SW Engineering CSC648-848-05 Fall2023

Project: BiteRate
Team number: Team 5
Team Lead: Alec Nagal | anagal1@mail.sfsu.edu
Frontend Lead: Ali Alsharif
Backend Lead: Gerry Putra
Doc Master: Gabriella Arcilla
Database Master: Kenneth Pang
Backend Assistant: Robel Ayelew
Team: James Lu, Robin Reyes
Milestone 2
Date: November 2, 2023

Milestone/Checkpoint #	**Date Submitted**	**Revision #**
Milestone 1v1 Checkpoint #1	09/10/2023	Revision #0
Milestone 1v1 Checkpoint #2	09/21/2023	Revision #0
Milestone 1v2	10/12/2023	Revision #1
Milestone 2v1	10/12/2023	Revision #0
Milestone 2v2	11/2/2023	Revision #1

Table of Contents

Data Definitions

1. ***User***

The user entity shall represent registered users who uses the BiteRate application. The user entity shall have the following functionalities: registering, posting reviews, managing personal profile page, and more.

Attributes:

- userID: unique id used to identify each individual user
- lastVisited: stores when the user visited the website last

2. ***Account***

The account entity represents a user's profile account on the BiteRate application. The account entity shall hold the user's personal information, such as name, email, password, and more.

Attributes:

- AccountID: unique id used to identify each individual account
- Password: encrypted string used to store the user's password
- firstName: string to represent user's first name
- lastName: string to represent user's last name
- Email: string to store the user's email address
- Address: string to store the user's address
- Phone: string to store the user's phone number
- dateCreated: string to store the date the user account was created

3. ***Admin***

The admin entity represents a type of user who has full control of the system. Admin entity shall have the following functionalities: access to all existing user accounts, add new restaurants and remove existing restaurants.

Attributes:

- adminID: unique id used to identify each individual admin
- Account: used to store the account associated with the admin user
- Role: string to represent what type of role this admin user has
- startDate: string to store the date the admin user started

- endDate: string to store the date the admin user ended
- name: string to represent restaurant's name

4. ***Restaurant***

The restaurant entity represents the physical business of the restaurant, containing the following information: address, location, rating score, cuisines, and more.

Attributes:

- Cuisines: used to store all the cuisines the restaurant provides
- Address: string to store the restaurant's address
- restaurantID: unique id used to identify each individual restaurant
- openDate: string to store the date the restaurant first opened

5. ***Friends***

The friends entity represents registered used who follow each other. The friends entity shall have the following functionalities: see other friends activities, such as restaurants they reviewed.

Attributes:

- followID: unique id used to identify when the users followed each other
- Account: used to store the account associated with the user

6. ***Point/Score***

The point/score entity represents the ranks registered users made for the restaurants they reviewed.

Attributes:

- Score: numeric number the user ranked the corresponding restaurant
- userID: id associated with user who ranked the restaurant

7. ***Rating***

The rating entity represents the overall rating of a single restaurant made by a user. The rating shall be calculated by averaging the rating that the restaurant received from all its customers.

Attributes:

- Rate: used to represent the rating the user scored
- Date: string to represent the date the user rated this restaurant
- dataModified: string to represent the data the user modified their rating

8. ***Review***

The review entity represents critiques and comments left by customers on a single restaurant. The review entity shall have the following functionalities: embed images, update, modify, or delete by the author.

Attributes:

- Review: string to represent the critiques left by the customers
- Images: used to represent the embedded images
- timeStamp: date to represent the date a review was made

9. ***Recommendation (Recs)***

The recommendation entity represents the highly rated restaurants that are recommended for a specific user. This entity shall be a form of storage where it keeps the high-rated restaurants to recommend restaurants to registered users in the future.

Attributes:

- Restaurants: used to represent the highly rated restaurants
- userID: id associated with the specific user that these restaurants are recommended to

10. ***User Activity/History***

The user activity entity represents all the activities of the user such as the restaurant he or she rated, reviewed, and recommended.

Attributes:

- Activities: used to represent all the activities made by the user
- userID: id associated with the specific user that made these reviews

11. ***ChatAssistance***

The chatAssistance entity represents a virtual helper that serves to answer all questions regarding registered restaurants, cuisines, and recommend restaurants to registered users based on the cuisine, restaurant name, or location.

Attributes:

- chatID: unique id used to identify each chatAssistance account
- Account: used to store the account associated with the chatAssistance
- chatLog: used to store the conversation with the user
- timeStamp: date to represent the date a review was made

12. Address

The address entity represents the location of restaurant businesses. This entity shall contain the following information: street number, name, city, state, and zip code.

Attributes:

- addressLine1: string that represents the address of the restaurant
- City: string to represent the city which the restaurant is in
- State: string to represent the state which the restaurant is in
- Country: string to represent the country which the restaurant is in
- postalCode: string to represent the postal code of the restaurant address
- addressID: string to represent the id of the restaurant's address

13. Notifications

The notifications entity represents important information that users are alerted by the system. This entity shall have the following functionalities: notify registered users if they receive messages, friend requests, restaurant recommendations, and updates about their reviews.

Attributes:

- Message: string to store the content of the notification
- timeStamp: date to represent the date a notification was made
- userID: id associated with the specific user that made these reviews

Prioritized Functional Requirements

Priority 1:

1. ***Registered Users***

1.2. A registered user shall be authenticated by the system to verify their identity before granting full access to the application.
1.3. A registered user shall be able to rate a restaurant.
1.4. A registered user shall be able to review a restaurant.
1.5. A registered user shall be able to recommend a restaurant to other registered users.
1.6. A registered user shall be able to search for any restaurants.
1.7. A registered user shall be able to search for any other registered users.
1.8. A registered user shall be able to dislike a restaurant.
1.9. A registered user shall be able to dislike a cuisine.
1.10. A registered user who rates a restaurant shall have 'points' accumulated as their 'score'.
1.11. A registered user shall be able to see all their past reviews.
1.12. A registered user shall be able to see all their past restaurant ratings that they rated.
1.13. A registered user shall be able to see all the restaurants that they have visited.
1.14. A registered user shall be able to edit/manage their profile page.
1.15. A registered user shall be able to send a personal message to another registered user.
1.16. A registered user shall be able to add/follow other users.
1.17. A registered user shall be able to bookmark restaurants to their favorites list

2. ***Regular User (non-registered)***

2.1 A regular user shall be prompted to sign up for a new account.
2.2 A regular user shall be able to create many accounts.
2.3 A regular user shall have access to public content only, such as public restaurant pages.
2.4 A regular user shall not be able to post, leave reviews, or follow other users.

3. ***Account***

3.1. An account shall be created by one user.
3.2. An account shall be verified by the user.
3.3. An account shall have a type, 'basic', dedicated for new customers that just join
3.4. An account shall have a type, 'admin', dedicated for employees.
3.5. An account shall have a type, 'restaurant', dedicated for restaurant owners.
3.6. A 'basic' account type user shall receive a 'rank' once the user accumulated a certain amount of points/score.

3.7. A ‘basic’ account type shall be able to receive rank of ‘silver’, ‘gold’, or ‘platinum’ once the user accumulated a certain amount of points/score.

3.8. An account shall be created as ‘restaurant’ to be able to post a restaurant.

3.9. A ‘restaurant’ account shall be able to post many restaurants.

3.10. An account shall need verification upon registering for the first time.

4. ***Admin***

4.1. An account with a type of ‘admin’ shall be able to view all existing users and access to their account information.

4.2. An account with a type of ‘admin’ shall be able to add new restaurants.

4.3. An account with a type of ‘admin’ shall be able to temporarily/permanently remove existing restaurants.

5. ***Restaurant***

5.1. A restaurant shall have information such as brand name, address, location, website.

5.2. A restaurant shall have a rating.

5.3. A restaurant shall declare the type of cuisine they are serving.

5.4. A restaurant shall have its own profile page showcasing their brand image, products, rating, customer reviews and photos.

6. ***Friend/Followers***

6.1. Friends shall be able to see each other’s activities

7. ***Rating***

7.1. A user rating shall be uniquely associated with a single restaurant.

7.2. The overall rating of a restaurant shall be determined by averaging the ratings from all individual users.

7.3. A rating shall be able to be modified by the user.

7.4. A rating shall not be able to be deleted.

8. ***Review***

8.1. A review shall be uniquely associated with a single restaurant.

8.2. A review shall allow users to embed images.

8.3. A review shall have a maximum capacity for the amount of characters it can take.

8.4. A review shall be able to be updated or deleted by the author.

9. ***User Activity/History***

9.1. User activities like reviewing, rating, or recommending shall be assigned a dedicated point/score value.

9.2. User activities on each user shall be recorded.

9.3. Certain activities shall be visible to their friends/followers.

10. ***Address***

10.1. An address shall have basic address information such as street name, city name, phone number, zip code, country.

10.2. An address shall have one city.

11. ***Cities***

11.1. A city shall belong to one address.

12. ***Notifications***

12.1 A user shall receive notifications for receiving new messages

12.2 A user shall receive notifications for friend requests

12.3 A user shall receive notifications for restaurant recommendations.

12.4 A user shall receive notifications on updates about their reviews

13. ***Recommendation (Recs)***

13.1. High rated restaurants shall be recommended to users

14. ***FAQs***

14.1. A FAQ section shall be provided for users to help with any technical issues they may have

14.2 The FAQ section shall answer common technical questions.

Priority 2:

1. ***Point/Score***

6.1. Point/Score shall be shown as the main information in a user profile page.

6.2. Point/Score of every user in the platform shall be listed by index as 'leaderboard'.

6.3. Point/Score value shall be assigned to certain activities in the app.

6.4. A certain amount of Points/Score shall be provided to a user who rated a restaurant.

6.5. A certain amount of Points/Score shall be provided to a user who reviewed a restaurant.

6.6. A certain amount of Points/Score shall be provided to a user who recommended a restaurant.

2. ***ChatAssistance (chatbot customer service)***

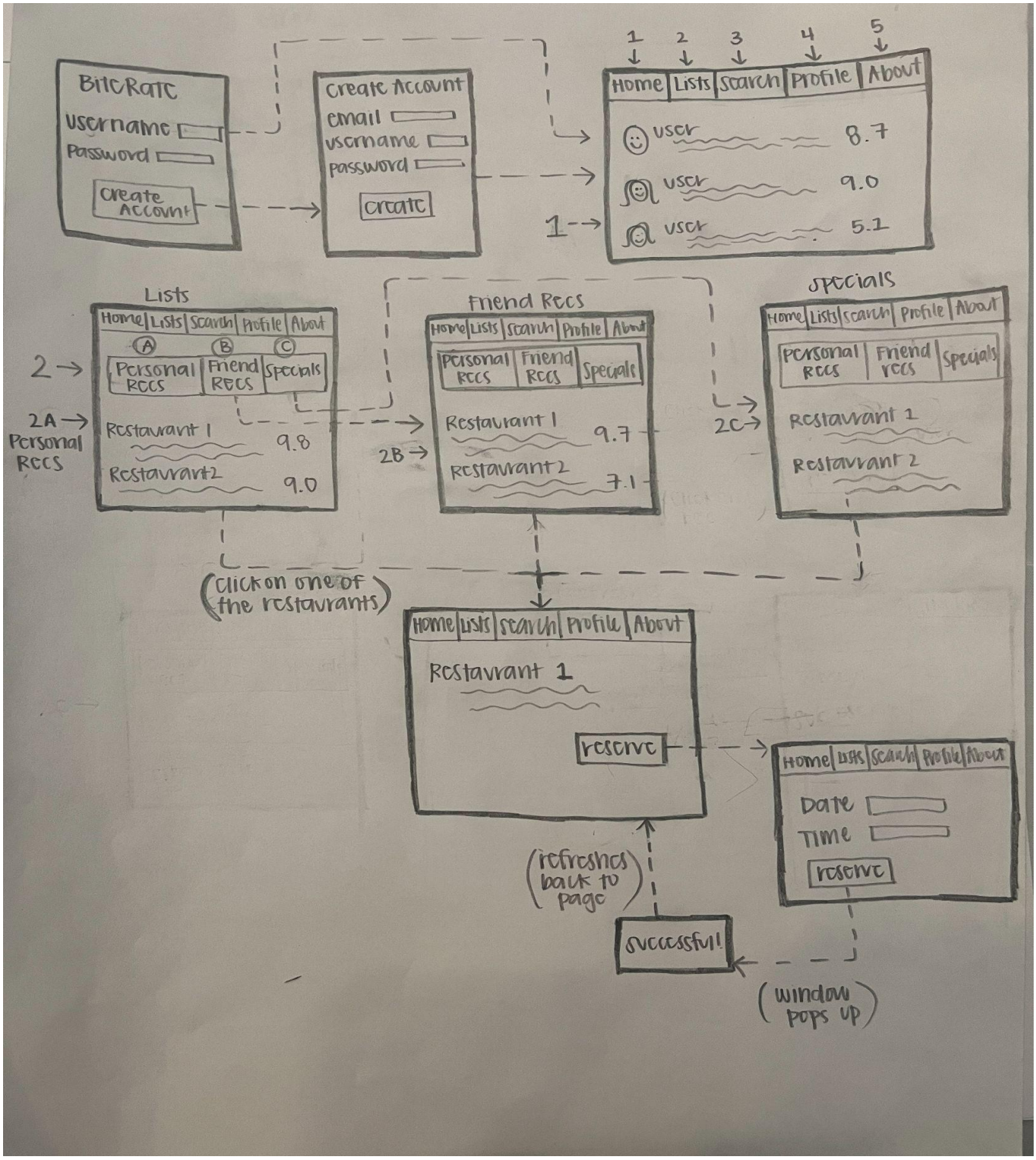
13.1. A chat assistance agent shall be able to serve many users.

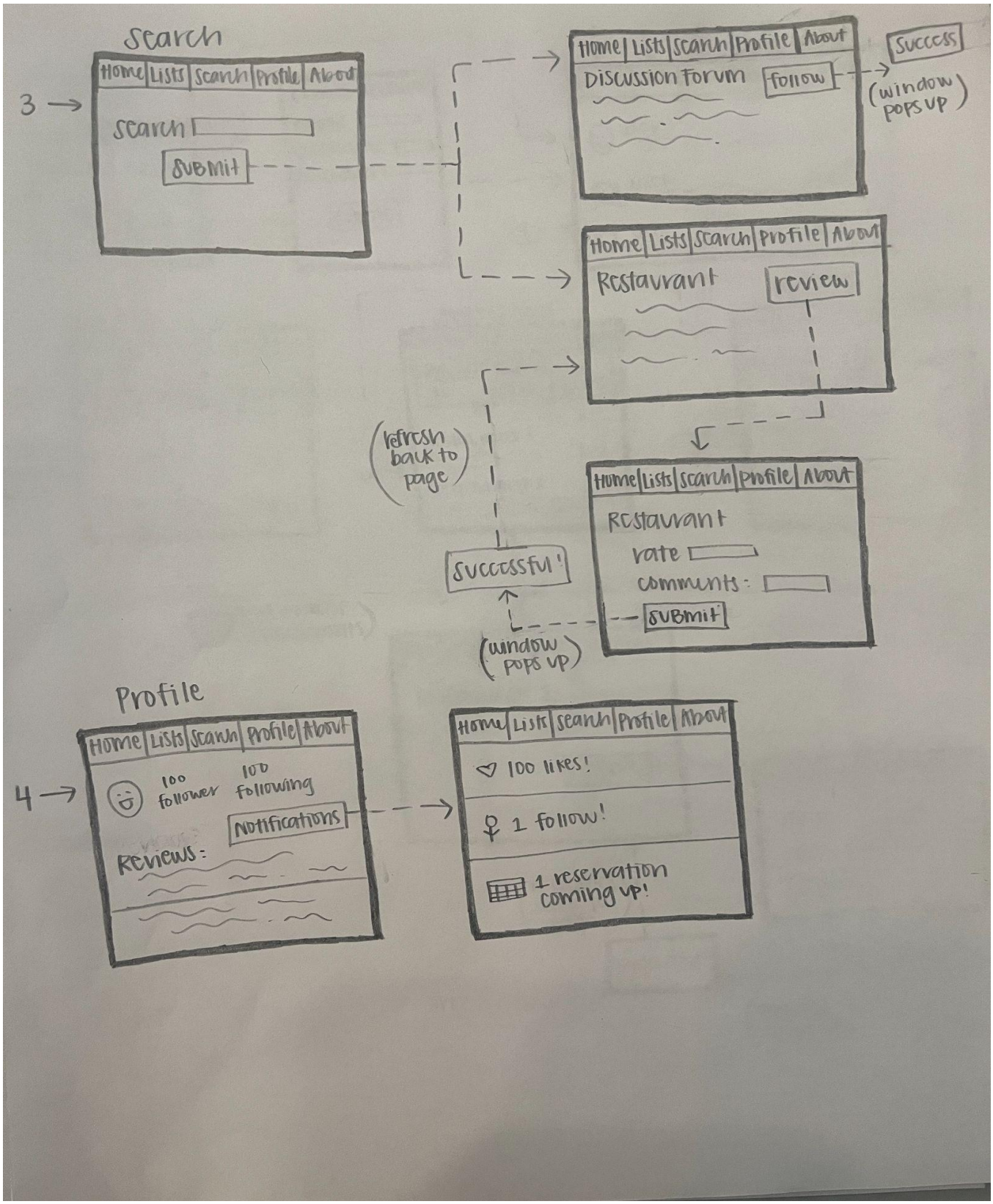
13.2. A chat assistance service shall allow users to ask about restaurants and/or the cuisine they like.

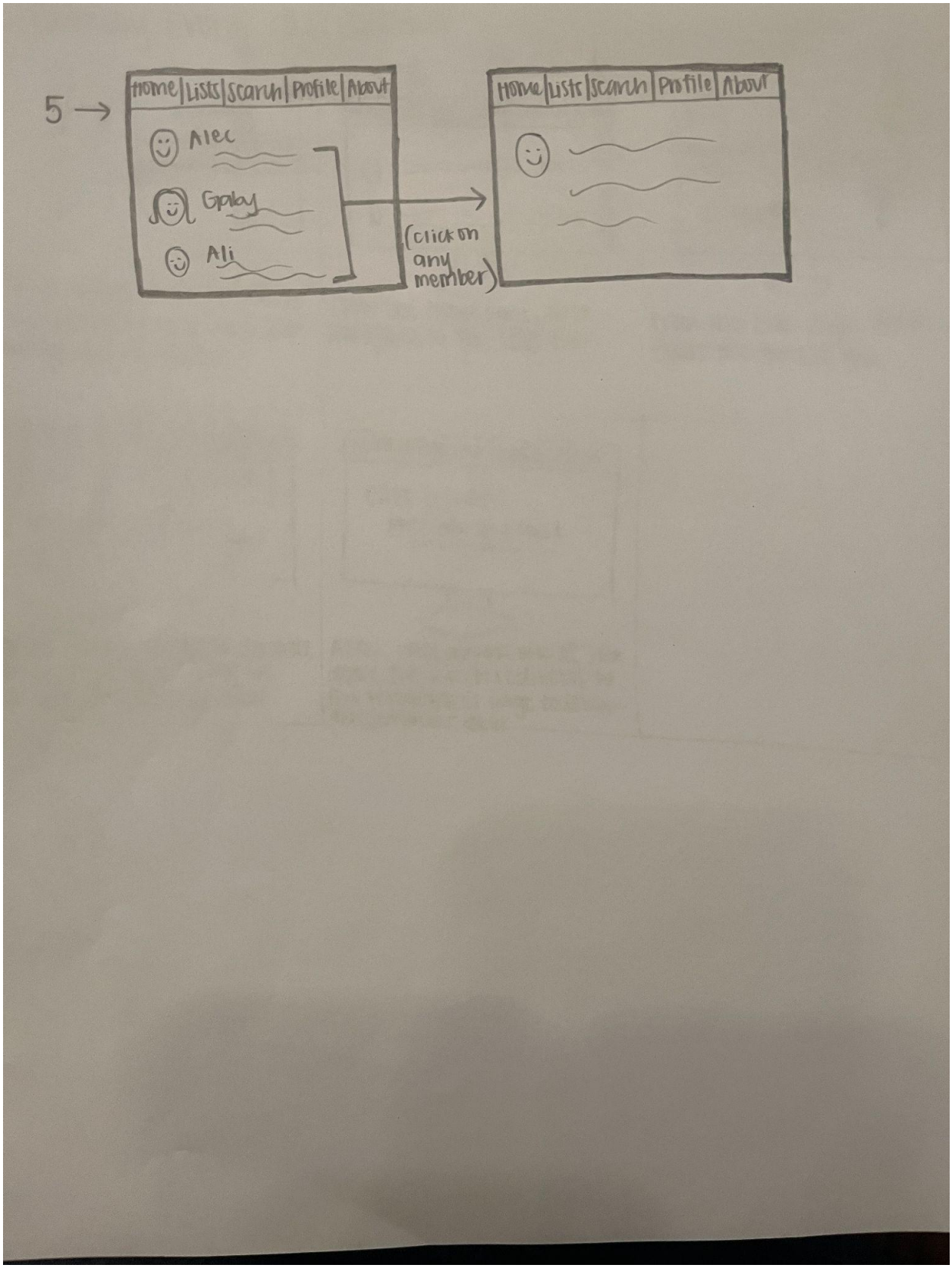
13.3. A chat assistance service shall be able to recommend restaurants to users based on cuisine, restaurant name, or location.

UI Mockups and Storyboards

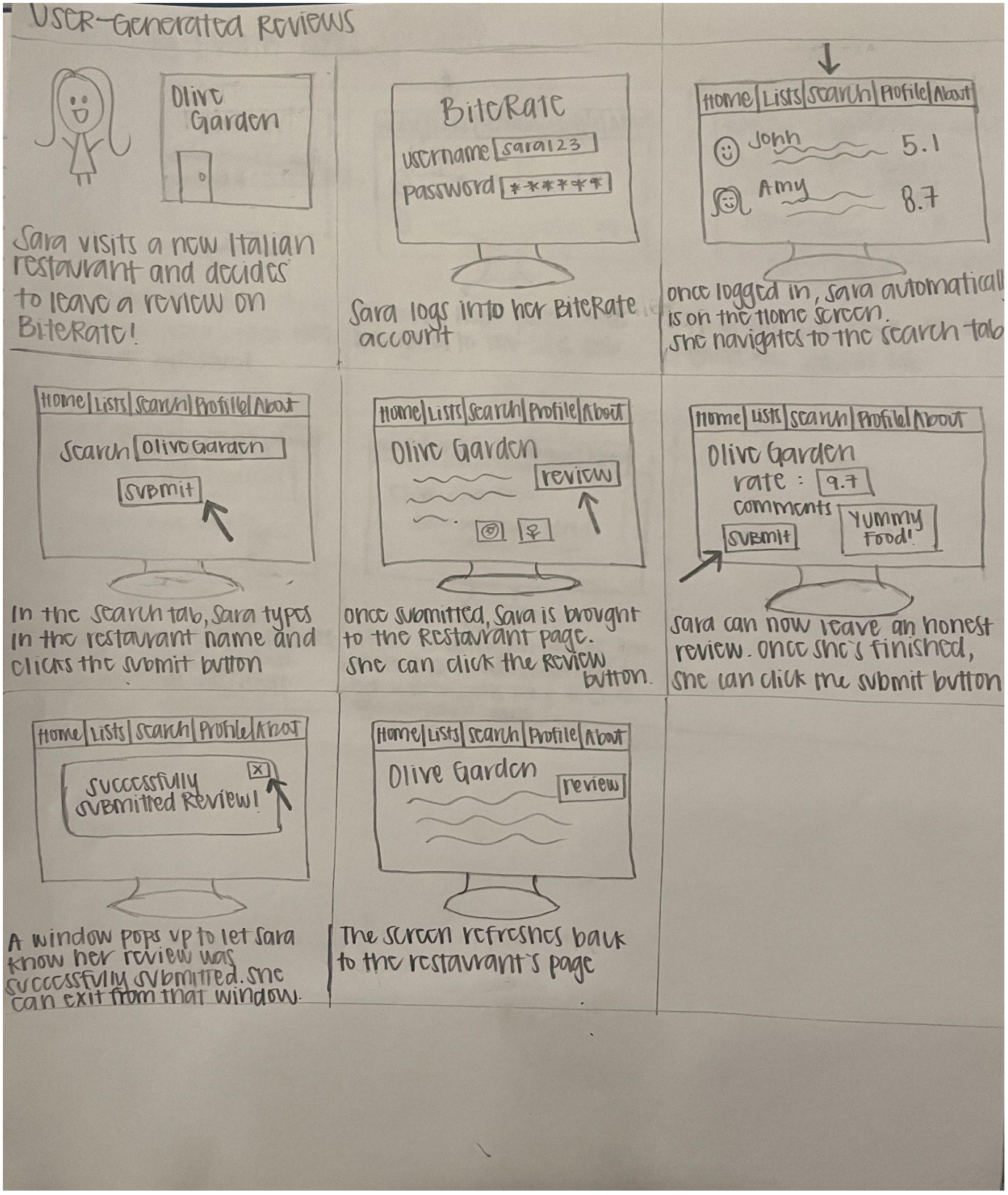
Overall Flow Mockup



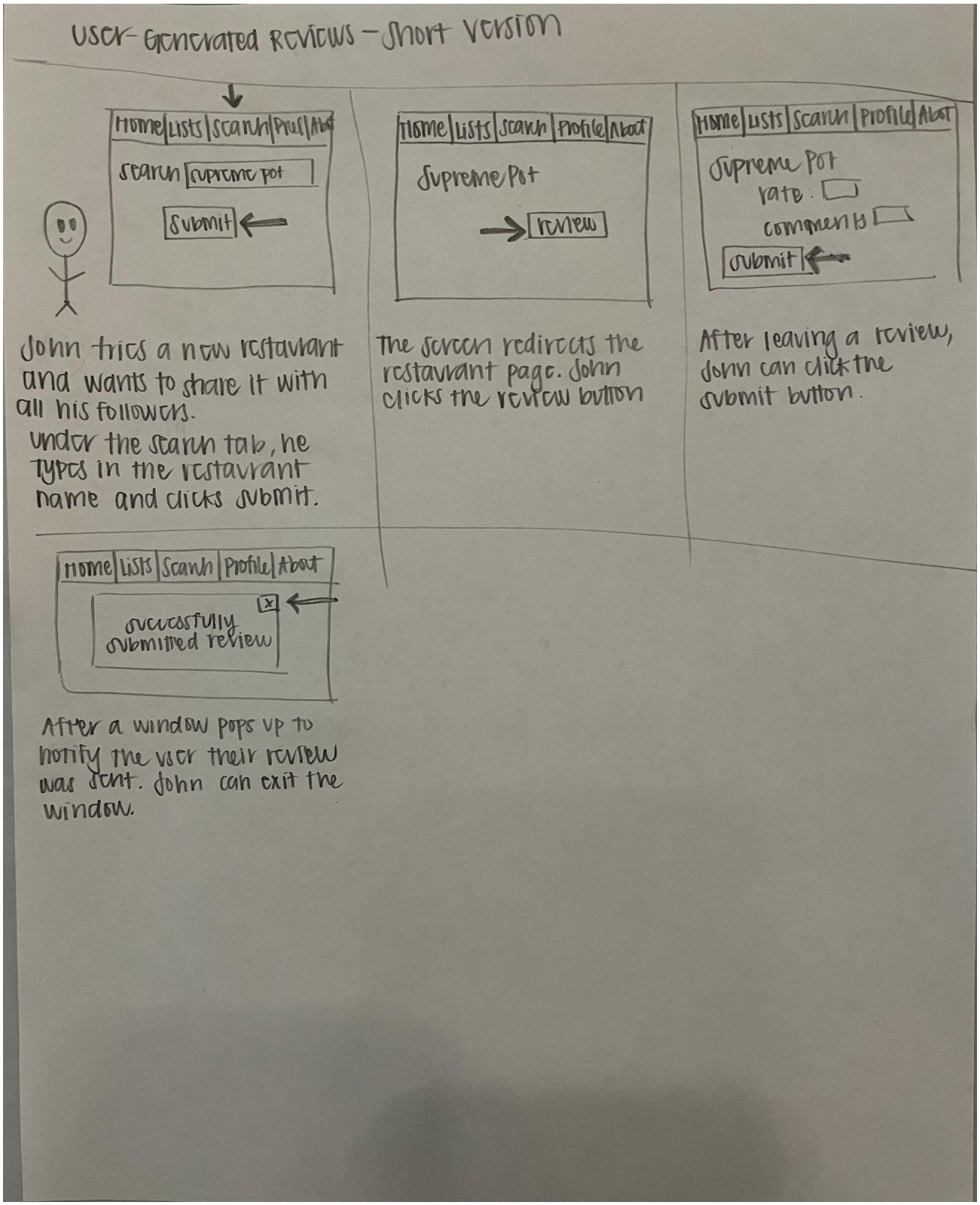

Search
Home Lists Search Profile About
3
search
submit
Home Lists Search Profile About
Discussion forum
follow
Success
(window pops up)
Home Lists Search Profile About
Restaurant
review
(refresh back to page)
Home Lists Search Profile About
Restaurant
rate
comments:
submit
Successful!
(window pops up)
Profile
Home Lists Search Profile About
4
100 followers
100 following
Notifications
Reviews:
Home Lists Search Profile About
100 likes!
1 follow!
1 reservation coming up!


5
Home
Lists
Search
Profile
About
Alec
Gaby
Ali
(click on any member)
Home
Lists
Search
Profile
About

Use Case: User-Generated Reviews

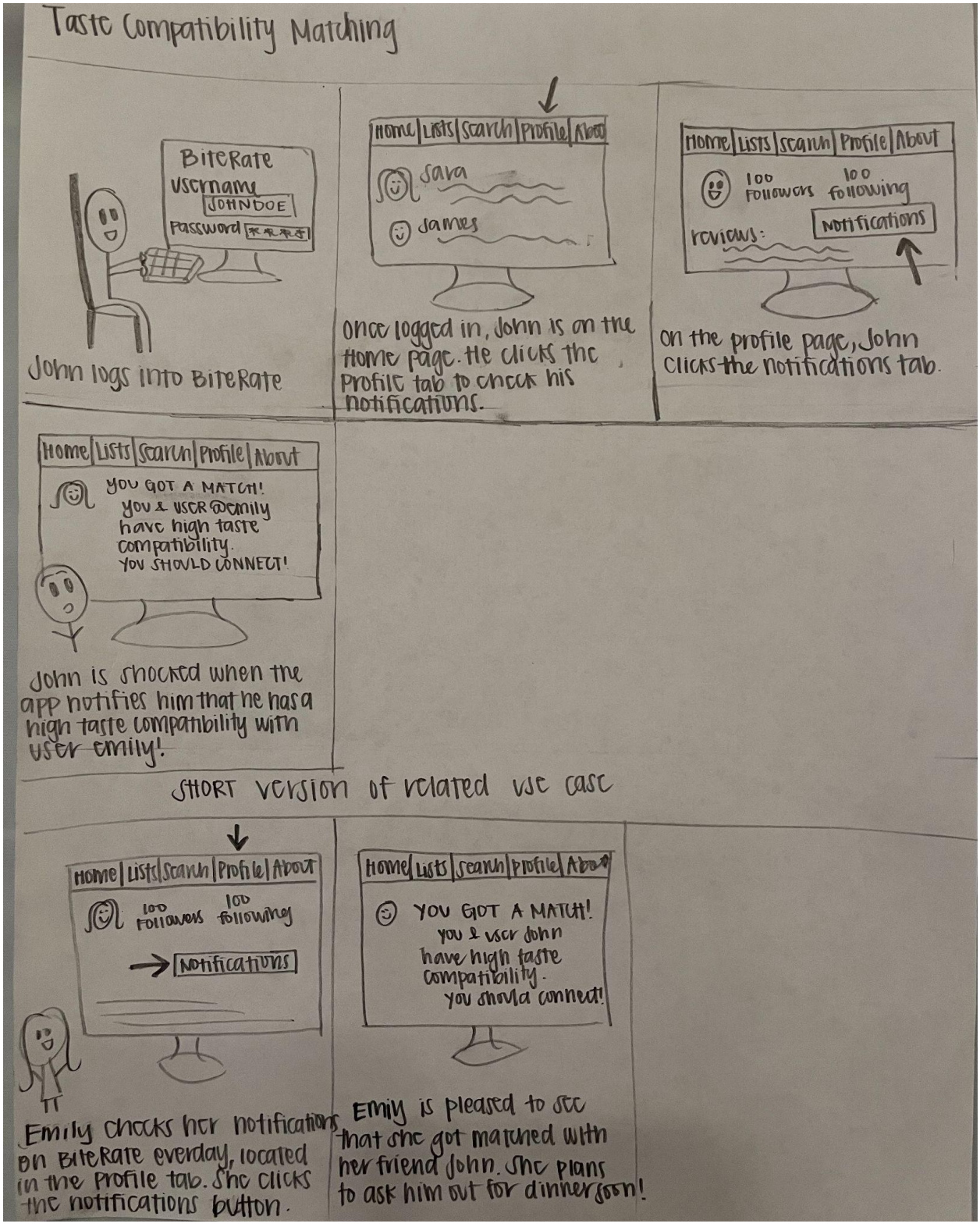


User-Generated Reviews - Shorter Version



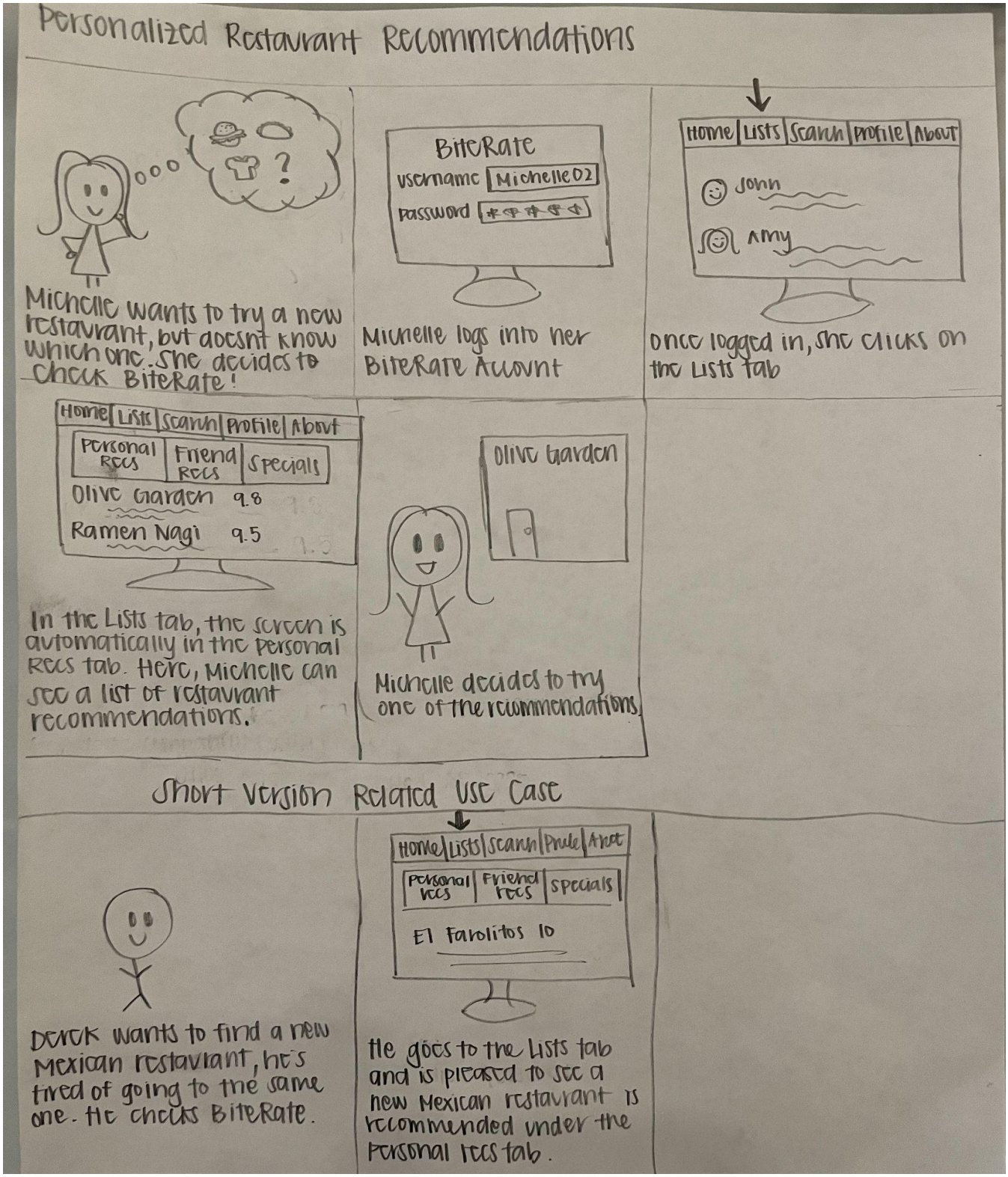
Use Case: Taste Compatibility Matching

(Short Version is on the bottom of the page)

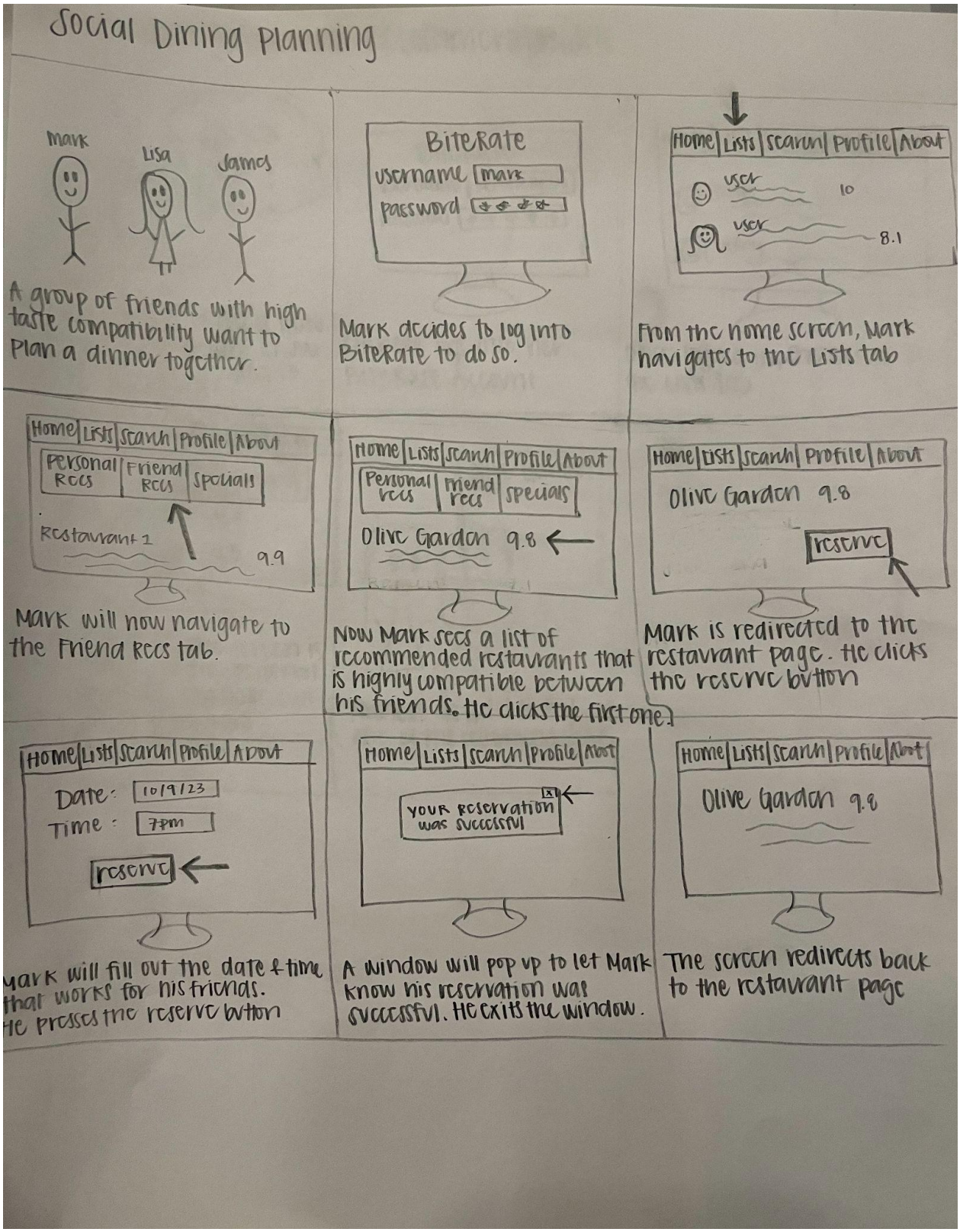


Use Case: Personalized Restaurant Recommendations

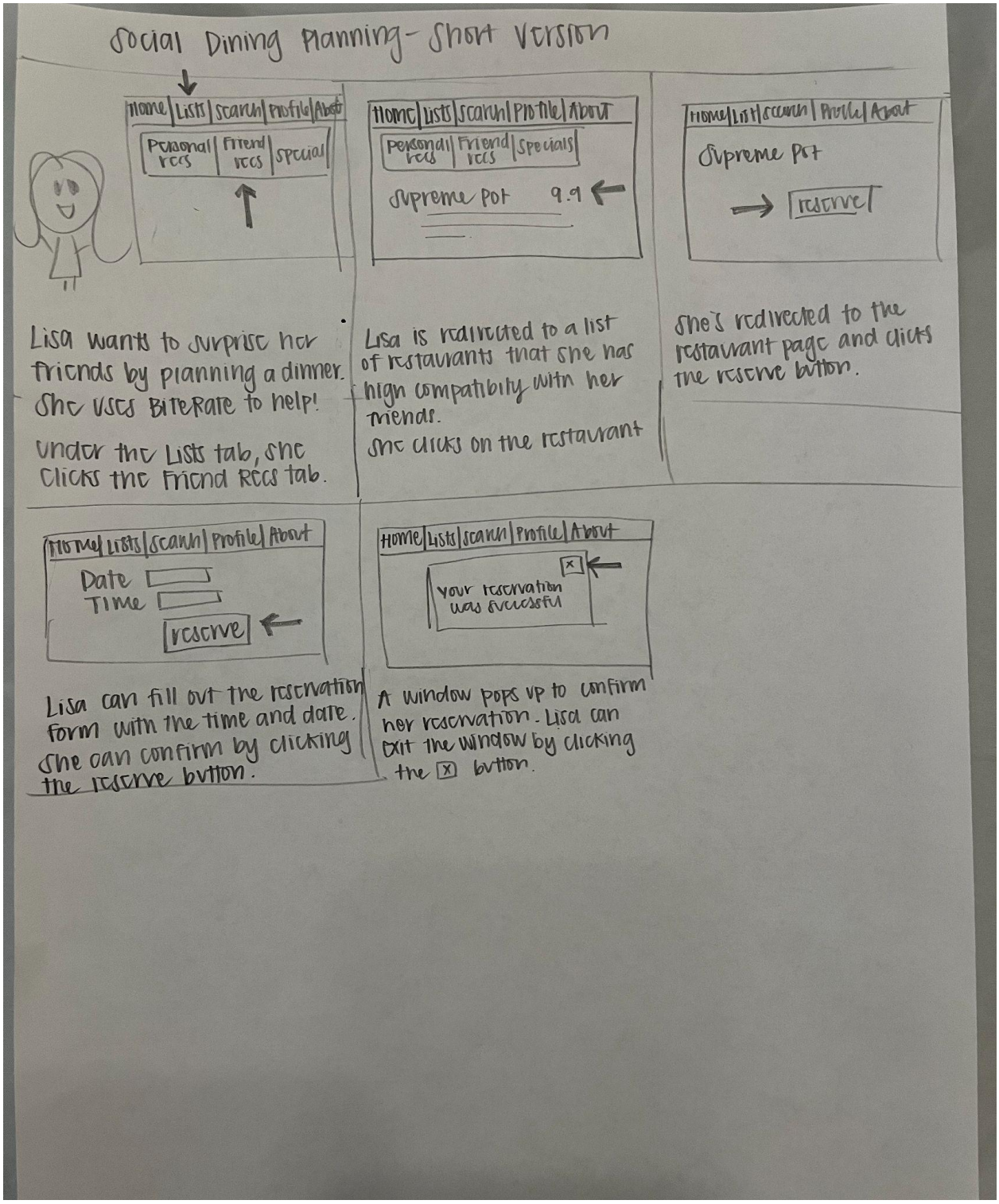
(Short Version is on the bottom of the page)



Use Case: Social Dining Planning

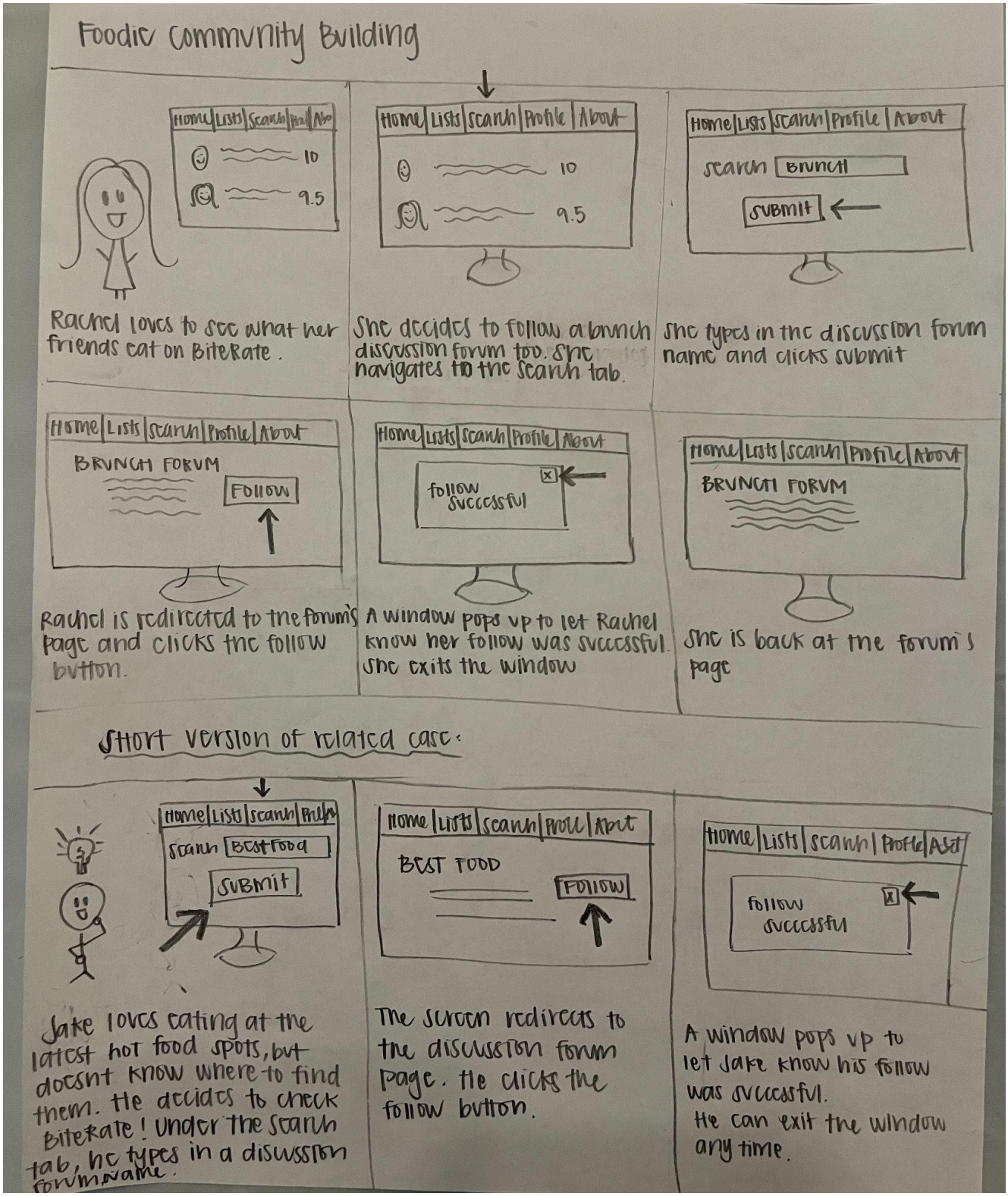


Social Dining Planning - Shorter Version



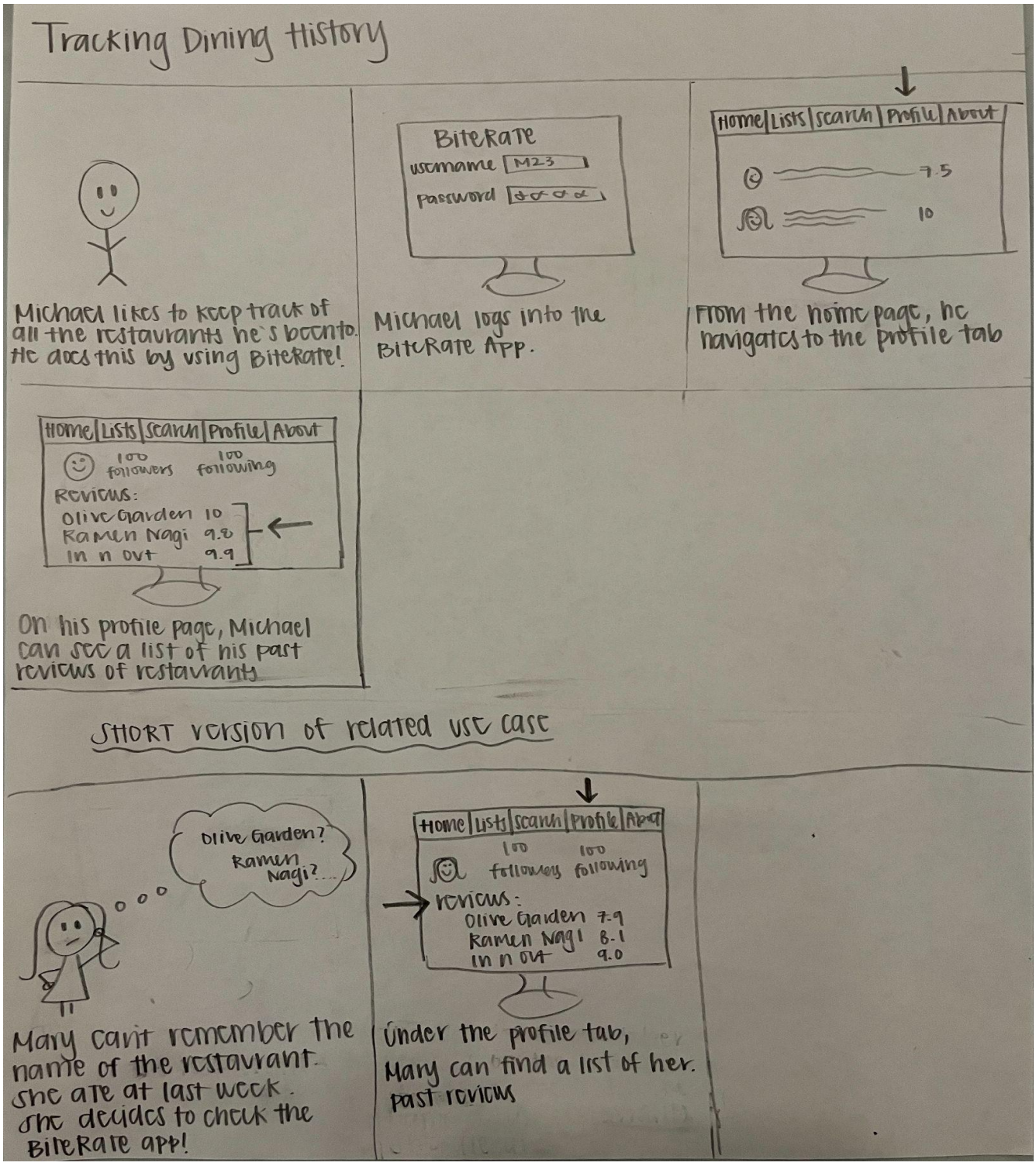
Use Case: Foodie Community Building

(Short Version is on the bottom of the page)



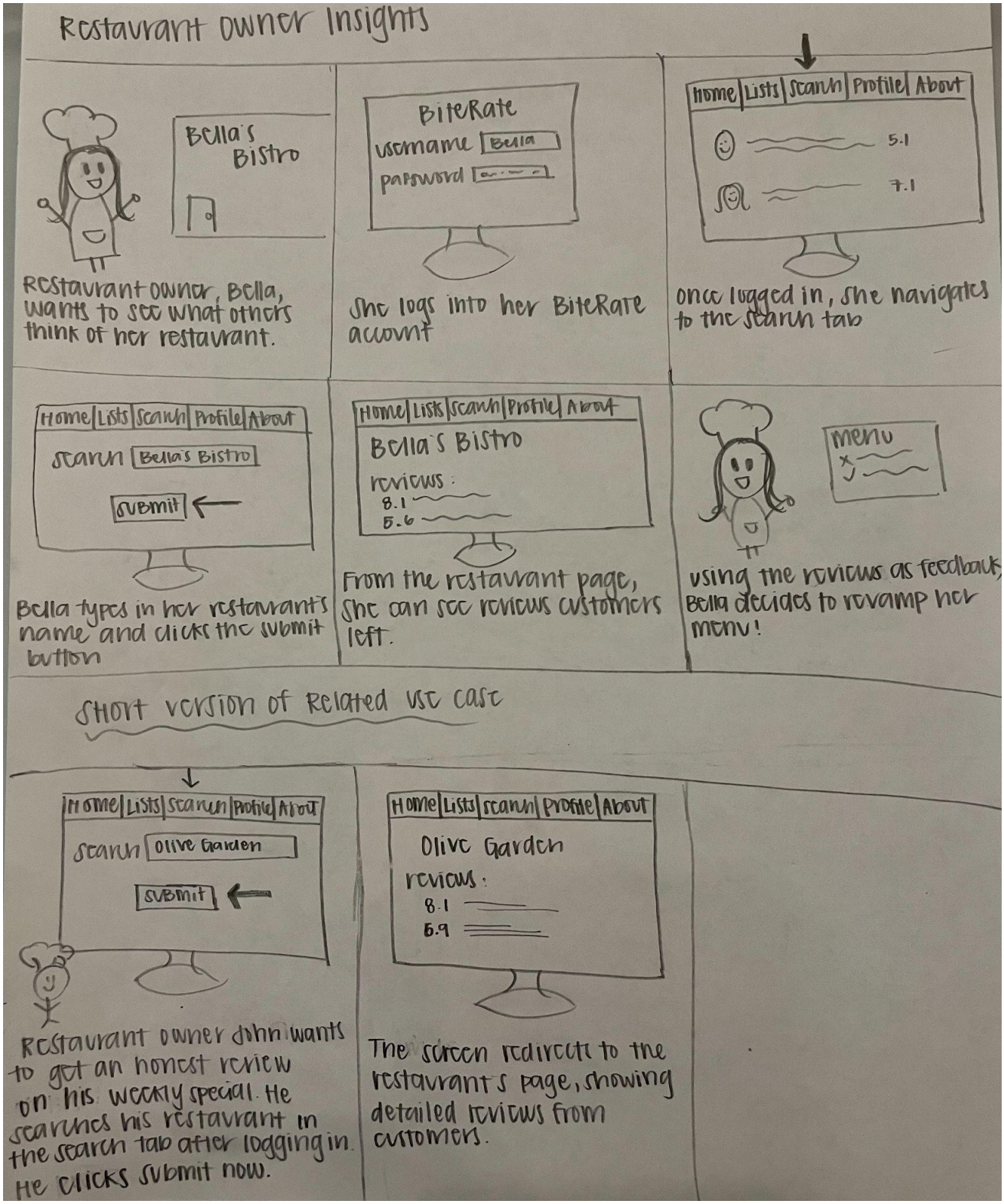
Use Case: Tracking Dining History

(Short Version is on the bottom of the page)



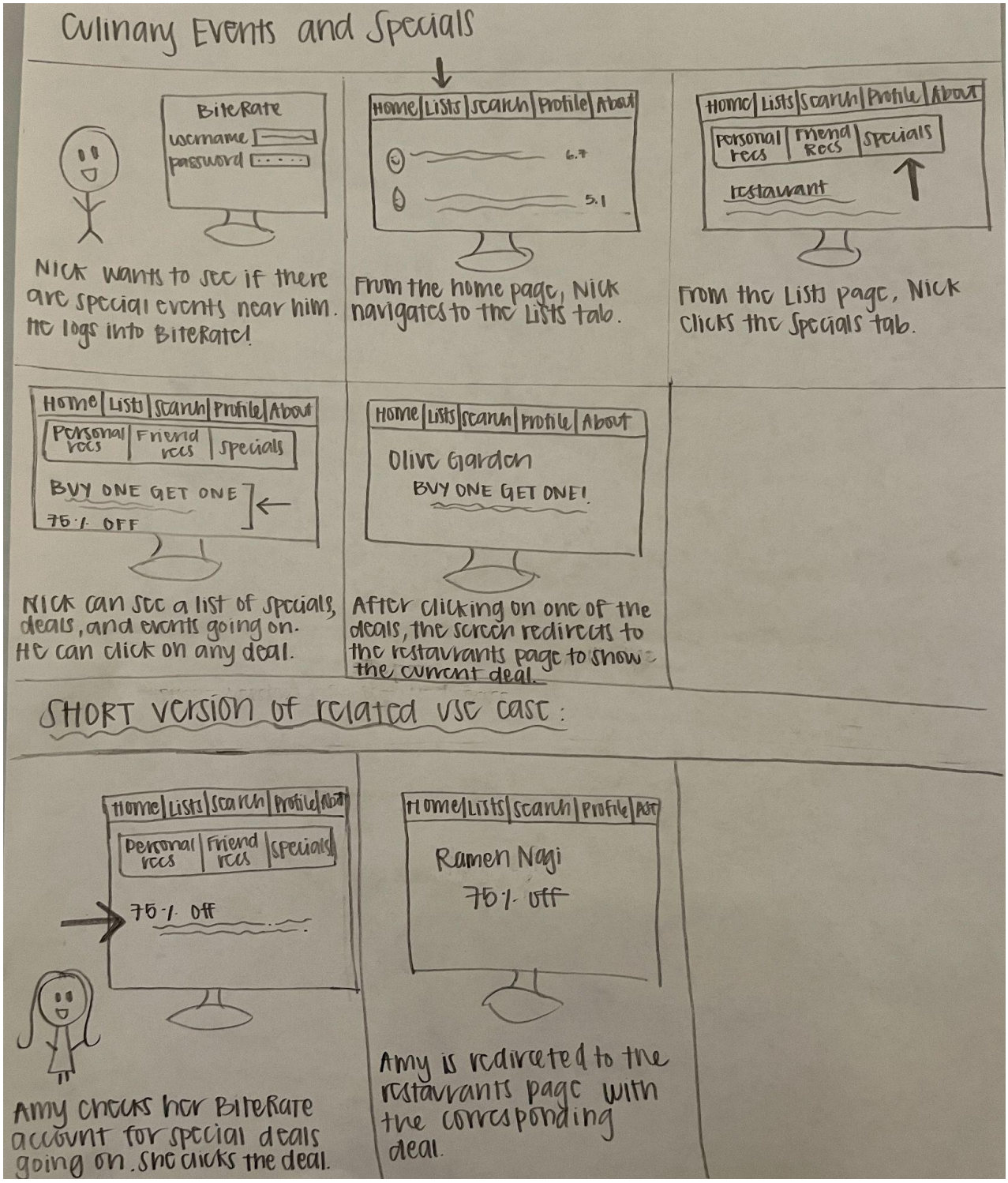
Use Case: Restaurant Owner Insights

(Short Version is on the bottom of the page)



Use Case: Culinary Events and Specials

(Short Version is on the bottom of the page)



High-Level Database Architecture and Organization

Database Requirements

- A user shall be able to create zero or many accounts.
- A registered user shall be able to rate zero or many restaurants.
- A registered user shall be able to post zero or one review for each restaurant.
- A registered user shall be able to do zero or one recommendation for each restaurant.
- A registered user shall be able to follow zero or many users.
- A restaurant account shall be able to post zero or one restaurant.
- A restaurant shall be able to post zero or many images.

Entity, Attributes, and Relationship Description

1. ***User (Strong)***
    - UserID: key, numeric
    - LastVisit: multivalue, timestamp

    ` Related to Account by one-to-many relationship.

2. ***Account (Strong)***
    - AccountID: key, numeric
    - User: key, numeric
    - Password: alphanumeric
    - FirstName: alphanumeric
    - LastName: alphanumeric
    - Email: alphanumeric
    - Address: key, alphanumeric
    - Phone: numeric
    - DateCreated: timestamp

    Related to User by many-to-one relationship.
    Related to Address by many-to-one relationship.
    Related to ChatAssistance by many-to-many relationships.
    Related to Admin, BasicAccount, RestaurantAccount by an inheritance one-to-one relationship.

3. ***Admin (Strong)***
    - AdminID: key, numeric

- Account: key, numeric
- Role: alphanumeric
- StartDate: timestamp
- EndDate: timestamp

Related to Account by an inheritance one-to-one relationship.

4. ***Basic Account (Weak)***
    - BasicAccountID: key, numeric
    - Account: key, numeric
    - BiteScore: numeric

    Related to Account by an inheritance one-to-one relationship.
    Related to RestaurantRecommendation, RestaurantRating, and RestaurantReview by many-to-many relationships.
    Related to Activity by many-to-many relationships.
    Followed by many-to-many relationships.

5. ***RestaurantAccount (Weak)***
    - RestaurantAccountID: key, numeric
    - Account: key, numeric

    Related to Account by an inheritance one-to-one relationship.
    Related to Restaurant as an associative entity between Restaurant and BasicAccount.

6. ***RestaurantOwner (Weak)***
    - RestaurantOwnerID: key, numeric
    - RestaurantAccount: key, numeric
    - Restaurant: key, numeric

    Related to Restaurant and BasicAccount as an associative entity.

7. ***Restaurant (Strong)***
    - RestaurantID: key, numeric
    - Name: alphanumeric
    - Cuisine: key, numeric
    - Address: key, numeric
    - OpenDate: timestamp
    - PostDate: timestamp

- Images: Array???

  Related to Address by many-to-one relationship.
  Related to RestaurantRecommendation, RestaurantRating, and RestaurantReview by many-to-many relationship.
  Related to Cuisine by many-to-one relationship.

8. RestaurantRating (Weak)

- RestaurantRatingID: key, numeric
- BasicAccountID: key, numeric
- Restaurant: key, numeric
- Rating: numeric
- RateDate: timestamp

9. RestaurantReview (Weak)

- RestaurantReviewID: key, numeric
- BasicAccountID: key, numeric
- Restaurant: key, numeric
- ReviewComment: alphanumeric
- ReviewDate: timestamp

10. RestaurantRecommendation (Weak)

- RestaurantRecommendationID: key, numeric
- BasicAccountID: key, numeric
- Restaurant: key, numeric

11. RestaurantImages (Weak)

- Image: key, numeric
- Restaurant: key, numeric
- Timestamp: timestamp

12. ImageURL (Strong)

- ImageID: key, numeric
- ImageURL: alphanumeric

13. Cuisine (Strong)

- CuisineID: key, numeric
- Name: alphanumeric
- CountryOrigin: alphanumeric

Related to Restaurant by a one-to-many relationship.

14. Address (Strong)

- AddressID: key, numeric
- AddressLine1: alphanumeric
- AddressLine2: alphanumeric
- City: alphanumeric
- State: alphanumeric
- Country: alphanumeric
- PostalCode: numeric

Related to Account by one-to-many relationship.
Related to Restaurant by one-to-many relationship.

15. UserActivity (Weak)

- UserActivityID: key, numeric
- BasicAccountID: key, numeric
- Activity: key, numeric

Related to BasicAccount and Activity acting as an associative entity between them.

16. Activity (Strong)

- ActivityID: key, numeric
- Name: alphanumeric

Related to BasicAccount by many-to-many relationships.

17. Follow (Strong)

- FollowerID: key, numeric
- BasicAccountID: key, numeric
- FollowerBasicAccountID: key, numeric
- DateFollow: timestamp

Related to BasicAccount by many-to-many relationships.

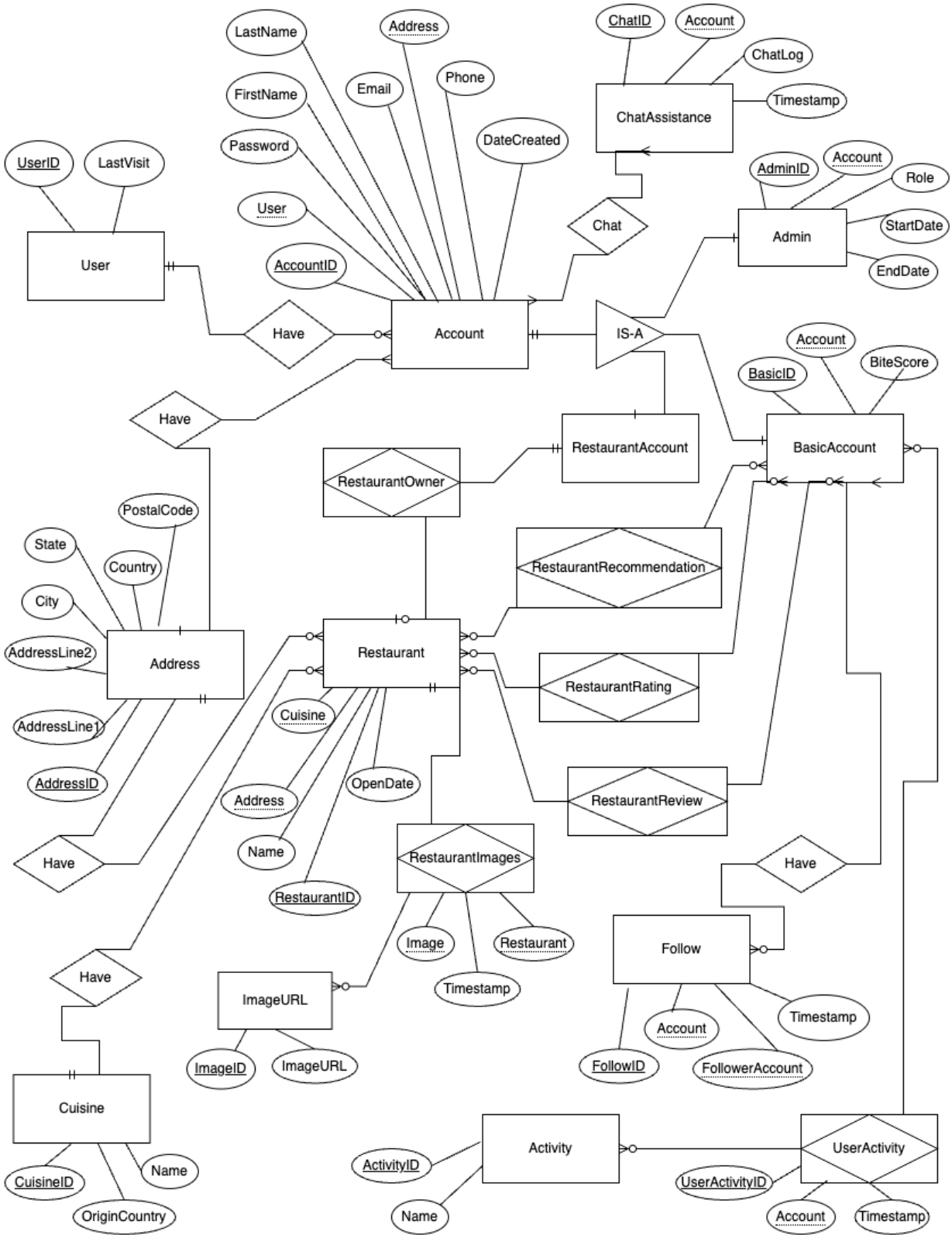
18. ChatAssistance (Strong)

- ChatID: key, numeric

- Account: key, numeric
- ChatLog: alphanumeric
- Timestamp: timestamp

  Related to Account by many-to-many relationship.

Entity Relationship Diagram (ERD)



What DBMS We Will Use

Based on what our team had discussed, all of us are more familiar with using mySQL for our relational database. And this is the main reason we decided to go with mySQL.

Media Storage

Based on what our team had discussed, we shall use DB BLOB to store images. But for this Milestone 2 checkpoint 2, Vertical SW Prototype, we have decided to keep it simple by storing images in the file system.

Search/Filter Architecture And Implementation

1. Search Algorithm
    - We shall mainly use mySQL 'LIKE' clauses for basic string/substring matching in our search.
2. Organization For User
    - We shall have a search bar on the home page to allow users to enter keywords to search for restaurants/users.
    - In addition to the search bar, we shall have optional filters next to the search bar for a more specific result.
3. Database Attributes For Search (Primary Search Field)
    - A user's first/last name.
    - A restaurant name.
    - A cuisine type.
    - A location or area.
4. MySQL Code Implementation/Examples

    When a user wants to search for a particular restaurant by name:

    ```
    SELECT * FROM restaurant WHERE name LIKE '%search_term%';
    ```

    When a user wants to search for a particular user by name:

    ```
    SELECT * FROM user WHERE name LIKE '%search_term%';
    ```

High-Level APIs and Main Algorithms

1. ***User Management API:***
- This API allows users to create accounts, log in, and manage their profiles.
- It includes features for user authentication, account creation, and profile editing.
- Additionally, it provides endpoints for searching and connecting with other users.

2. ***Restaurant Management API:***
- This API facilitates the creation, modification, and retrieval of restaurant-related information.
- Users can use this API to search for restaurants, view restaurant profiles, and submit ratings, reviews, and recommendations.

3. ***Point/Score System API:***
- The Point/Score API tracks user activities and calculates scores/points.
- It includes logic to assign scores for actions like rating, reviewing, and recommending restaurants.
- The API also provides a leaderboard to display user rankings.

4. ***Recommendation Engine:***
- This algorithm analyzes user behavior, such as ratings and reviews, to provide restaurant recommendations.
- It considers user preferences, historical data, and restaurant ratings to suggest suitable dining options.

5. ***User Activity Tracking API:***
- This API records user interactions and activities within the application.
- It stores data related to user actions, allowing users to see their own history and enabling the visibility of certain activities to friends/followers.

6. ***Chatbot Integration:***
- The chatbot integrates with a chat assistance service.
- It uses natural language processing (NLP) and algorithms to understand user queries about restaurants and cuisines.
- The chatbot then recommends restaurants based on user preferences and location.

7. ***Notification Service:***
- This API handles the generation and delivery of notifications to users.
- It sends notifications for events such as new messages, friend requests, restaurant recommendations, and updates on user reviews.

8. ***Ranking Algorithm:***

- The ranking algorithm calculates a restaurant's overall rating by averaging user ratings.
- It ensures that ratings are appropriately weighted and considers factors like recency and the number of ratings.

9. ***Geolocation API:***

- The geolocation API assists in determining a user's location and finding nearby restaurants.
- It may use geospatial algorithms and data to provide accurate location-based services.

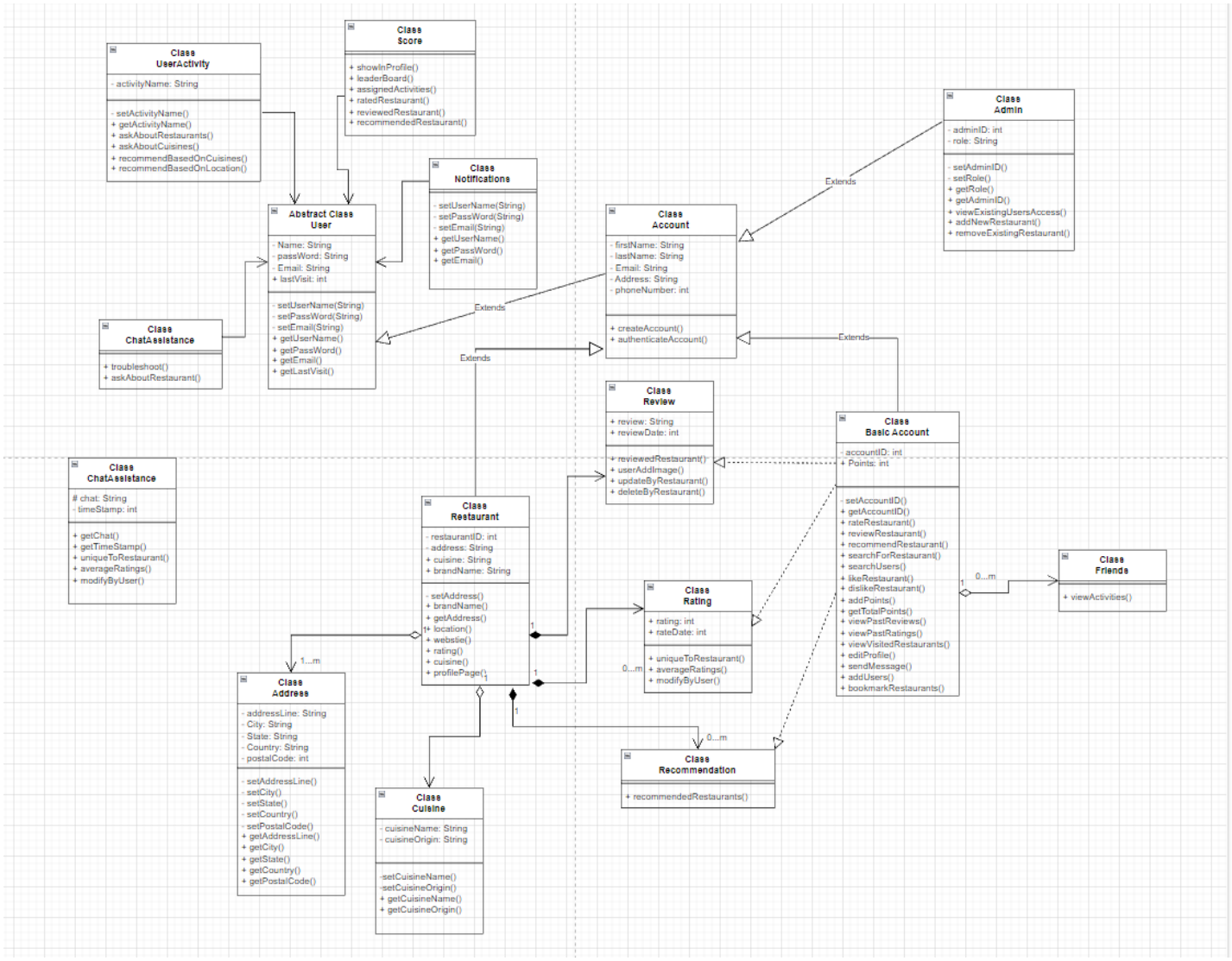
10. ***Data Storage and Retrieval:***

- The application may utilize MySQL (8.1) as the database system to store and retrieve user data, restaurant information, reviews, and more.

11. ***User Interaction API:***

- This API manages interactions between users, including friend requests, messaging, and following other users.
- It facilitates communication and social networking within the application.

High-Level UML Diagrams



High-Level Application Network and Deployment Diagrams

Identify *Actual* Key Risks for the Project

Skills Risks

1. **Inadequate Documentation Skills**

Risk: Our team has a lack of experience in documenting, as for most of us, this is our first time documenting the progression of projects. This is an actual risk since we don't possess the right skills to give adequate documentation, which can lead to a deduction of points on our milestone.

Solution: To address this issue, our team is using the feedback given on Milestone 1 to evaluate the mistakes made in our past work. By doing so, we will improve in the future. We are also carefully reading the instructions on the Milestone 2 assignment, to ensure we will not miss any requirements.

2. **Using an Unfamiliar Framework**

Risk: The framework our team is using for the frontend of our web application is Angular. This particular framework is unfamiliar to most of our team as we mostly worked with React. This can cause bugs/errors in our program since we don't have the knowledge to implement a web application using Angular.

Solution: Our team made the decision to learn the basics of Angular by watching tutorials online and reading the necessary documentation. Due to the project's time constraint, our team decided to learn only the basic functionalities of Angular, instead of diving deep into all its complexities. By doing so, we have enough knowledge to get our project into production while keeping in mind the project's time constraint.

Schedule Risks

1. **Schedule Conflicts Related to School:**

Risk: Most of the team members have different class schedules which causes our free time to overlap. This makes it difficult to schedule additional team meetings to work on the project. This is an actual risk because this can cause a delay in production for our project.

Solution: The team has a set meeting time every Wednesday, which is mandatory. For additional team meetings, our team decided to meet via Discord as needed. This ensures that we have enough time to finish our project.

2. **Underestimating Time Needed**

Risk: For Milestone 1, our team made the mistake of underestimating the time needed to finish the milestone. Our team is at risk of making the same mistake, where we do not allocate

enough time to finish all parts of the project. This is an actual risk because it can cause a deduction of points in our Milestone.

Solution: Our team regularly updates each other on the progress of their work through Discord. Also, our team lead assigned a one-week deadline for the documentation, to ensure that we have enough time to work on the prototype. By managing our time wisely, we have enough time to finish the project.

Technical Risks

1. **AWS/MySQL Connection Issue**

Risk: Our team has recurring issues connecting our AWS host to the MySQL database. For this reason, we had a project delay since the database needed to be connected before we could start integrating other parts.

Solution: Our team discussed the issue during class, through Discord, and during team meetings. We accessed the issue, pinpointed the problem, and came up with ideas to resolve it.

2. **Integration Risks**

Risk: For the prototype, our team split up the work between the backend and frontend. This may cause problems in integrating the separated software into one, causing bugs and errors.

Solution: Although we cannot predict integration errors, we can try our best to solve them earlier if it happens. Our team decided to start integrating the frontend and backend early before the due date, to solve any errors if they arise. By doing so, we will have enough time to resolve it.

Teamwork Risks

1. **Missing Team Meetings**

Risk: During team meetings, we exchange important information regarding the project. Missing a team meeting can throw the members out of sync causing miscommunication on the project's objectives and individual responsibilities.

Solution: Our team agreed on a mandatory team meeting every Wednesday. We also hold extra team meetings as needed. In addition to team meetings, our group is in constant contact to relay any questions, concerns, or information we may have.

2. **Failing to Finish Responsibilities**

Risk: Each member is given their own task to finish by the due date, but there is a potential risk of someone missing their individual deadline. This will cause a delay in the production of the project.

Solution: Our team lead assigned roles and tasks early on in the project, so each individual has a clear idea of the assignment. Our team also messages each other via Discord on the progress of our tasks. We also hold weekly team meetings to discuss the overall progression of our project.

Legal/Content Risks

1. **Copyright Violations**

Risks: Our team is at risk of copyright issues when gathering the necessary resources for the project. One example could be gathering pictures to display on our web application.

Solution: Our team decided to be careful when downloading images from the internet as most pictures are copyrighted. To do this, we used websites like Unsplash and Pexels to download copyright-free images.

Project Management

For M2 tasks, we primarily utilized Notion, a project management and collaboration platform. Notion allowed us to create a structured workspace where we could document project goals, functional requirements, and task lists. It served as a central hub for organizing and tracking our progress throughout Milestone 2 (M2).

We used Notion's features, such as tables, Kanban boards, and task databases, to maintain a clear overview of our tasks, assign responsibilities, and monitor the status of each task. This provided us with a collaborative environment to discuss and prioritize work efficiently. Moving forward, we recognize the importance of adopting a unified task management system that provides a holistic view of all tasks and their statuses. We plan to also integrate Trello.

Trello offers an intuitive dashboard view that aligns well with our task management needs. It enables us to create boards for different aspects of the project, assign tasks to team members, set deadlines, and track progress visually. By implementing Trello, we aim to streamline task management and improve transparency across the team, ensuring that future tasks are planned, executed, and monitored effectively.

Detailed List of Contributions

Alec Nagal: 7/10
- Worked on API/Main Algorithms: I was responsible for designing and implementing the core algorithms and API (Application Programming Interface) that the application relies on. This includes the logic that drives the key functionalities of the application.
- Project Management: I took on a project management role, overseeing the coordination and progress of the project. This involves setting milestones, tracking progress, and ensuring that the team stays on schedule.
- Frontend Development: Alec worked on the user interface (UI) and user experience (UX) aspects of the application, ensuring that it is visually appealing and user-friendly.

Ali Alsharif: 7/10
- UML Diagram: Ali was responsible for creating UML (Unified Modeling Language) diagrams, which are essential for visualizing the software architecture, data structures, and relationships within the application.
- Frontend Development: Ali contributed to the development of the frontend, working on the presentation layer of the application to ensure a seamless user experience.

Gerry Putra: 7/10
- High-Level Database Design: Gerry worked on the high-level database design, which involves defining the overall structure and organization of the database system that the application uses to store and retrieve data.
- Network/Deployment: Gerry was responsible for configuring and setting up the network and deployment environment for the application, ensuring that it can be accessed and used effectively.

Gabriella Arcilla: 7/10
- UI Mockups: Gabriella created mockups for the user interface, providing visual representations of how the application's screens and components should look.
- Risk Identification: Gabriella played a crucial role in identifying potential risks and challenges that the project may face, allowing the team to proactively address them.
- Document Revisions: Gabriella contributed to the revision of both the milestone 1 and milestone 2 documents, ensuring that they are accurate and up-to-date.

Kenneth Pang: 7/10
- Prioritization of Functional Requirements: Kenneth was responsible for prioritizing the functional requirements of the application, helping the team focus on the most critical features and functionalities.

- Database: Kenneth worked on the database aspects of the project, including data modeling and management, to ensure efficient data storage and retrieval.

Robel Ayelew: 7/10

- Data Definitions: Robel was in charge of defining the data structures and data schemas used in the application, ensuring consistency and integrity of data.
- Frontend Development: Robel contributed to the development of the frontend, working on the user interface to create a responsive and user-friendly design.

James Lu: 7/10

- Document Revision: James was responsible for revising the milestone 1 document, ensuring that it is well-written, coherent, and accurately reflects the project's status and goals.

Robin Reyes: 7/10

- Document Revision: Robin contributed to the revision of the milestone 1 document, ensuring its clarity and completeness.
- Main Use Case Diagram: Robin created the main use case diagram, which illustrates how users interact with the application and the various use cases it supports.