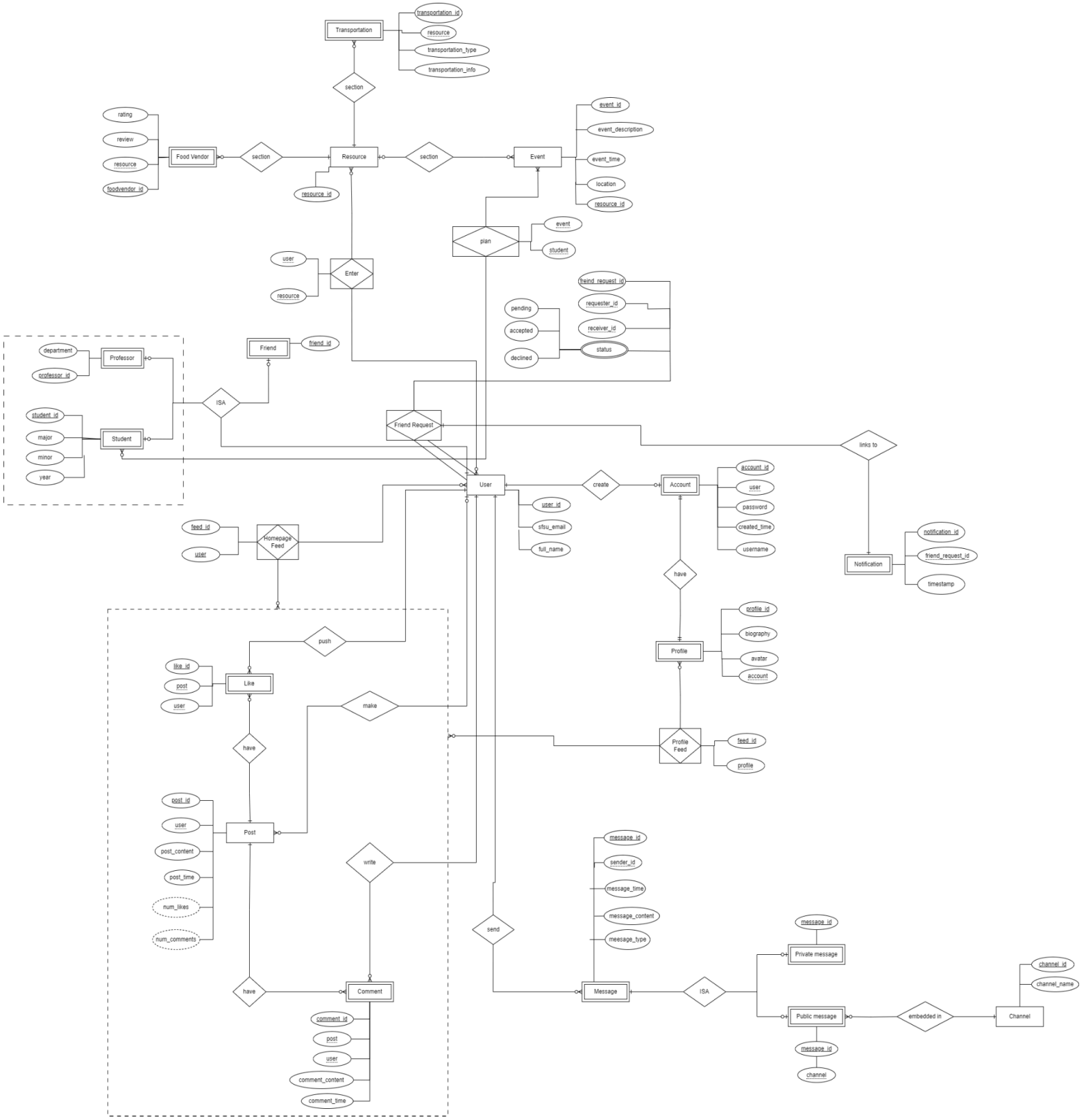
16. transportation (Weak)
    - transportation_id: key, numeric
    - transportation_type: alphanumeric
    - transportation_info: alphanumeric
    - Resource_id: foreign key, numeric

17. Event (Weak)
    - event_id: key, numeric
    - event_description: alphanumeric
    - event_time: composite, date
    - event_location: composite, alphanumeric
    - Resource_id: numeric

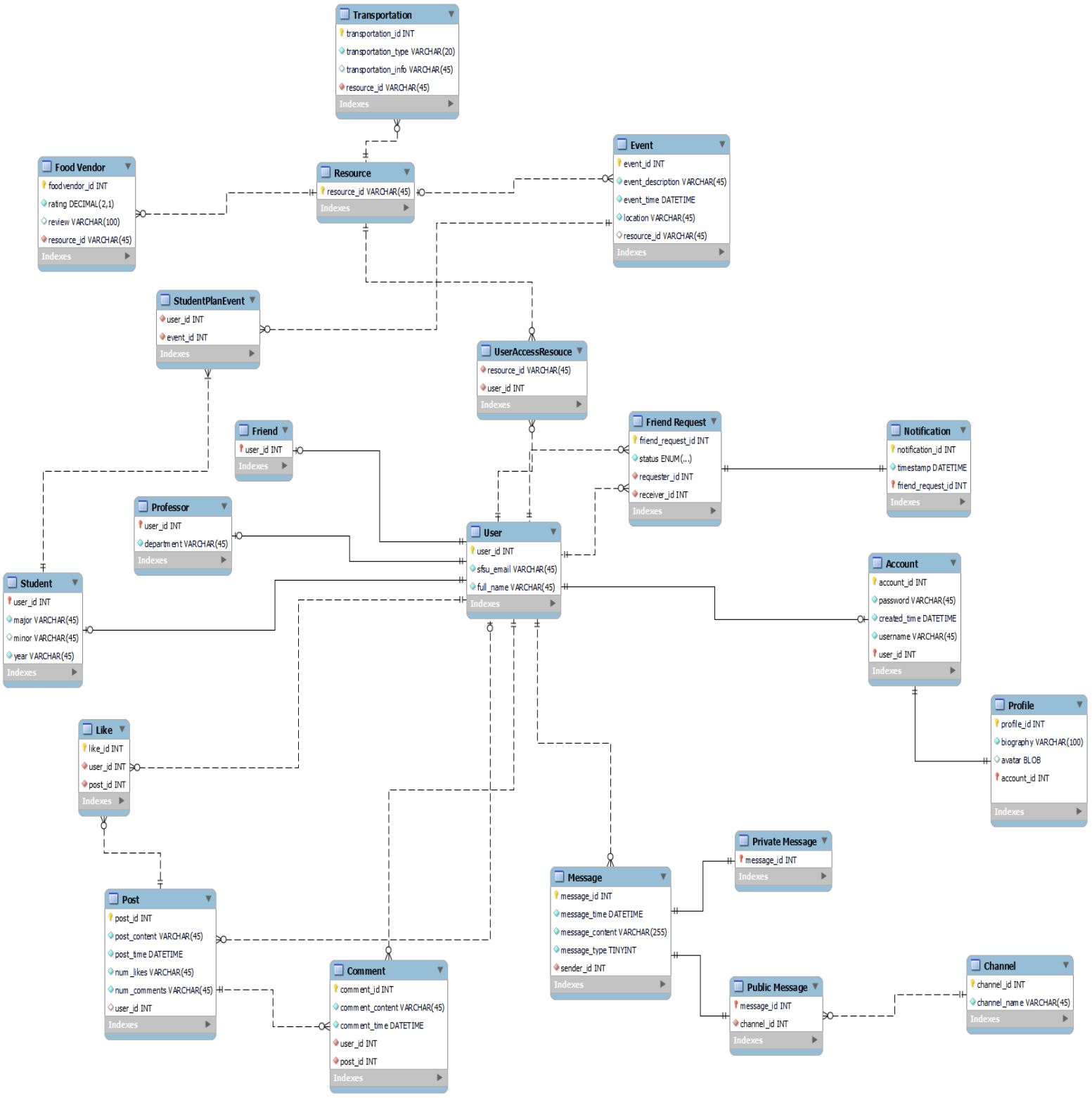
18. Friend (Weak)
    - friend_id: key, numeric

b. An Entity Relationship Diagram (ERD)

GatorConnect.drawio



c. An Entity Establishment Relationship Diagram (EER)



d. Forward engineering the database model

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_
ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

DROP DATABASE IF EXISTS mydb;
CREATE DATABASE IF NOT EXISTS mydb;
USE mydb;

-- -----------------------------------------------------
-- Table `mydb`.`User`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`User` (
  `user_id` INT UNSIGNED NOT NULL,
  `sfsu_email` VARCHAR(45) NOT NULL,
  `full_name` VARCHAR(45) GENERATED ALWAYS AS (CONCAT(first_name, ' ', last_name))
VIRTUAL,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE INDEX `sfsu_email_UNIQUE` (`sfsu_email` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Account`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Account` (
  `account_id` INT UNSIGNED NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `created_time` DATETIME NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  `user_id` INT UNSIGNED NOT NULL,
```

```
  PRIMARY KEY (`account_id`, `user_id`),
  UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Profile`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Profile` (
  `profile_id` INT UNSIGNED NOT NULL,
  `biography` VARCHAR(100) NOT NULL,
  `avatar` BLOB NULL,
  `account_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`profile_id`, `account_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Student` (
  `user_id` INT UNSIGNED NOT NULL,
  `major` VARCHAR(45) NOT NULL,
  `minor` VARCHAR(45) NULL,
  PRIMARY KEY (`user_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Professor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Professor` (
  `user_id` INT UNSIGNED NOT NULL,
  `department` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`user_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Friend`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Friend` (
  `user_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`user_id`))
```

```
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Friend Request`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Friend Request` (
  `friend_request_id` INT UNSIGNED NOT NULL,
  `status` ENUM('pending', 'accepted', 'declined') NOT NULL,
  `requester_id` INT UNSIGNED NOT NULL,
  `receiver_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`friend_request_id`),
  INDEX `fk_Friend Request_User1_idx` (`requester_id` ASC) VISIBLE,
  INDEX `fk_Friend Request_User2_idx` (`receiver_id` ASC) VISIBLE,
  CONSTRAINT `fk_Friend Request_User1`
    FOREIGN KEY (`requester_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Friend Request_User2`
    FOREIGN KEY (`receiver_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Notification`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Notification` (
  `notification_id` INT UNSIGNED NOT NULL,
  `timestamp` DATETIME NOT NULL,
  `friend_request_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`notification_id`, `friend_request_id`),
  INDEX `fk_Notification_Friend Request1_idx` (`friend_request_id` ASC) VISIBLE,
  CONSTRAINT `fk_Notification_Friend Request1`
    FOREIGN KEY (`friend_request_id`)
    REFERENCES `mydb`.`Friend Request` (`friend_request_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`Resource`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Resource` (
  `resource_id` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`resource_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Food Vendor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Food Vendor` (
  `foodvendor_id` INT UNSIGNED NOT NULL,
  `rating` DECIMAL(2,1) NOT NULL,
  `review` VARCHAR(100) NULL,
  `resource_id` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`foodvendor_id`),
  INDEX `fk_Food Vendor_Resource1_idx` (`resource_id` ASC) VISIBLE,
  CONSTRAINT `fk_Food Vendor_Resource1`
    FOREIGN KEY (`resource_id`)
    REFERENCES `mydb`.`Resource` (`resource_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`UserAccessResouce`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`UserAccessResouce` (
  `resource_id` VARCHAR(45) NOT NULL,
  `user_id` INT UNSIGNED NOT NULL,
  INDEX `fk_Resource_has_User_User1_idx` (`user_id` ASC) VISIBLE,
  INDEX `fk_Resource_has_User_Resource1_idx` (`resource_id` ASC) VISIBLE,
  CONSTRAINT `fk_Resource_has_User_Resource1`
    FOREIGN KEY (`resource_id`)
    REFERENCES `mydb`.`Resource` (`resource_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Resource_has_User_User1`
    FOREIGN KEY (`user_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE RESTRICT
```

```
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Transportation`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Transportation` (
  `transportation_id` INT UNSIGNED NOT NULL,
  `transportation_type` VARCHAR(20) NOT NULL,
  `transportation_info` VARCHAR(45) NULL,
  `resource_id` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`transportation_id`),
  INDEX `fk_Transportation_Resource1_idx` (`resource_id` ASC) VISIBLE,
  CONSTRAINT `fk_Transportation_Resource1`
    FOREIGN KEY (`resource_id`)
    REFERENCES `mydb`.`Resource` (`resource_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Event`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Event` (
  `event_id` INT UNSIGNED NOT NULL,
  `event_description` VARCHAR(45) NOT NULL,
  `event_time` DATETIME NOT NULL,
  `location` VARCHAR(45) NOT NULL,
  `resource_id` VARCHAR(45) NULL,
  PRIMARY KEY (`event_id`),
  INDEX `fk_Event_Resource1_idx` (`resource_id` ASC) VISIBLE,
  CONSTRAINT `fk_Event_Resource1`
    FOREIGN KEY (`resource_id`)
    REFERENCES `mydb`.`Resource` (`resource_id`)
    ON DELETE SET NULL
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Post`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Post` (
  `post_id` INT UNSIGNED NOT NULL,
  `post_content` VARCHAR(45) NOT NULL,
  `post_time` DATETIME NOT NULL,
  `num_likes` VARCHAR(45) NOT NULL,
  `num_comments` VARCHAR(45) NOT NULL,
  `user_id` INT UNSIGNED NULL,
  PRIMARY KEY (`post_id`),
  INDEX `fk_Post_User1_idx` (`user_id` ASC) VISIBLE,
  CONSTRAINT `fk_Post_User1`
    FOREIGN KEY (`user_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Like`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Like` (
  `like_id` INT UNSIGNED NOT NULL,
  `user_id` INT UNSIGNED NOT NULL,
  `post_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`like_id`),
  INDEX `fk_Like_User1_idx` (`user_id` ASC) VISIBLE,
  INDEX `fk_Like_Post1_idx` (`post_id` ASC) VISIBLE,
  CONSTRAINT `fk_Like_User1`
    FOREIGN KEY (`user_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Like_Post1`
    FOREIGN KEY (`post_id`)
    REFERENCES `mydb`.`Post` (`post_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Comment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Comment` (
```

```
  `comment_id` INT UNSIGNED NOT NULL,
  `comment_content` VARCHAR(45) NOT NULL,
  `comment_time` DATETIME NOT NULL,
  `user_id` INT UNSIGNED NOT NULL,
  `post_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`comment_id`),
  INDEX `fk_Comment_User1_idx` (`user_id` ASC) VISIBLE,
  INDEX `fk_Comment_Post1_idx` (`post_id` ASC) VISIBLE,
  CONSTRAINT `fk_Comment_User1`
    FOREIGN KEY (`user_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Comment_Post1`
    FOREIGN KEY (`post_id`)
    REFERENCES `mydb`.`Post` (`post_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Message`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Message` (
  `message_id` INT UNSIGNED NOT NULL,
  `message_time` DATETIME NOT NULL,
  `message_content` VARCHAR(255) NOT NULL,
  `message_type` TINYINT NOT NULL,
  `sender_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`message_id`),
  INDEX `fk_Message_User1_idx` (`sender_id` ASC) VISIBLE,
  CONSTRAINT `fk_Message_User1`
    FOREIGN KEY (`sender_id`)
    REFERENCES `mydb`.`User` (`user_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Private Message`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Private Message` (
```

```
  `message_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`message_id`),
  CONSTRAINT `fk_Private Message_Message1`
    FOREIGN KEY (`message_id`)
    REFERENCES `mydb`.`Message` (`message_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Channel`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Channel` (
  `channel_id` INT UNSIGNED NOT NULL,
  `channel_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`channel_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Public Message`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Public Message` (
  `message_id` INT UNSIGNED NOT NULL,
  `channel_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`message_id`),
  INDEX `fk_Public Message_Channel1_idx` (`channel_id` ASC) VISIBLE,
  CONSTRAINT `fk_Public Message_Message1`
    FOREIGN KEY (`message_id`)
    REFERENCES `mydb`.`Message` (`message_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Public Message_Channel1`
    FOREIGN KEY (`channel_id`)
    REFERENCES `mydb`.`Channel` (`channel_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`StudentPlanEvent`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`StudentPlanEvent` (
  `user_id` INT UNSIGNED NOT NULL,
  `event_id` INT UNSIGNED NOT NULL,
  INDEX `fk_Student_has_Event_Event1_idx` (`event_id` ASC) VISIBLE,
  INDEX `fk_Student_has_Event_Student1_idx` (`user_id` ASC) VISIBLE,
  CONSTRAINT `fk_Student_has_Event_Student1`
    FOREIGN KEY (`user_id`)
    REFERENCES `mydb`.`Student` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Student_has_Event_Event1`
    FOREIGN KEY (`event_id`)
    REFERENCES `mydb`.`Event` (`event_id`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Media Storage

A user is not able to post or send images, videos or audios on our application. We will provide avatars which the user can choose from and use them as their profile pictures. Those avatars and some other images we might have inside our application will be stored inside DB BLOB's. Those are database types to store binary data like images. We will provide around 5 different Avatars with each at most 60 x 60 Pixel.

V. HIGH LEVEL APIs & MAIN ALGORITHMS

High Level APIs

1. Login API:

   This API allows users to log in using a username and password. A POST request is sent to the database when users enter their username and password. The API checks if the username exists and the password is correct. This uses JWT which is a token-based authentication method. When a user logs in, the server generates a JWT for the user then the user includes JWT and subsequent requests to the server, allowing the server to verify and authorize their access to the resources, and the user stays logged in across different pages. The API provides the user with an error message when the user enters an incorrect username or password. The API will not specify whether a username is incorrect or password to ensure security.

2. Creating Accounts:

   This API allows users to create an account and register by adding a username, email, password, and fullname. There is an option for a user role and if the user chooses a student, there is an option for the major they are in and the year they graduate. The API only checks if the email ends with sfsu.edu to make sure it's a San Francisco State student. When the user clicks the register button, and if everything is valid, the user's information is added to the database. After registering, the user is automatically logged in.

3. Search API:

   This API allows users to search for other users in the search bar. The user can narrow down the search with these parameters. Which are username, major(degree), and year of graduation. The API will use GET to retrieve data and search to query the database based on the parameters, the API fetches a list of users with information such as username, major, and year of graduation that matches the search criteria under the search bar. The API will provide error messages if there are no users that match the criteria in the database.

4. Notification API:

   This API is responsible for sending notifications. The API notifies the users when it receives a new friend request. The notification pop-up has the username of the user that sends the request with a message that says " accept a friend request" or "decline friend request". When a user's friend request is accepted, the API notifies both users that they are friends now. The API handles errors very efficiently, it notifies users if a friend request is successfully sent and if not it lets the user know there is an error. API also has a rate limiting, if the user spams the send friend request button more than 5 times, the API will respond with "too

many requests, try again later". This is to make sure users aren't being spammed. (We can also make it so there can only be one friend request sent)

5. Detecting offensive word on public post: (priority 2)
   This API will scan the post and check if the word is inappropriate and will get flagged. There will be an option for users to report a person's post and message. It will be looked at by the administrator if it is offensive or not. The API POST user ID of people who have been flagged and there will be a limit on the flagged they can get and if they succeed the flagged limit.

Non-Trivial Algorithms

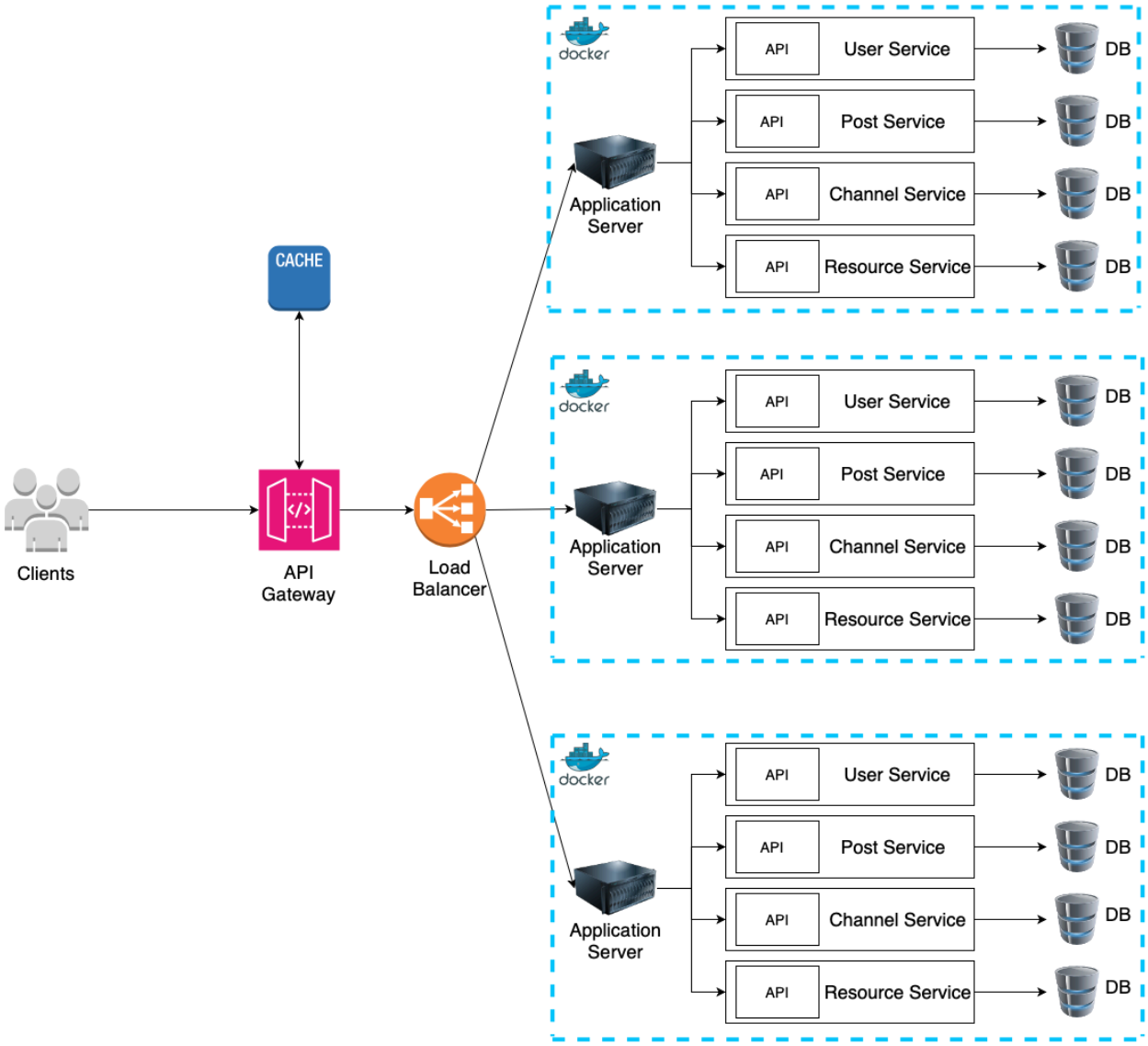
Categorizing posts: (Priority 3)
This Algorithm will use a dictionary where keys are pre-determined categories and values are lists of posts that belong to that category. The algorithm will do its best to determine which categories the post should go in. The Algorithm will use the Natural Language Processing technique to determine the categories the post should fall under.

New changed SW tools and frameworks

No changes in SW tools and frameworks

VI. SYSTEM DESIGN

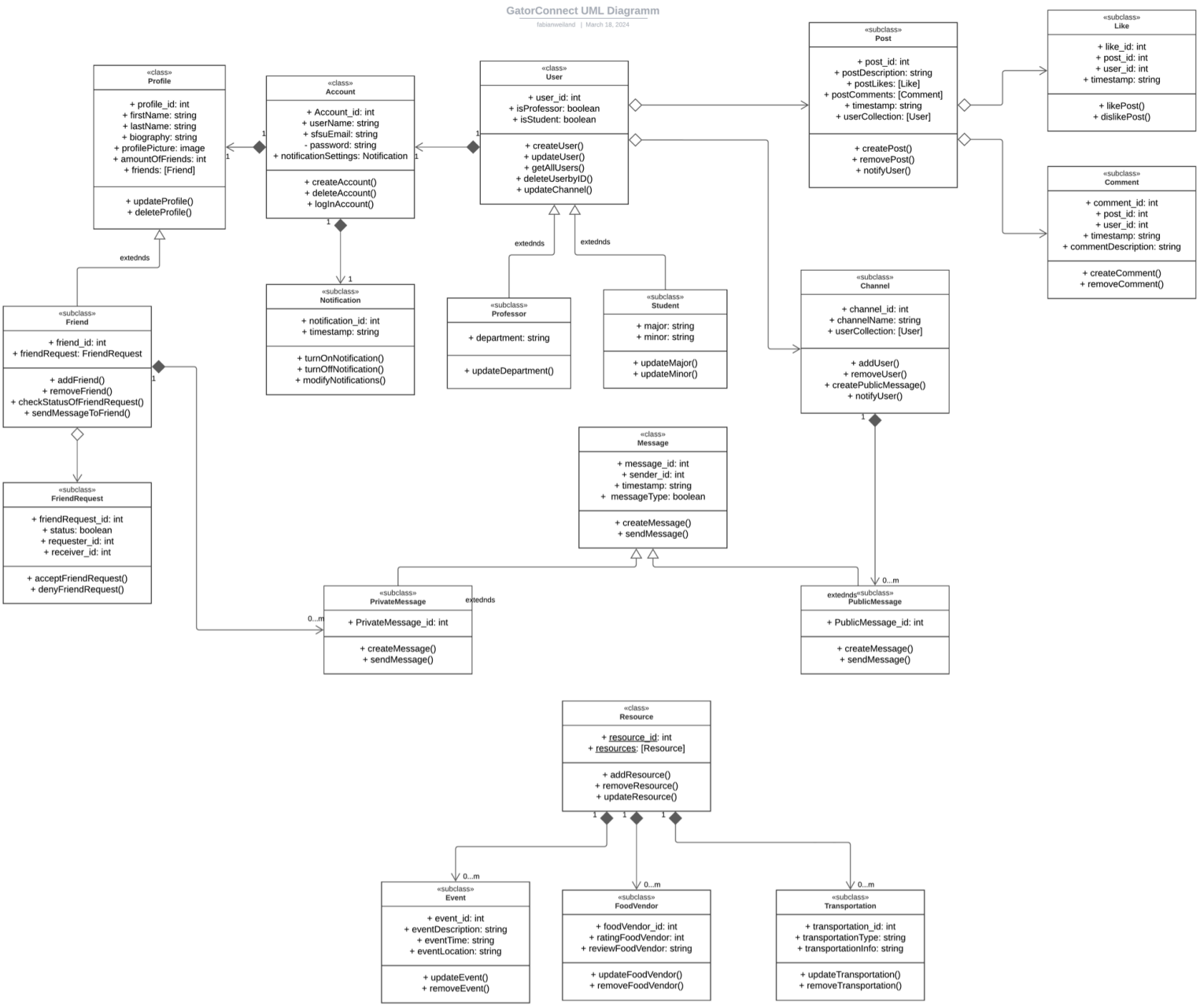
System Diagram



Summary of the components of your system architecture

Our application utilizes microservices architecture to leverage benefits such as scalability, reliability and fault tolerance, and flexibility. By breaking down complex functionalities into smaller, independently deployable services, we can scale components individually, ensure resilience against failures, and rapidly adapt to changing requirements, ultimately enhancing the agility and efficiency of our system. Our cache can be accessed via the API Gateway. This improves application performance by storing frequently accessed data closer to the user, reducing the need for repeated database or network requests. Our Load Balancer distributes incoming traffic across 3 servers and prevents any single server from becoming overwhelmed. By evenly distributing the workload, we enhance the reliability, availability, and scalability of our application. All of our three servers are inside a Docker Container to streamline development, deployment, and scaling processes. By encapsulating each server/service and its dependencies within lightweight, isolated containers, we ensure consistent and reliable execution across various environments, enhancing portability and efficiency in managing our microservices architecture. To enhance the Security of our application we are using distributed systems to prevent bottlenecks. We are also developing a strong password policy, and two-factor authentication.

UML Class diagrams

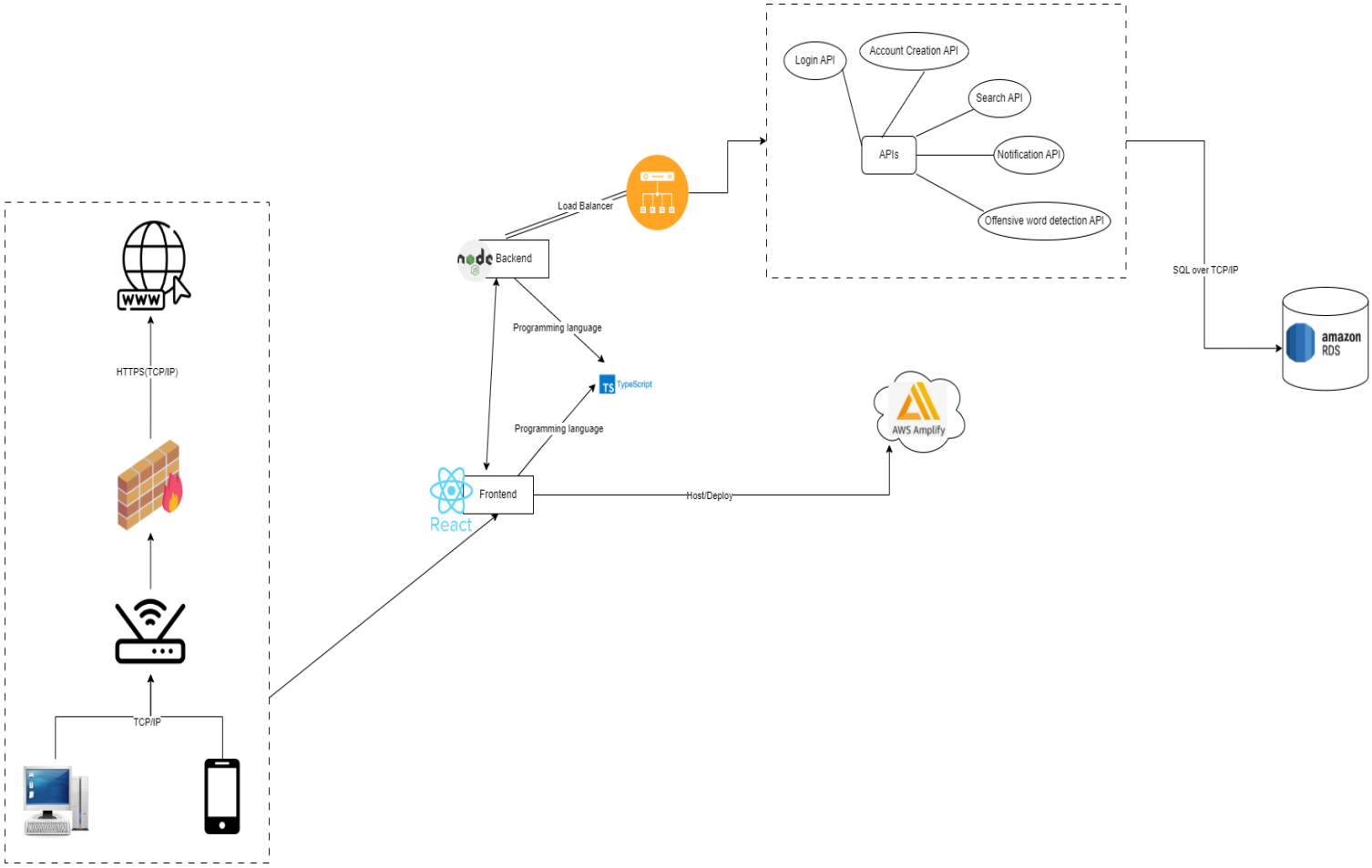


Summary

In my UML Diagram I used a behavioral Design Pattern. As we are implementing a social media application for students/professors at San Francisco State University, the behavioral Design Pattern, especially the “Observer” fits best for our needs. The User class in our case is the Observer and the Student & Professor are the concrete Observers. The Channel and the Post are the Subject and they hold a collection of Users. You can add or remove Users from the Channel and once there is an update in the Channel all the Users who are stored inside the userCollection get notified and updated.
We also have to distinguish between a privateMessage and a publicMessage. Both classes inherit from the parent Message class but they are used in different scenarios. A PublicMessage object can just exist inside a Channel class, therefore it has an aggregation relationship with the Channel class. The privateMessage object on the other hand just exists inside the Friend class, as you can just send private messages to friends.

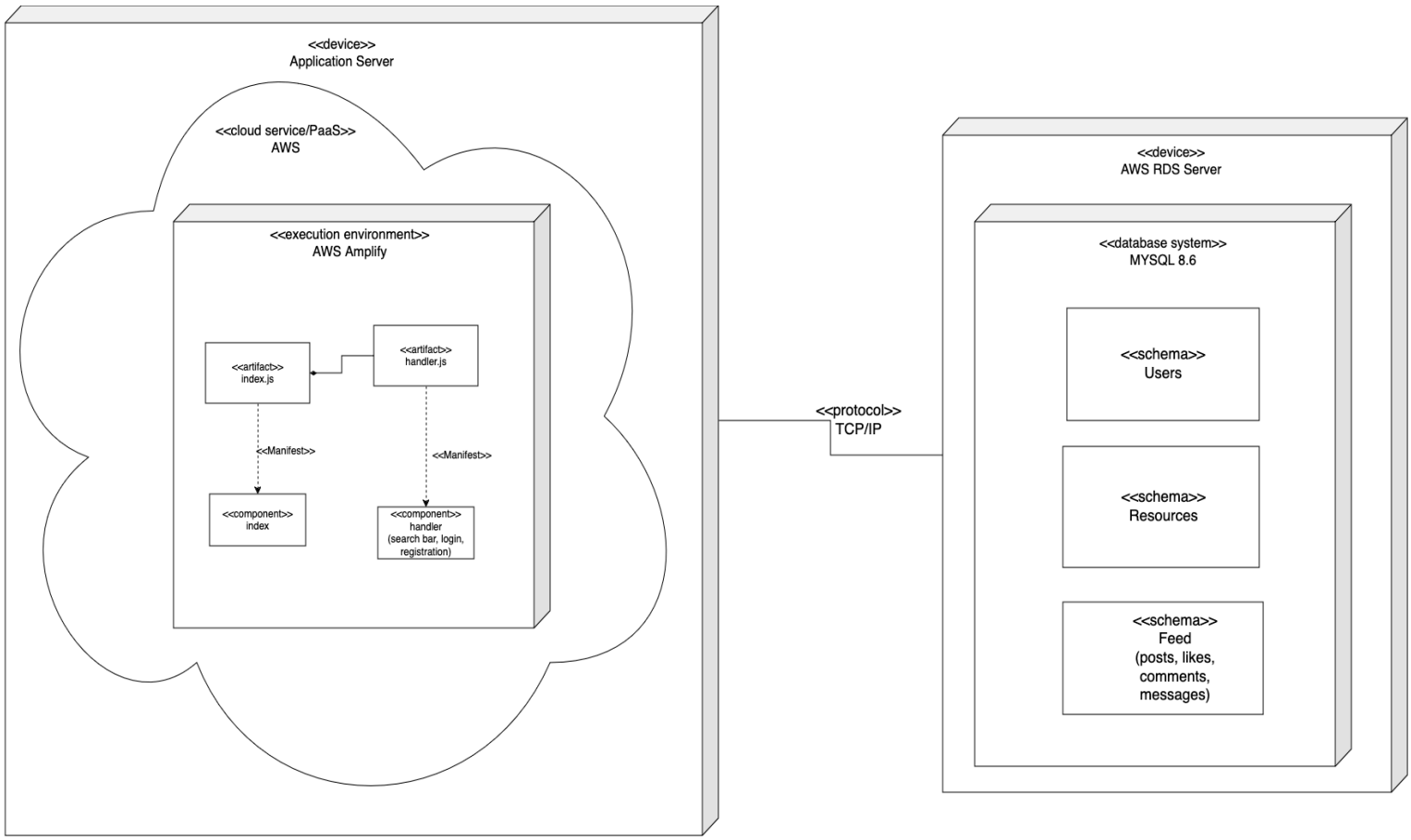
VII. HIGH LEVEL APP NETWORK & DEPLOYMENT DESIGN

Application Networks Diagram



High Level Deployment Diagram

https://drive.google.com/file/d/1Z6-98ExJw1G3YJX3INrRVZTzp6X5cEiy/view?usp=sharing



VIII. IDENTIFY *ACTUAL* KEY RISKS

Skill Risks
Some of our team members do not have much experience with Typescript which is our key backend language. We plan to improve these skill risks by adding several online/in-person meetings to encourage knowledge sharing and cross-training among team members to broaden their skill sets and mitigate reliance on individual expertise in the next milestones. Also, actively share useful frameworks or potential sources to learn together.

Schedule Risks
The two weeks leading up to spring break were proved to be rather difficult for in-person meet-ups due to the high volume of midterms and projects for several other classes. We fought this issue by maintaining constant communication via Zoom, Discord, and Messenger to ensure a successful completion of M2.

Technical Risks
Our team was quite unsure about using the EC2 instance provided by AWS, it seemed like an inefficient way to deploy the site and overall manage our progress. So, on top of EC2, we use AWS Amplify which manages updates for us. We still use EC2 because the professor requires it, but we deploy our website with Amplify.

Teamwork Risks
As a team, we have been consistent with our internal deadlines following our meeting with the professor on Thursday, 03/07/24. During the meeting, the professor emphasized the importance of listening to our lead when she sets a due date and keeping open communication if we feel our timeline needs a slight extension. Before this interaction, we were not cohesive with our work and constrained by rigor when it came to accomplishing tasks on time.
Another issue we noticed came down to the usage of GitHub and being on top of pushing our code. We have had instances where some team members needed to write portions of code that were already completed, it just wasn't pushed to GitHub yet. Since then, we've all been on top of ensuring clear pushing and pulling when beginning to work a new day or even a later hour.

Legal/Content Risks
A potential legal risk is that we get sued if we use the official SFSU pictures or other image sources to decorate the avatar or some main pages for our application. We will solve this issue by using the open-source images and trying to design unique images that will relieve us of any copyright issues.
Also, the potential content risk involves professor names and information to use to rate my professor. We will solve this by temporarily leaving this feature as Priority 2 or just using some of the professor names that have been allowed as examples for this feature.

IX. PROJECT MANAGEMENT

For managing M2 tasks, our team employs **Trello** as our primary task management tool. Trello's unified dashboard view allows us to streamline task tracking efficiently. Each team member has their designated list where they outline their detailed tasks in chronological order and update their progress status by themselves such as TODO, IN-PROCESS, and DONE. This layout provides a clear overview of the workload and progress for each member and team lead. Additionally, we maintain a dedicated list titled "Questions" where team members can pose important queries or seek clarification from the role lead or professor. Also, a "Project Resources" list serves as a repository for crucial documents such as milestone version links, team notes, meeting summaries, and deadlines for smaller milestone tasks.

Moreover, we utilize **Google Docs** as collaborative workspaces. It ensures everyone is on the same page regarding content details and expectations on the project. The ability to leave a comment on a document facilitates providing feedback, asking questions, or idea sharing. This promotes effective communication, synergy, and clarity among team members.

In addition to in-person meetings and classroom interactions, **Discord** (channel #TEAM03 from CSC648 course's server) serves as our primary communication channel for discussions, questions, and general team collaboration. It enables efficient resolution of queries, quick exchanges, and effective delegation of tasks in general.

In the future, we continue to utilize Trello, Google Docs, and Discord for task management and communication.

X. DETAILED LIST OF CONTRIBUTIONS

NO	Member	Contribution	Rating
1	Ralph Quiambao	* Worked on data definitions * Implemented the new homepage * Developed authentication pages and fixed related issues * Implemented the search bar features and fixed related issues * Good communication with others	9.5
2	Karma Gyalpo	* Revised the functional and non-functional requirements on M1v2 * Worked on data entity & domain description * Completed the high-level APIs and main algorithms * Fixed the AWS instance issue and hosted the web on Amplify * Implemented the backend for login/signup and connected to the database * Helped the search bar features * Strong communication with others	10
3	Dustin Meza	* Helped with revising the competitive analysis on M1v2 * Worked on database requirements in DB architecture * Worked on all prioritized functional requirements * Completed define the DBMS * Helped with the ERD diagram and related issues * Organized the database content and M2 documentation in general * Completed the deployment diagram * Completed the identity actual key risks * Implemented the contents of optional pages for frontend * Super strong communication with others and professors	10
4	Fabian Weiland	* Revised the main data description on M1v2 and committed it to GitHub * Helped with data entity & domain description * Completed the system diagram and its summary * Completed the UML class diagram and its summary * Worked on media storage * Good communication with others	9
5	Jeawan Jang	* Revised the competitive analysis and separated the diagrams for each use case on M1v2 * Helped with revising the functional and non-functional requirements on M1v2 * Worked hard on the ERD and EER diagrams and committed them to GitHub * Completed the forward engineering db model * Completed the application network diagram * Helped with organizing M2 documentation in general * Strong communication with others	10

6	Hoang-Anh Tran (team lead rating by the professor)	* Set up M2v1 documents, organized Trello management, and invited all members & professors * Grammar checked and cleaned the M1v2 document * Sketched all UI mockups and storyboards by hand * Completed the project management & detailed list of contributions section * Helped with the ERD diagram * Added one more content for the identity of actual key risks * Organized and formatted the M2v1 regularly * Reviewed all tasks of everyone every week * Plan the meeting and next tasks in general to Discord * Implemented the UI for all main pages and fixed the responsive issues for smaller width-screen * Fixed react-icons issue and added more post-card to the homepage