Route-Constraint Group Shopping Optimization

Final Report

Team Number: 34
Client: Goce Trajcevski
Advisers: Goce Trajcevski, Ashfaq Khokhar
Team Members/Roles:

Tavion Yrjo - Meeting Scribe, Backend Engineer
Colin Willenborg - Frontend Engineer
Erich Brandt - Web Developer
Elizabeth Strzelczyk - Web Developer
Christian Baer - Backend Engineer, Data Analyst
Colin Thurston - Trello, Tester, Algorithm Developer

Team email: sdmay21-34@iastate.edu
Team website: http://sdmay21-34.sd.ece.iastate.edu

Revised: 4/22/21

Executive Summary

Development Standards & Practices Used

- Agile: We used an Agile-like methodology for team development. By using meetings every week to assign new tasks as well as summarize work done, we were able to efficiently control development time.
- Version Control: The main method for version control was GitHub. By using GitHub we were able to accurately track any changes made to the project as well as revert to older builds if something catastrophic occurred.
- IEEE/ISO/IEC 14764-2006 Software Life Cycle Maintenance Standards: The team used maintenance standards to ensure that updates to the software would be overall beneficial to the project and not break other functionality.
- IEEE 1008-1987 Software Testing Standards: We used unit tests as well as functional testing to ensure that the software functions as intended.

Summary of Requirements

- A route to drive
- Mobile and desktop application
- Database full of store locations and items with their prices
- Optimized by user spending and distance
- Multiple users

Applicable Courses from Iowa State University

- Computer Science 309: Software Development Practices
- Computer Science 319: Software Construction and User Interface
- Computer Science 327: Advanced Programming Techniques
- Software Engineering 329: Software Project Management
- Software Engineering 339: Software Architecture and Design
- Computer Science 363: Intro to Database Management Systems

New Skills/Knowledge acquired that was not taught in

- Web API
- Kotlin
- React

List of Figures

1 INTRODUCTION

1.1 ACKNOWLEDGEMENT

The Route-Constrained Group Shopping Optimization team would like to thank both our advisors, Professor Goce Trajcevski and Ashfaq Khokhar for meeting with us on a weekly basis and helping guide us through the design process of this project. We would also like to thank the Iowa State University College of Engineering for giving our team access to professional guidance, resources and experts.

1.2 PROBLEM AND PROJECT STATEMENT

The goal of this project is to design and implement an application for both mobile and web platforms that would help registered groups of users (e.g., family members, members of a social circle, co-workers, etc.) coordinate and optimize their respective shopping routes for a given shopping list. In the context of this project, optimization is with respect to the length of an individual user's route, as part of collective efforts to complete a group-based purchase of a set of items. For example, group members will be able to add constraints such as who will purchase dairy and bread products vs. who (else) would purchase beverages - to further customize their shopping routes by including things like the group member's starting location, the distance from the store, and others. The application will then generate the optimal shopping route for that group.

The problem will involve constructing algorithms to determine the optimal shopping paths, getting data for item prices and store locations, and the constraints. The solution for both mobile and web will be the same. A group is defined as a set of individuals that share the same shopping list. In the application, users can join a group with their family, friends, coworkers, etc. Any group member can update the shopping list. To make the route a user will select the items they want to shop for and this will update other members of the group about what items are left to be purchased. The application will then check the group's shopping list to find items within the group's maximum distance they are willing to travel. Through a map system the route would be shown and take into account the distance constraint. With all of this the application will output an optimized route for all group members.

1.3 OPERATIONAL ENVIRONMENT

Since this project will have a mobile and web application, the physical environment will depend on which platform the user is currently accessing. On the cyber side, we would use online systems and local systems to create the application and keep it maintained.

1.4 REQUIREMENTS

The Route-Constrained Group Shopping Optimization project will have constraints to optimize the user experience. The constraints will limit the scope of the problem to make the algorithms more efficient. Additionally, there will be functional and nonfunctional requirements that our project aims to meet in order to fulfill user needs.

Constraints

- Radius of the map of stores and locations
- The time it takes to travel to different stores
- Starting the trip from home vs. varying locations
- Start time of the trip

Functional Requirements

- Store location accuracy
- Outputting the closest store with desired items with respect to distance/time to travel
- Output fastest travel time to any given store at desired start time

Nonfunctional Requirements

- Routes must generate in real time
- SQL Data must be in real time
- Application must be intuitive and easy to read

1.5 INTENDED USERS AND USES

The main user for this application is a group who wants to minimize their distance traveled and money spent while shopping.The final result is a route that is optimized for a single group member on spending, and the constraint of distance, for efficient shopping. Figure 1 shown below is a use case diagram for the use cases of the project.

Users

- Individuals
- Group members

Uses

- Join a group with family members, coworkers, friends, etc.
- See shopping list of user and group members
- Add or remove items from shopping list
- See location on a map
- Have applications access user location
- Show their optimized route

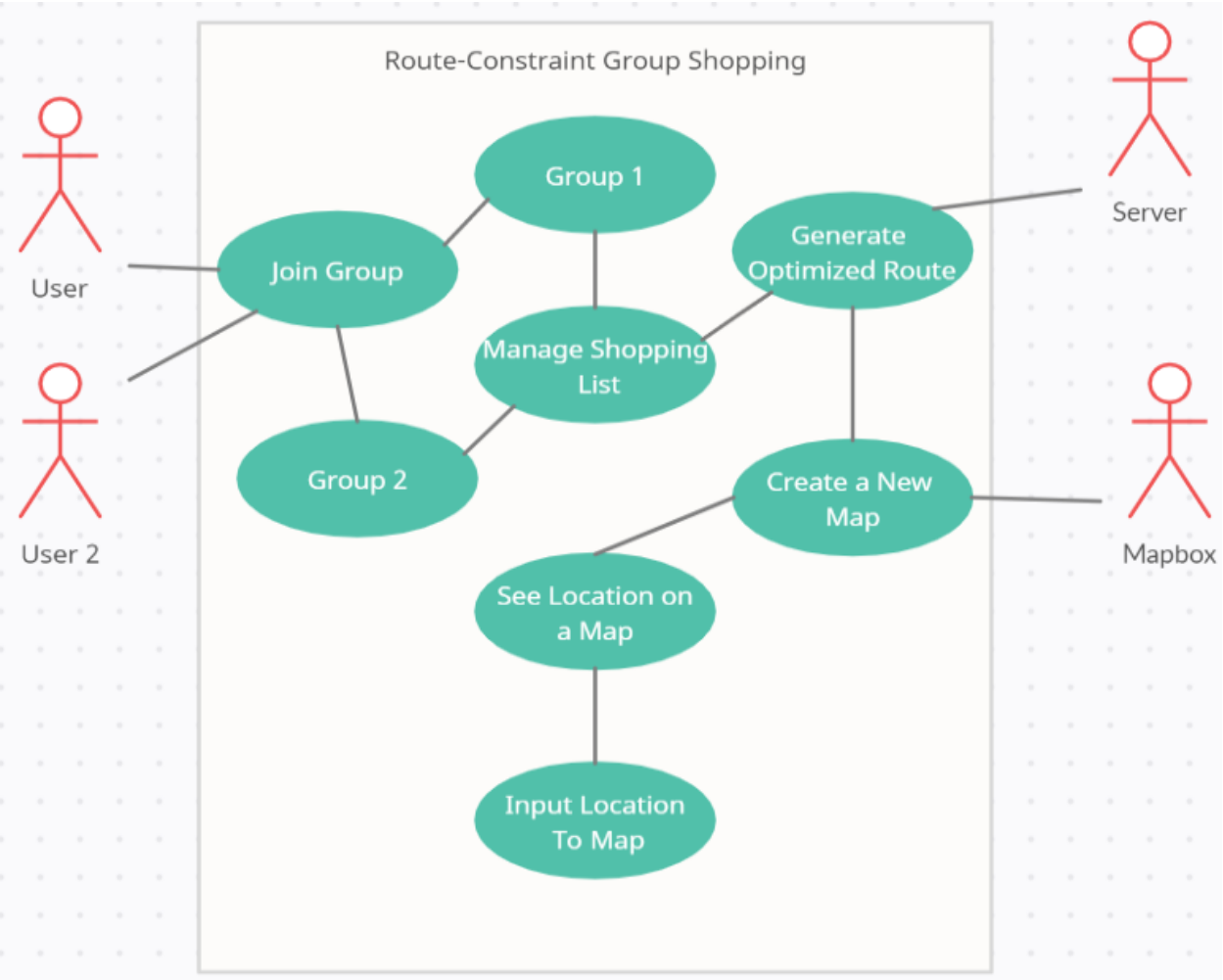


Figure 1: Use Case Diagram

1.6 ASSUMPTIONS AND LIMITATIONS

The assumptions and limitations will assist in the decision making process for this project. Assumptions help create a more defined idea for the intended user, as well as also defining the limitations of this project.

Assumptions:

- The customer must create an account to be routed to stores
- The customer has a phone or PC to access this application
- The customer has access to an internet connection
- Maximum number of users is unknown, however the project can be scaled to accommodate more users
- Multiple users can join a particular group and collaboratively update the shopping lists
-

Limitations:

- This application will only work inside of the United States
- Routes and Maps will be generated using MapboxAPI
- Loading times will be determined by the quality of the users internet connection
- Loading times will be affected by the size of the user's list
- The webscraper will only input the top 10 results for any given item into the database

1.7 END PRODUCT AND DELIVERABLES

Goal	Description
User log in	A user can log into both applications using a username and password.
User registration	A user can register a new account to the application.
Adding items to list	A user needs to be able to add items to a specific list.
Deleting items from list	A user can click on an item in a shopping list to remove the item from that list.
Joining groups	A user can join a group.
Finding items	A user can search for an item from our list of items.
Route generation	The user selects items and depending on the items selected, the route is generated
Deleting groups	A user who owns (aka who created the group) must be able to delete a group
Leaving groups	A user has the ability to leave a group.
Displaying the route to users	The backend will send a list of waypoints to then be displayed for the user to then go and do the shopping.
Adding New Users to Groups	Users can enter a username of a user they would like to add to their group.
Selecting specific items from the list for a route	Users can select which items they would like to generate a route for.

Deliverable	Description
Web Application	The project will feature a web application that can deliver all the above functionality on an easy to use service
Android Application	The project will feature a mobile app that can deliver all the above functionality through a mobile platform.
Web Scraper	When an item is not in the database, the web scraper will search for the item and put the results into the database
Algorithms	When generating a route, we use Dijkstra's algorithm to generate the most efficient route from a dynamic starting point

2. SPECIFICATION AND ANALYSIS

2.1 TASK DECOMPOSITION

In this section, we break down each of the stages of the development process for the project. The Gantt chart below illustrates how long each stage should take, and when it will be started upon. The sections are explained more in detail below as well. See Figure 2 below for a diagram.

1. Identify Components
    a. In this stage, we would be talking with our client and start determining preliminary design. This would also include finding all the requirements and a general timeline when things should be done.
2. Research
    a. For this stage we would start researching components that would be a part of the solution. This would include researching getting item data from the stores, getting the road network information from an area, and the distance from stores to stores. Development environments that would work well with the ending product.

3. Develop Basic Design Plan
    a. Make a dynamic timeline
    b. Define a concrete solution.
    c. Cost and risk analysis.
    d. Determine end users and preliminary look of end product.
    e. Discuss and revise preliminary design with clients.
4. Make Use-Cases and In-Depth Design
    a. Make scenarios for each step of design.
    b. Build design for each component of our solution.
5. Finalize Design
    a. Set in stone the design of each component.
    b. Create a component diagram that shows how each component is connected to each other.
6. Finalize Basic Function
    a. Get a basic application working on either Android or Web.
    b. Be able to interact with the application with buttons or text boxes.
    c. Iterate through each scenario and build the required component functionality for each.
7. Review Design
    a. After the winter break review the design in order to refresh our minds.
    b. Add any details that we might have missed.
8. Begin Implementation
    a. Work with the basic functioning application.
    b. Make databases for the data that is going to be needed.
    c. Connect those pieces of data to be shown on the screen.
    d. Begin working out the algorithms for the route generation.
9. Complete Implementation with Testing
    a. Complete algorithms with simple inputs.
    b. Use the data being pulled in to interact with the algorithms.
    c. Design a better looking and functioning UI.
    d. Start testing the application for integration and unit testing.

10. Demos and Bug Fixes
    a. Get a working product that can be demoed.
    b. Fix any bugs that appear while testing and demoing the product.

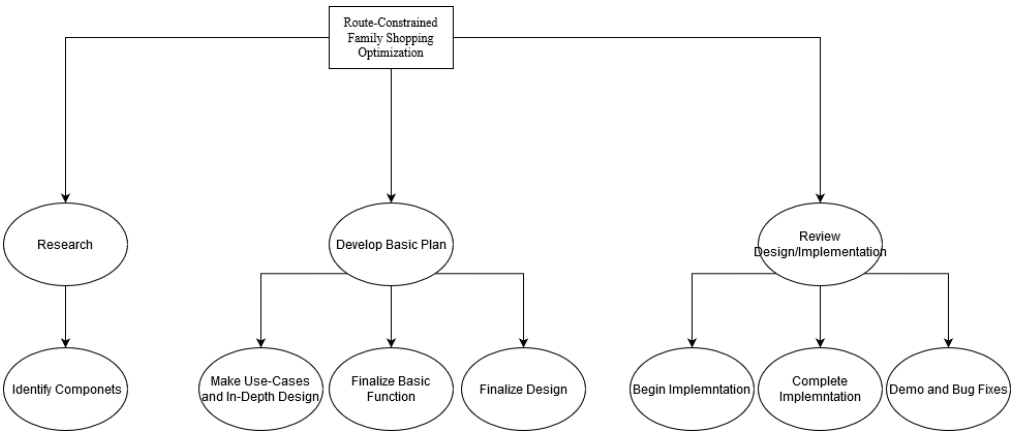


Figure 2: Task Decomposition

2.2 RISKS AND RISK MANAGEMENT/MITIGATION

Task/Component	Risk	Risk Probability	Risk Mitigation
Identify Components	Component is not identified or a component is incorrectly identified. Not all requirements are gathered	0.7	Meet with Client again to firmly establish requirements and components
Research	Information found is false or Information not available	0.6	Compare research results with multiple credible sources.
Develop Basic Design Plan	Preliminary design is not what the client asked for.	0.6	Meet with Client again to firmly establish requirements and components
Making Use Cases and in Depth Design	Scenarios do not correctly reflect how the end user will use the application	0.4	None
Finalize Design	Component Diagram incorrectly shows how each component is connected to one another	0.7	Meet with team and remake component diagram to make sure that each component is correctly connected
Finalize Basic Function	Scenarios were incorrectly formed on false requirements	0.2	None
Review Design	Design Document is missing information	0.8	Add any details missed in the design
Begin Implementation	Databases and Server cannot be accessed by the Web Application	0.5	None
Complete Implementation with Testing	Testing reveals bugs in code or implementation incomplete	0.6	Fix bugs and meet with whole team to fix issues regarding implementation
Demos and Bug Fixes	A bug is encountered that cannot be fixed	0.3	None

2.3 PROJECT PROPOSED MILESTONES,METRICS, AND EVALUATION CRITERIA

For the first semester our milestones are: To have a semi-completed design done halfway through the semester. The final milestone for this semester would be a completed design and design document. For the second semester our milestones will be the different levels of functionality for our project. Single user multiple stores, single group multiple stores, multiple starting times, travel_distance vs. travel_time, and Scalability. These milestones can be measured by their functionality and if they work. Another milestone will be the completion of the web application and mobile application which will be measured again by their completeness and functionality. One way to do this is to get test users and get feedback throughout the development process to get a continuous stream of feedback and evaluation criteria.

2.4 PROJECT TIMELINE/SCHEDULE

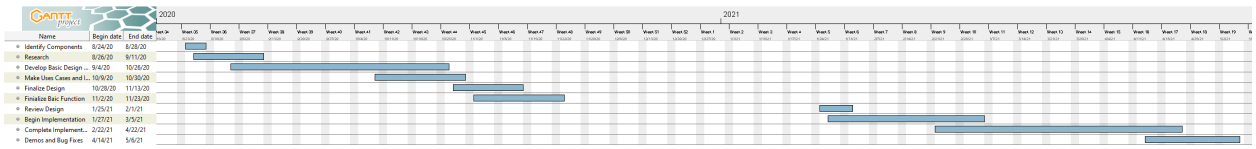


Figure 3: Gantt Chart

See Section 2.1 for a detailed description of each task associated in the Gantt chart.

2.5 PROJECT TRACKING PROCEDURES

Our team used Trello for project tracking and management and used GitHub for the file tracking and collaboration. We also used Discord to conduct team meetings and communication and also to keep track of meeting notes, assignment deadlines, and important information/announcements we want pinned.

2.6 PERSONNEL EFFORT REQUIREMENTS

Task	Effort Requirement (Person hours)
Identify Components	3
Research	12
Develop Basic Design Plan	30
Making Use Cases and in Depth Design	60
Finalize Design	36
Finalize Basic Function	30
Review Design	30
Begin Implementation	24
Complete Implementation with Testing	300
Demos and Bug Fixes	90
Total	615

2.7 OTHER RESOURCE REQUIREMENTS

There are no physical parts and materials needed to complete this project. The project was completed on various coding platforms. These platforms are Android Studio with Koltin, React.js, ASP.NET with C#, and Python for the algorithms.

2.8 FINANCIAL REQUIREMENTS

The Mapbox API we used costs $0 for every 50,000 requests per month. The team stayed within this 50,000 request limit, so the API had no cost to the team. Hosting a server for the web and mobile application cost $0 a month as we found a web host for free. Additionally, the web scraper and routing algorithm was put on an Amazon Web Server instance. This is free for the first 12 months if the use of the instance does not exceed 750 hours. Publishing the application to the Android store is a $25 one time payment. The route generation to find the stores incurs a cost of $0 unless we go over 500 uses in a month. In total, the cost for this project was $0.

3. STATEMENT OF WORK

3.1 PREVIOUS WORK AND LITERATURE

There is substantial literature and previous work available for research to aid in the development and design of the project. After surveying multiple options through the research of this literature, the group has decided upon utilizing Mapbox for a road network API. Mapbox has developed an API for generating road networks and maps which can help with shopping route generation. The advantage of using Mapbox is that it is free for the purpose of this project, and can generate an accurate map of an area. We initially decided upon utilizing a modified version of Dijkstra's shortest path algorithm called the A star (A*) algorithm for calculating the shortest path from a starting location to a desired store. However, for this project, we decided to use Dijkstra's algorithm because of its simplicity and popularity for shortest path generation. However, an additional factor was that we could not obtain road-maps with travel-time distributions across a broader geographical region(s).

Previous Work

- Mapbox
- Dijkstra's Algorithm
- A* Algorithm
- Distance Indexing

Literature

- "Distance Indexing On Road Networks" by Haibo Hu, Dik Lun Lee, and Victor C. S. Lee

3.2 DESIGN THINKING

In the figure below, the design process diagram developed by the team can be found. The main "define" aspect that has shaped the current design is to "utilize the client's suggestions to make a problem statement." The team has worked very closely with the client, meeting weekly to go over progress made and the expectations of what work needs to be done in days to follow. This has allowed the team to properly understand the expected requirements and make design decisions that best fit the project. For example, by meeting with our client consistently, the team was able to choose the proper technologies to develop the project.

The “define” stage was followed closely by the “ideate” stage, where the team has mainly focused on “brainstorming solutions to the problem statement.” In this stage, the team has established the simplest variants of the solution, and worked our way up to the most complex variant. This determines the timeline of later development, as well as the use cases for each variant.

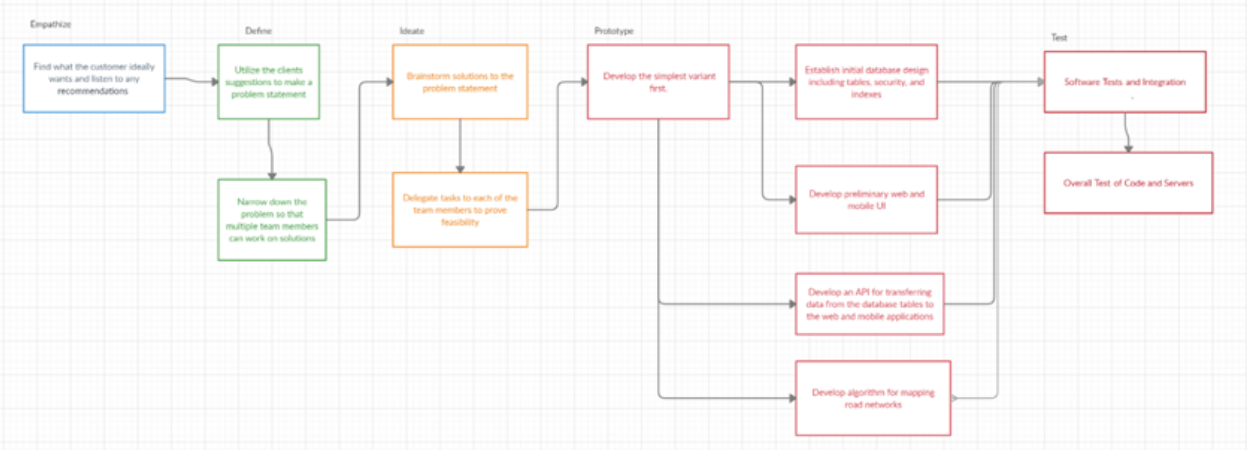


Figure 4: Design Process Diagram

3.3 FINAL DESIGN

To properly fit our design to the problem at hand, we investigated (and, subsequently, opted for) a divide-and-conquer approach, breaking down the problem into smaller ones - based on different facets of the problems and the corresponding tools/platforms to be used for implementing them. As we completed the smaller problems, we could combine them together to create a fully finished design.

Routing Algorithms:

For our project we must find the most efficient path between a user and the stores they wish to visit. To solve this problem, a pathing algorithm is needed. The two pathing algorithms that we came up with were the A* algorithm for time dependent shortest path and Dijkstra’s algorithm for shortest path. We chose to use Dijkstra's algorithm as we only need to find the fastest path to the stores. This will help us complete the following requirements: Outputting the closest store with desired items with respect to distance/time to travel | Output fastest travel time to any given store at desired start time | Routes must generate in real time

Storage of User and Store Data:

To properly store the information that we get from users and stores we populate tables in our database. To do this efficiently, we normalized the data before putting it into the database. All store information such as name and location is stored in one table. All product and price information is stored in another table. All user information such as name and address are stored in one table. All shopping list information is stored in another table. Normalizing the data allows us to update, delete, and add to the database faster. It also decreases the amount of repetitive data we have. This design will complete the requirements: SQL Data must be in real time | Store location accuracy

Creating a UI:

The UI was made both on mobile, with an Android application, and web application, with React. The Android Application was developed using Kotlin and Android Studio. This allowed us to use Google's most recent Android development tools. The UI provides a clean and easy to use interface for users. This was achieved by creating pages with minimal information and buttons that are clear and concise for users to avoid confusion. These same ideas will be used in the web application, giving the users the information they need and not cluttering the screen with unnecessary buttons and information.

Web Scraping Store Data:

For the project, we needed to be able to get an item and its price from the stores in a certain distance and store it in the database. We used web scraping to access the stores in an area and grab the item that is on the shopping list from the UI. Finally the item is inserted into the database via the web scraper. This was developed in Python.

Hosting a Server:

For the project, we hosted a server to contain all elements of the project solution. This server includes the API, UI, and the MYSQL Database. It is important that each piece of the project is hosted in a place where the other components can access and communicate with it. If this were not the case, then the project would not work. Hosting the server for our project is tied to all of the requirements discussed in earlier sections. We decided on using the web hosting site of Gearhost and AWS.

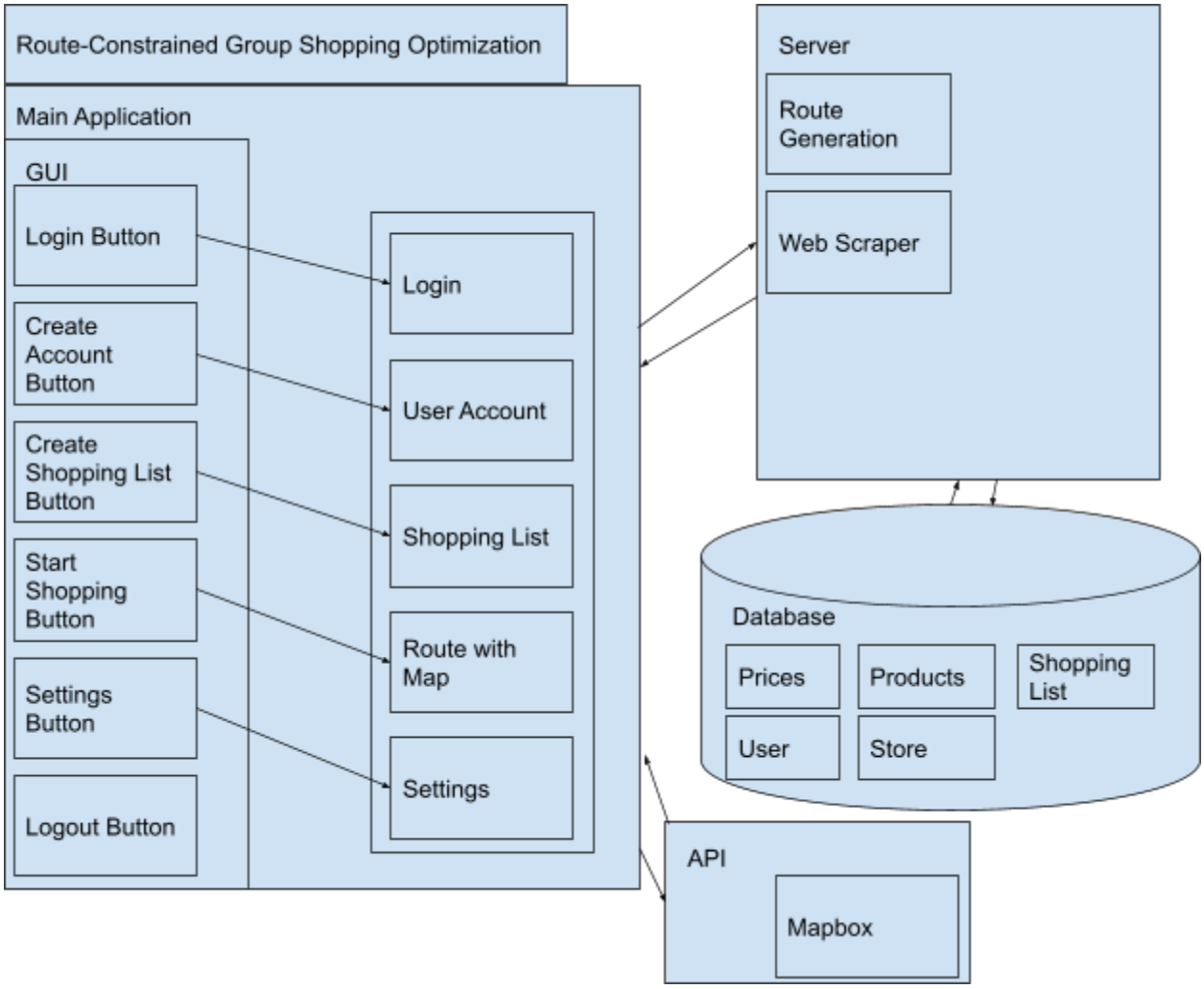


Figure 5: Block Diagram

3.4 TECHNOLOGY CONSIDERATIONS

The technology utilized within the project spans multiple languages and platforms. The development of a web and mobile application needed different software programs for their development. Additionally, the database needed to be maintained, as well as an API for the map used to find routes. The following technologies were determined to be the best fit for the development of the project.

- Android Studio
    - Mobile application

- React
    - Web application
- ASP.NET API
    - Grab data from the database
- MYSQL Database
- Mapbox API
    - Getting the map of the area needed
    - Stores in the location
- Windows IIS Server
    - Server for the backend
- Algorithms
    - Web scraper
    - Route generation

3.5 DESIGN ANALYSIS

We will break down the analysis into 4 sections: Android, Web, Backend and Algorithms.

Android:

The android application for this system was implemented using a mix of Kotlin and Java. The reason behind the mixture of these two languages is strictly due to developer experience and preferences with these languages. Additionally, the Mapbox API included great support for both Kotlin and Java.

The mobile app ended up having all UI components that were needed to complete the earlier stated goals. Some of these goals include: creating accounts, groups and shopping lists, modifying groups and shopping lists, selecting items to create a route with, and starting/displaying a route. Most of the issues present were purely due to time constraints with the semester ending and sacrificing the visual experience with the functionality to complete our end goals. This sacrifice resulted in the UI having some very obvious bugs, but the overall end goal was achieved.

Web:

The web application for this system was implemented using a mix of Javascript and React. By using Javascript and React, we were able to develop a fast, flexible, and polished application. Additionally, React has a component for the Mapbox API, which allowed us to easily integrate our routing implementation into the project.

In the web application, all the UI components needed to complete the previously stated goals were completed, including creating accounts, groups, and shopping lists, modifying groups and shopping lists, selecting items to create a route, and displaying a route. The web application does not look how we originally envisioned, but it still looks clean and professional, and all of the necessary functionality was completed.

Backend:

Our database is a MYSQL database and uses many tables to store the data necessary for our application to function correctly. To save on space, we normalized the data to an extent to prevent redundant data. To speed up getting data, we created views that retrieve data from a maximum of only three tables. For our API we use an ASP.NET application to act as a middle man for data transfer between the database and the frontend. We are using a package called Swashbuckle to give the API a sleek looking UI that allows the frontend team to test and access the API method calls. Each controller and it's methods are accounted for in this generated Web Application.

Algorithms:

The algorithms were initially designed to be as fast and efficient as possible. For the web scraper finding the items was parsing HTML within Python. This worked great as we got good results and filled out the database when a new item was needed. There was an issue that all web scrapers have where certain websites block these types of scripts. This did happen to us with Google Shopping blocking the web scraper. We found an alternative with Bing. For the Route generation it works and can take in items, filter the items for the stores and send a route to the front end to be displayed. Unfortunately, parsing the data for large amounts of stores does take some time. It is not as good as we hoped but it does work and has room for improvement.

3.6 DEVELOPMENT PROCESS

The main development process for this project is a modified version of Agile. We used Agile as it pushed the project forward through test-driven development. By having team meetings every two weeks, the design process was streamlined and allowed for a clear understanding of design goals by every team member.We had updates to the applications every week or two weeks. When integration of different parts of the program started, we met up in smaller groups depending on who was working on what. Every two weeks we had a sprint to discuss the previous two weeks of work and what to accomplish in the next two weeks. Additionally, we would meet with our advisor the day after sprint to discuss progress on the project.

Version Control and Project Tracking:

Our project used Github as our version control system so we could all collaborate on the project synchronously and avoid having conflicts with each other. For project tracking and task management, we used Trello to allow us to assign tasks to each other and track those tasks throughout the development process. Both of these tools were crucial for our team following the Agile development process.

3.7 OVERALL DESIGN PLAN

The main design plan was to use the five pillars of design thinking. By empathizing, we communicate with our client and find out the requirements for the problem. When defining, we are establishing the requirements and constraints for the project moving forward. In ideation, we determined multiple variants to the problem and ways to solve all of them. Then by assigning roles to each of these problems/solutions we are able to start prototyping. Once prototyping began, we started development of the application to figure out what ideas worked and what we can change to improve the final product. We then started unit testing and integration testing to start refining our project into a quality product.

3.8 Security Concerns and Countermeasures

The main security concern from the project stems from the fact that the application requires the user's location in order to generate a shopping route. To provide a safe application for our users, our project implemented password encryption to protect our users' login information. By encrypting our users' login information, we add another layer of security to protect the information necessary for our users to use our application. Another security countermeasure implemented in the project is that the web application created Private Routes to the application's urls. By using Private Routes, the web application forces the user to login to be able to navigate to the different web pages implemented in the application. If a user doesn't login, every page of the application will redirect to the login page, forcing the user to login. This protects the project from user's trying to interact with the application in unintended ways.

Another security countermeasure we practiced while developing this project is that we created multiple repositories to back up our code. By having multiple backups, should anything happen to our code and cause it to drastically malfunction, we can revert to one of our backup repositories and begin implementing again.

4. TESTING

4.1 UNIT TESTING

Some of the software components we have tested include: the algorithms, mobile app, web app, backend database storage, map routing, and the web scraping. Testing allows us to individually test each component before fitting all the pieces together for the final product. Testing also allows for focus on one area at a time, and thorough testing of that aspect of the product. Unit testing allows for quick identification and isolation of problems.

- Methods for achieving these unit tests include a manually created database to compare with the actual created database as data is sent so the two could be compared.

- For the web scraping it is to make sure that we get data back after parsing the data.

```
class TestWebScraper(unittest.TestCase):
    def test_NonNull(self):
        self.assertIsNotNone( WebScraping.productJson, "Should retuen the obj")

    def test_Around10(self):
        self.assertGreaterEqual( len(WebScraping.productJson),10, "Should have 10 results")

if __name__ == '__main__':
    unittest.main()
```

- For mobile we can have a third party create various unit tests, for the algorithms we give it sample data and have the algorithm run through the code to see what the result is. Then compare these results to the expected results.

The next versions of the project we would want to test more specific API to get the items. So this can be testing things like the Walmart API to see if we are getting the price and the name of the item. Also, to confirm that the item is in the store. We would also want to go more in depth with the testing as we had some done but need more specific tests.

4.2 INTERFACE TESTING

Interface testing was done by testing different scenarios within the Web and Android Applications. These scenarios included testing different use cases such as adding a user to a group and starting a shopping trip with a set list of items. Once all of the use cases were tested, we did extensive QA testing to make sure that all features of the applications worked as intended.

- Algorithms and database
    - To test the web scraping algorithm with the database, we used the web scraper locally to add items to the database. To ensure the functionality of the web scraper we validated that the new products and stores appeared in the database. The pathing algorithm did not need any implementation testing with the database.
- Algorithms and user interface
    - To test the pathing algorithm with the user interface, we sent product data to the pathing algorithm and ensured that the mapbox data that was sent back correctly pathed to each item that was given in the product data. To test the web scraping algorithm with the user interface, we called the web scraper with the name of a new item we wanted to add. To ensure functionality, we validated the appearance of the new products and stores within the database.

- Database and user interface
    - There were many instances that needed to be tested in terms of the union between the user interface and database. To test all of these transfers of data we used http tests to make sure that each call to the API from the interface received an HTTP OK status as well as the correct application format from the response.

- Algorithms, database, and user interface
    - Testing everything together to see if the project is working smoothly. This is done by showing our advisor and letting other users besides us to test it. Also, to make a demo video to be at the level of presentation for the final presentation.

By testing the units in groups of two before testing all of the units together it will allow quick identification of problem areas in the units.

4.3 ACCEPTANCE TESTING

The use cases were the key to make sure that our functional and nonfunctional requirements were met. We ran each test for each use case many times to ensure that each test passed without any failures. We met with our advisor as well to discuss our final product. During this meeting we demonstrated the functional and nonfunctional requirements. He was satisfied with the results of the demo. We also reviewed the project description before and after working on our use case tests. We did this to ensure that each functional requirement was met.

4.4 RESULTS

With the testing we started out by doing unit testing. We tried to do as much as we could locally before tying in more systems. The unit testing included having the web scraper getting results back. This can be shown above in the unit testing section. We also tested adding data to the database and making sure the data was correct and secure. With the testing of routes we do something similar and see if there is data coming back from the route generation. For the interface testing we mainly focused on doing two of the components and making sure they connect together. This worked well as we found problems and it was easy to find them. Instead of having to worry about everything in the project we were able to narrow stuff down and when bringing it all together it went relatively smoothly. For acceptance testing we combined everything and had other people test it. We then went through the entire process in order to make a demo for our advisor and for the final presentation.

5. IMPLEMENTATION

5.1 OVERVIEW

Our first decision in the implementation was to split off into 3 different implementation groups. The 3 groups were Android, Web, and Algorithm/Backend development.

Android implementation was to be done in Kotlin and Java. The main framework for the mobile app was created using Kotlin, while the map related code was mostly done in Java. Kotlin was the choice for the framework since it is generally more lightweight and faster to compile than Java. The reason that the map related code was done in Java was due to developer competence in Java rather than Kotlin. The initial mobile app was created with hardcoded values for everything to get the initial UI created. This then allowed us to add in api calls that would populate the various data fields and text fields that were displayed. Once we had that finished we were able to connect the UI and Map/algorithm pages completing all the intended goals.
In the development of the Web application, the first step was to set up some simple UI designs for various pages. We used React and NodeJS to run our application. We started by creating a login page and home page that could route to one another on a login submission. To start, all of our data was placeholder data that was hard coded into the application to create the proper UI design. After that we expanded to specific list pages, a settings page, and a page to hold the map. Once the initial UI design was complete, we set out to connect all of our information to the database, so we could stop using placeholder data and instead start using some real life calls from the database. This proved to be challenging at times, because what had previously worked for placeholder data didn't always work the same with real data. Some of the previous UI had to be re-written and re-tested for the implementation to work correctly. Finally, the last step in the implementation was to connect the web scraper and Mapbox routing algorithm to the application in a clean, user-friendly way. This step was fairly smooth since connecting these elements did not require much UI overhaul and were quick to connect.

With the Algorithms a lot of research went into which language would work best. C# was one of the language candidates for the web scraper but when seeing what would be efficient and easy to work with Python was selected. This was done in one file and would encompass finding results for the item and then adding it to the database. It went from local data to being able to get data from the internet. Along with this file there was the route generation. This was also done in Python to make it easier to call when it was eventually put onto a server. The server was made with node.js. The node.js app is a simple web server that the front end can call for both web scraper and route generation. The web server would spawn a new process for each call.

Our database design has been modified since the beginning of our design process. We ended up having to add more tables to our database to account for the normalization of some of our tables. We also needed to create more views for use of data access for the frontend teams. Modifying our database led to a ripple effect of modifications that needed to be made to the rest of our backend. Our API needed more database models to be able to account for the new tables and extra views that were created. New methods that modified and retrieved this data also had to be created. We also switched from the idea of using our own server to host pieces of our project. We ended up using a free hosting service.

5.2 EVOLUTION OF DESIGN

Our project had undergone substantial design changes throughout the implementation stages. One of the earliest changes was the switch from a desktop application to a web application. Another change was switching from the A* algorithm to Dijkstras. This was a big change because it reduced the amount of complexity but still had the same result we were expecting. With the web scraper we had to come up with contingency plans if the web scraper was blocked from getting results by Google's anti-script software. The solution here was to switch from Google Shopping to Bing Shopping. The route generation was originally made in Java, but with using node.js it was easier to make new processes with Python. So the code for the route generation was made in Python. Another evolution was to have the route generation algorithm to be on a server instead of the front end code bases.

6. CLOSING MATERIAL

6.1 CONCLUSION

We have completed our design and implemented all steps of functionality for our project. We have the web application, API, database, android application, and web scraper all working together. To complete those aspects, we iterated through each of our simplified scenarios until we reached the most complex and complete scenario. The application is now being hosted on cyshopper.gear.host and all use cases are implemented. Once all use cases were completed, we went through each of our Unit and Use tests to ensure the functionality of the application.

6.2 APPENDIX I (OPERATION MANUAL)

Download app on Android:

Go to the Google Play store store and look for CyShopper

Go to the website:

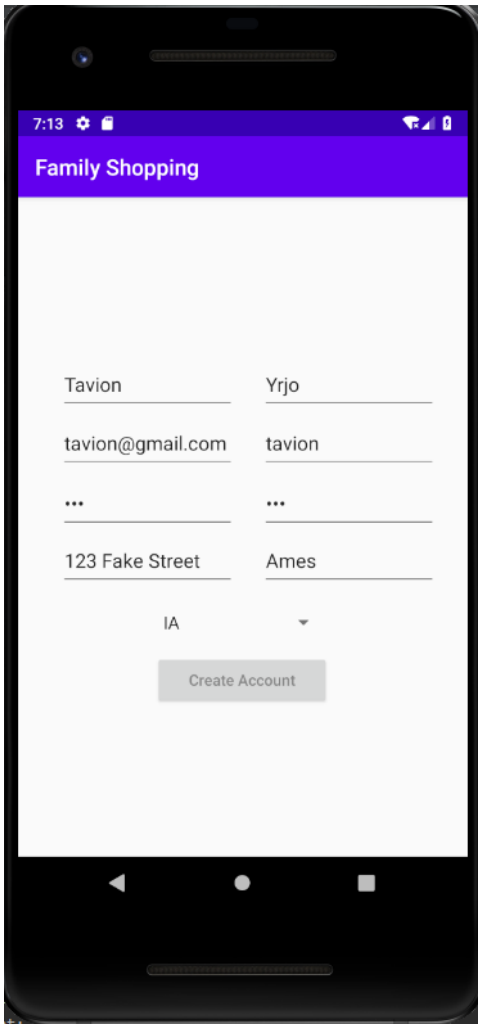
Navigate to our website at cyshopper.gear.host

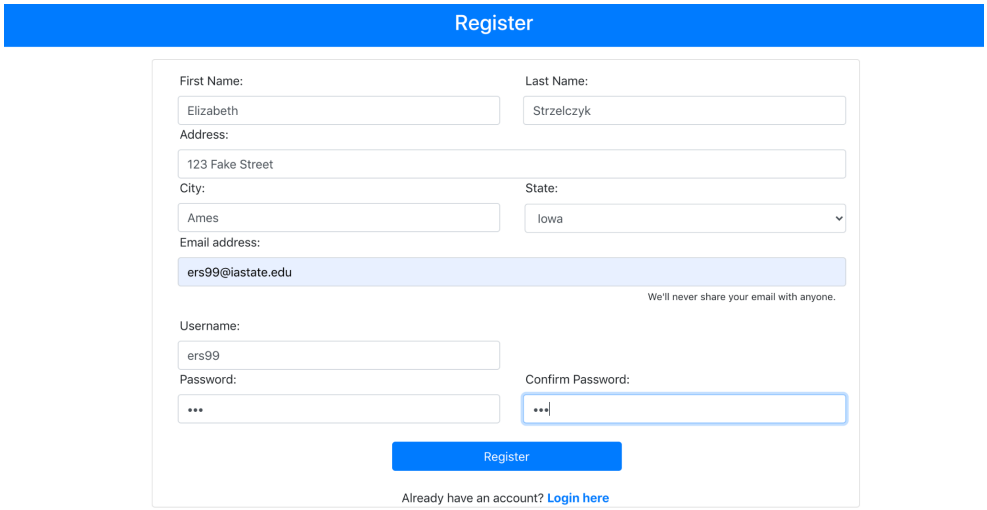
Register as a user:

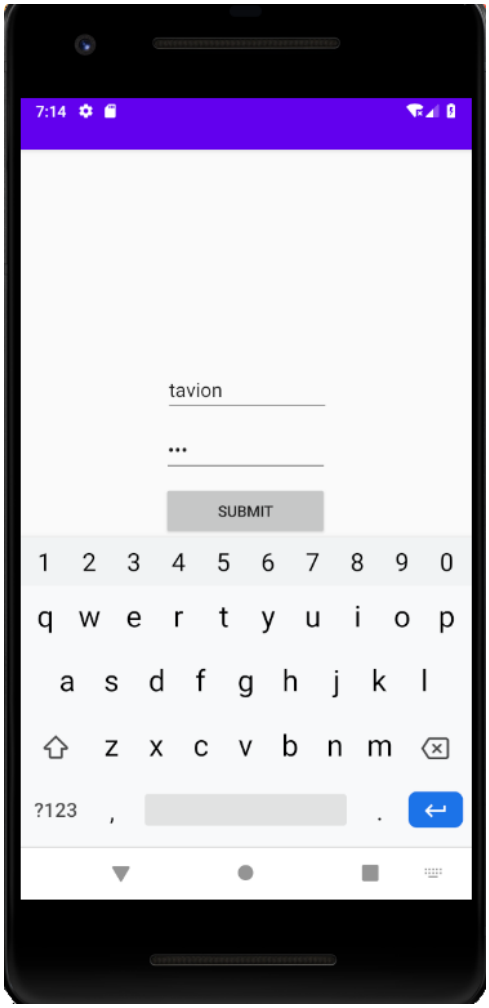
Enter your user information that is: name, location, email, username and password. Each has a text box and hit submit when all items are filled.

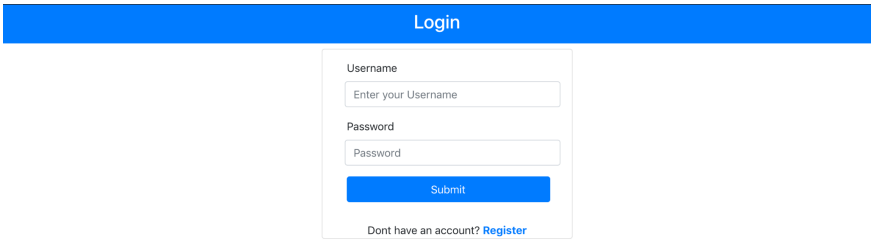
Login as the user:

You will be redirected to the login page. Login with your new credentials. You will be redirected to the homepage.



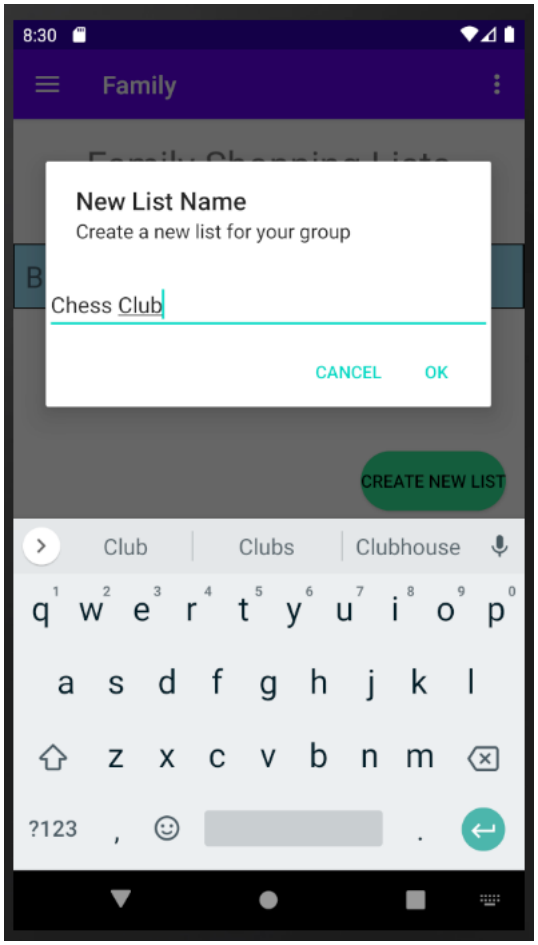


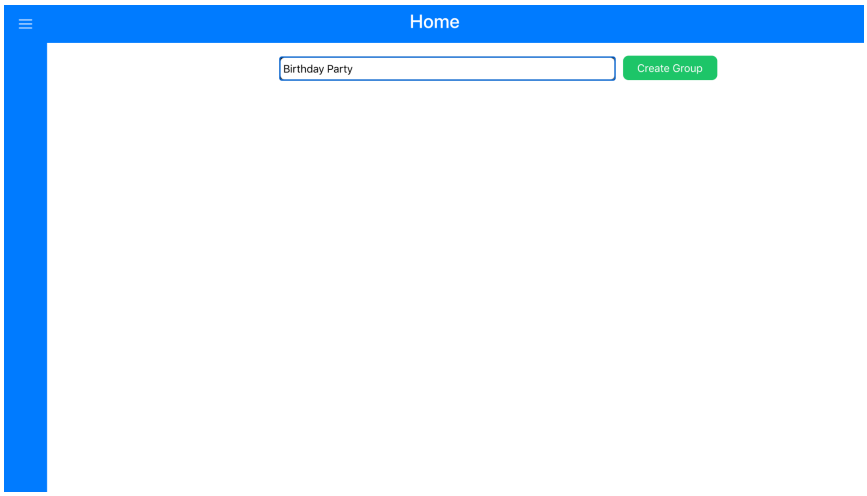

7:14
tavion
SUBMIT
1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123


Login
Username
Enter your Username
Password
Password
Submit
Dont have an account? Register

Create a List:

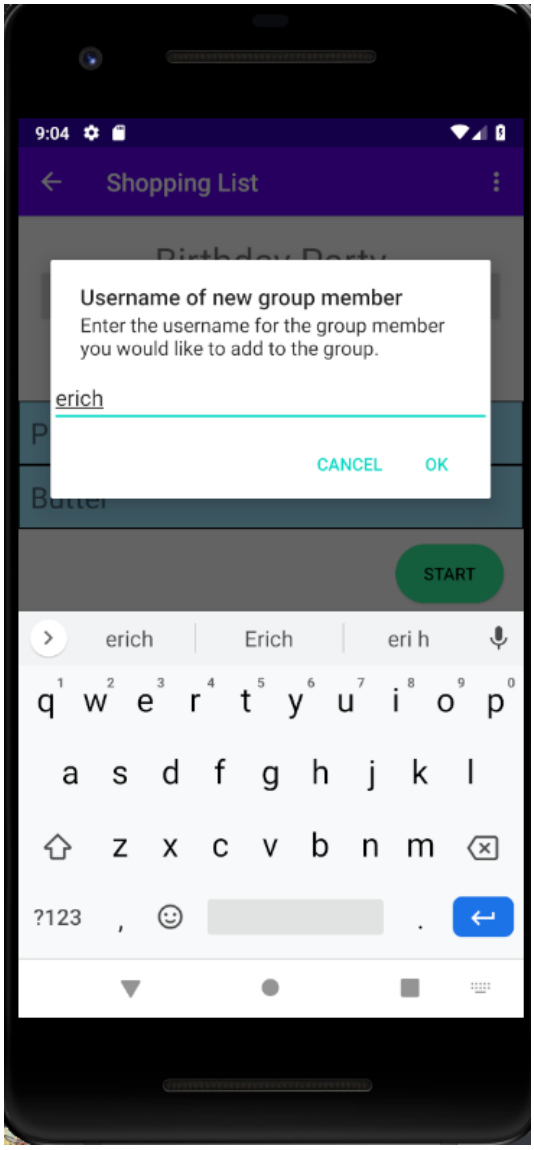
Once on the homepage create a new list. Put a name in the text box and hit the add button. Once the list has been created you can click on it or create other lists. If you do click on the list then you will be taken to that list's items.

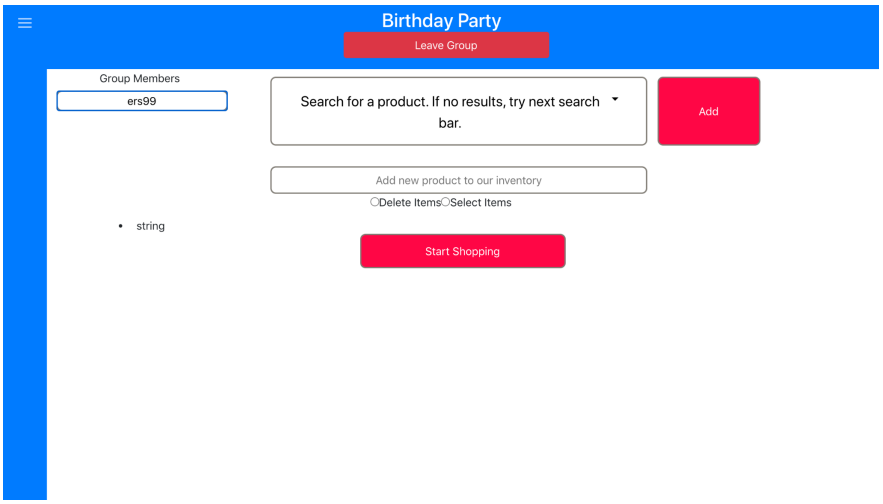




Add other users to the group:

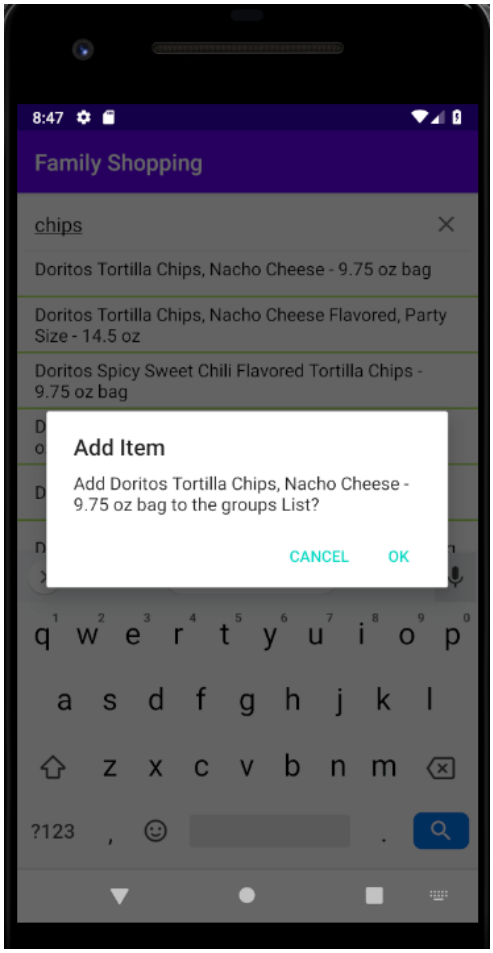
Add other users to the group you created by using their username.

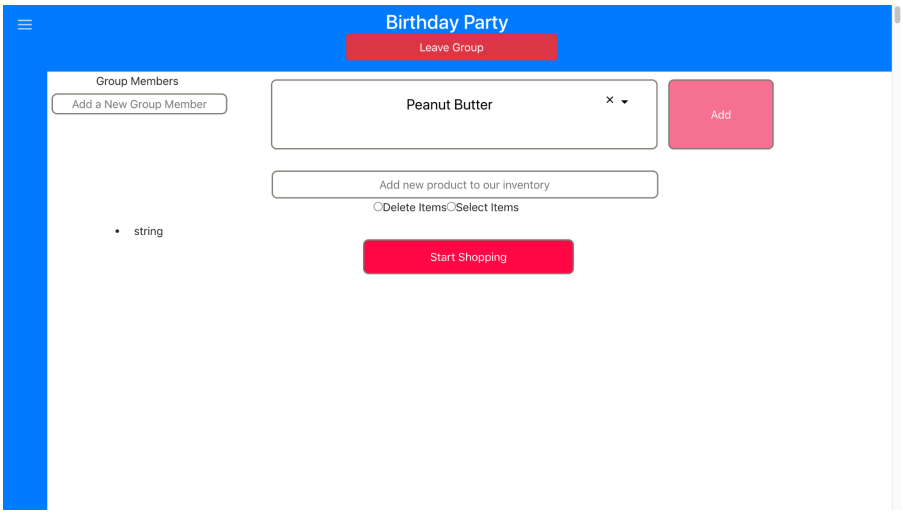




Add items to the list that exists:

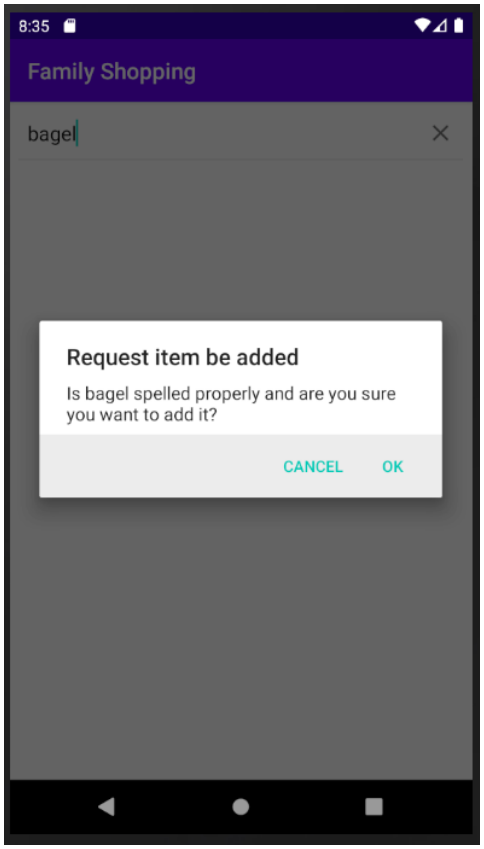
Once in the list page, refer to the drop down box above to add items to the list that are already in our database. These items can be searched for by typing into the box. Once the item you want is found, click on it and it will be added to the list.

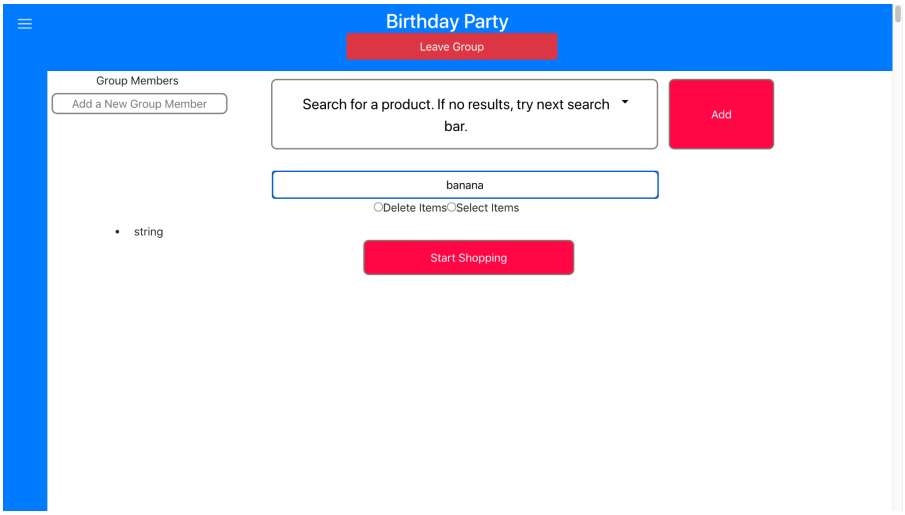




Add items that do not exist:

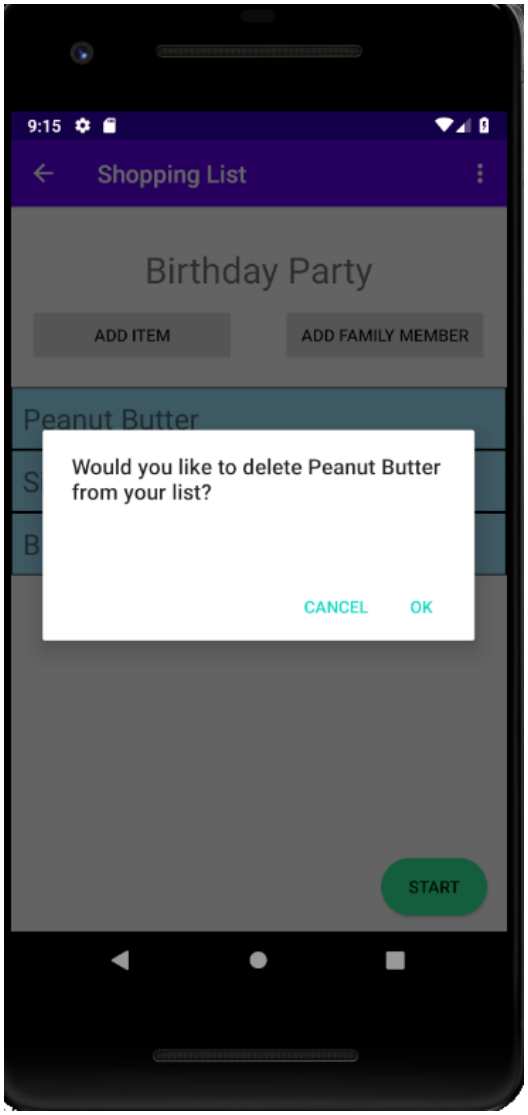
If you do not see the item in the search/drop down box enter the item's name in the text box that says "Add a product to our inventory". Once you do hit the enter button on your keyboard and it will add items with that name to the search/drop down. Then follow the step above to add the item to your shopping list.

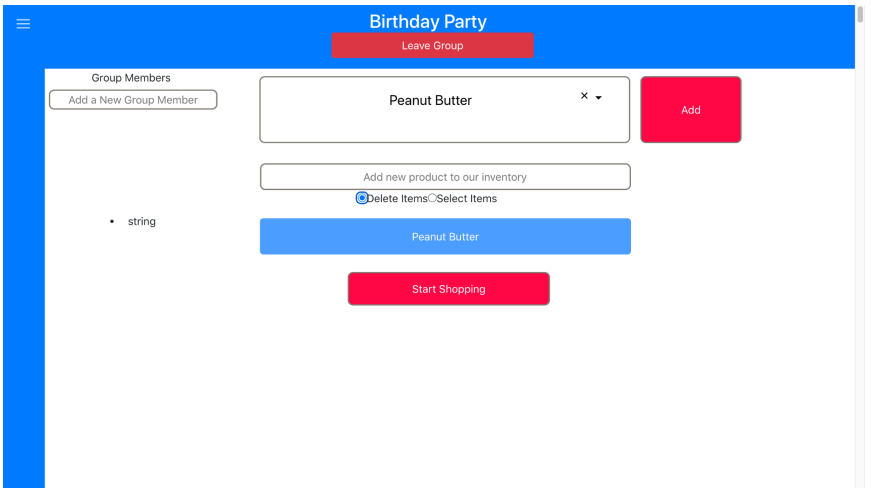




Delete item from list:

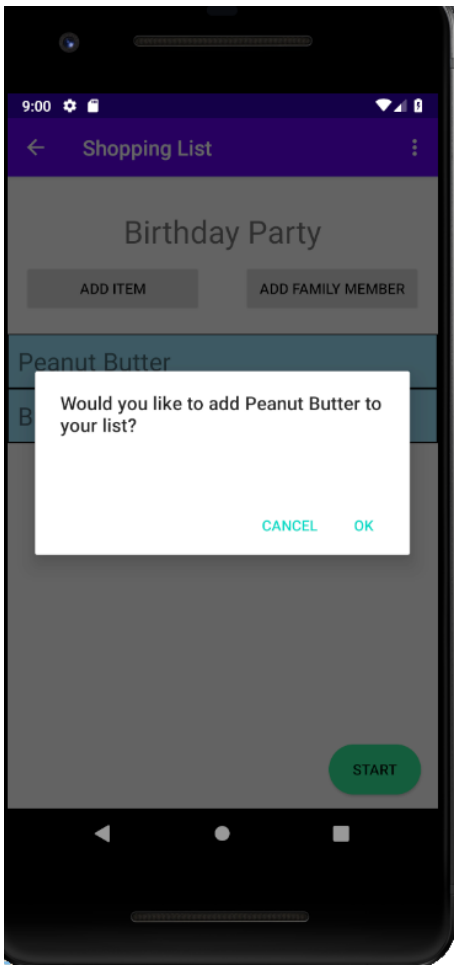
Hit the radio button that says “Delete Items”. Once this radio button is selected, you can click on any of the items listed within the shopping list and they will be deleted from the list.

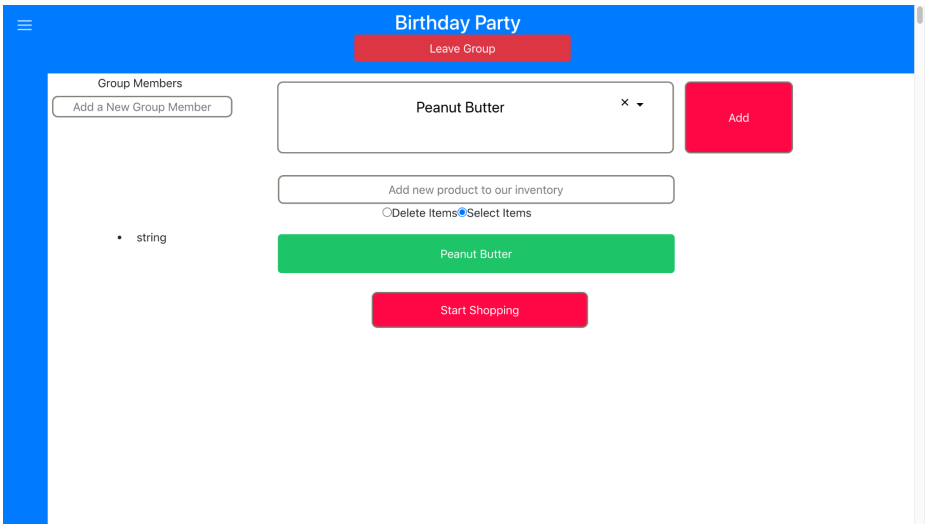




Select items for route:

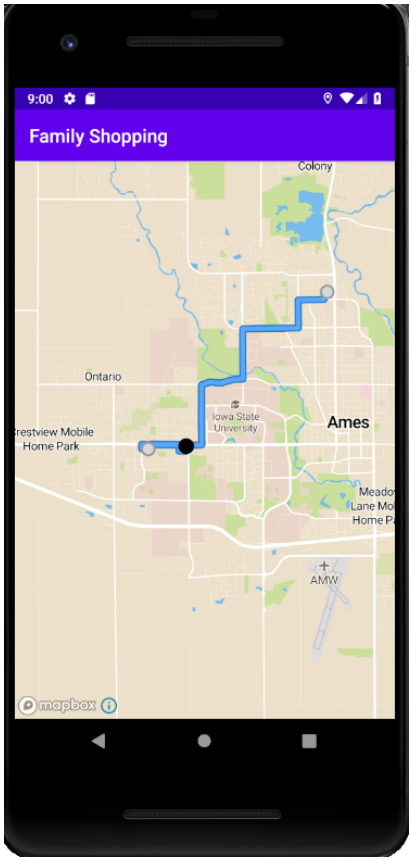
Hit the radio button that says “Select Items”. Once this radio button is selected, you can select all the items you want to shop for. Then you can hit the start shopping button.

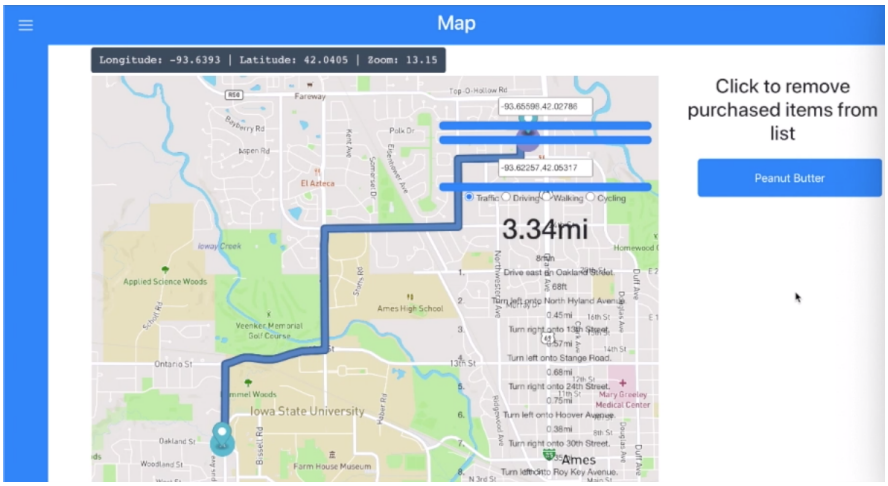




Generate route with the items selected:

Once you hit the “Start Shopping” button it will take you to the map page. It will take the items and generate a route based off of those items and their respective stores. After a brief loading period, the map on the page will generate an optimized route for the user to retrieve all of the selected items on their list.





Remove items when on route

Not available on mobile but does work on Web application

Users will remove items from their list as they pick them up. They will be able to select the item and it will disappear from both the map list page and the general shopping items page.

Once you have traveled on the route, you can start the process over again or logout and until other time shopping.

6.3 APPENDIX II (PREVIOUS DESIGNS)

Pictures from our first semester design can be found in section 6.5 and shows a rudimentary design of what our mobile and web app were going to look like. This idea was modified at the start of the semester as we started coding our apps and found what worked/looked best with the tools available to us. At initial second semester meetings we also had some design changes occur due to our group's idea of what we wanted our end goal to look like and what our project and team advisor had in mind. We were going to focus more on a broad range of groups and group dynamics and our advisor pointed us in the direction of focusing on a simple approach and capping it at a single family to avoid unnecessary complexity in the project.

6.4 APPENDIX III (OTHER CONSIDERATIONS)

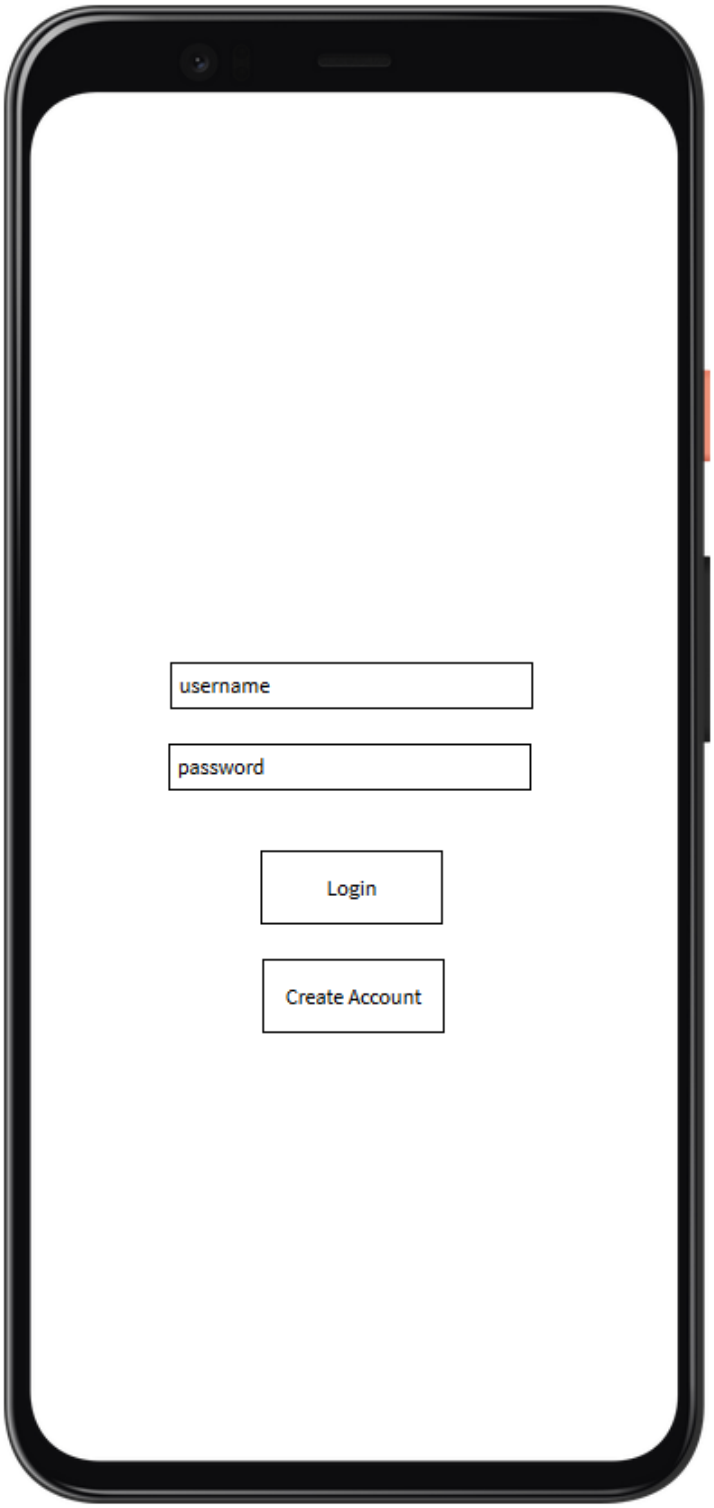
6.4.1 WHAT WE LEARNED

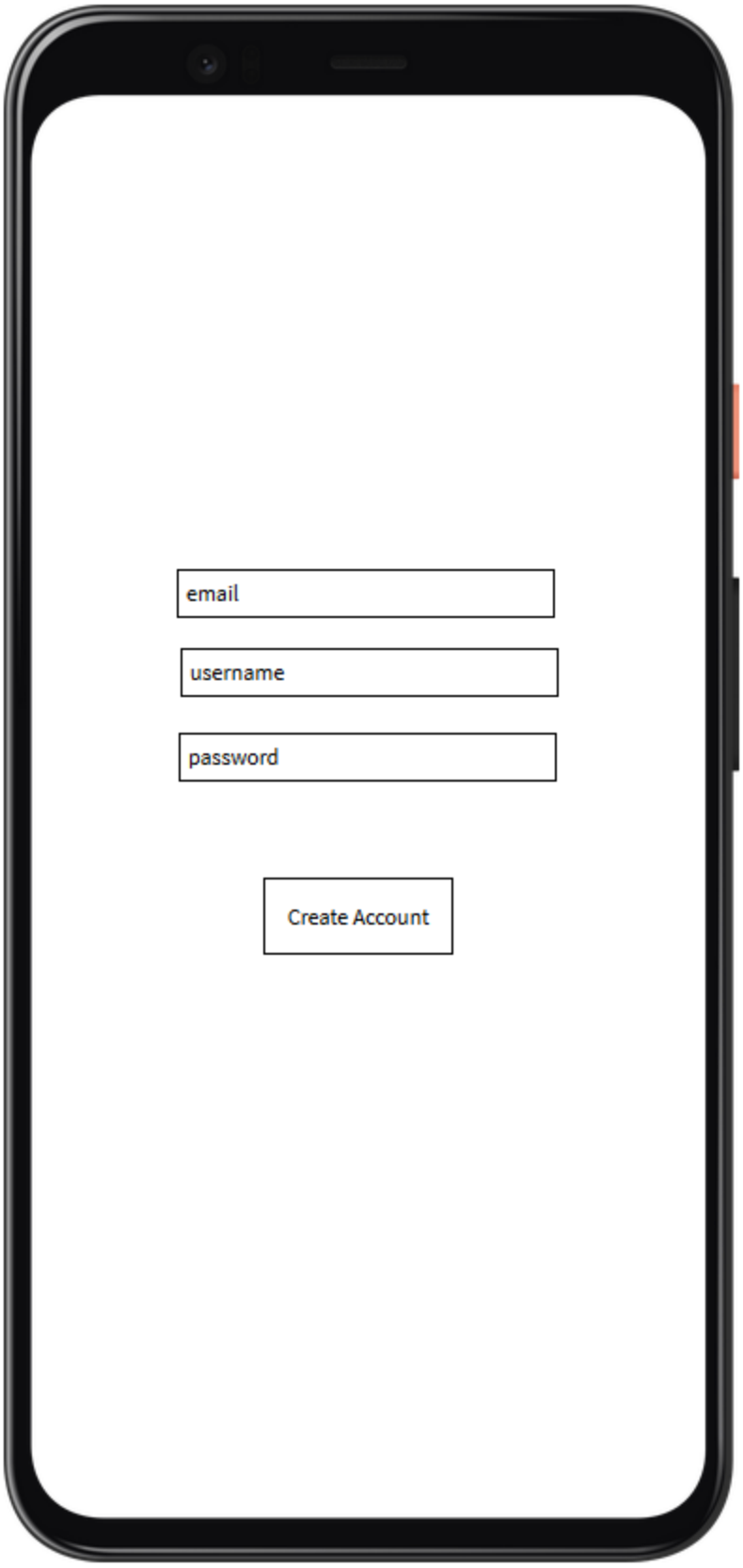
Through the year long senior design class our group had to tackle many new problems and work with new technologies that some of us had zero experience with before the project. On our android side Tavion and Colin W. learned/got more practice in Kotlin and android development. Our web team of Elizabeth and Erich had no experience with react and outside of the one web development class at ISU had no web development experience so there was a steep learning curve. None of us had worked with mapbox prior to this project so working with a map api was a new tool that we were able to add on to our toolbelt. The other big learning experience was the combination of the web scraper and algorithm that we had to use to circumnavigate the paid api services that were available on company websites. Using python we were able to implement them both that took a lot of trial and error to get them to work. Throughout the semester we also learned how to work in a team and work with team members of different skill sets. We had opportunities to practice communication skills between team members and project managers/team leads. These skills will translate nicely into the workforce as we all transition into other jobs and areas of our lives.

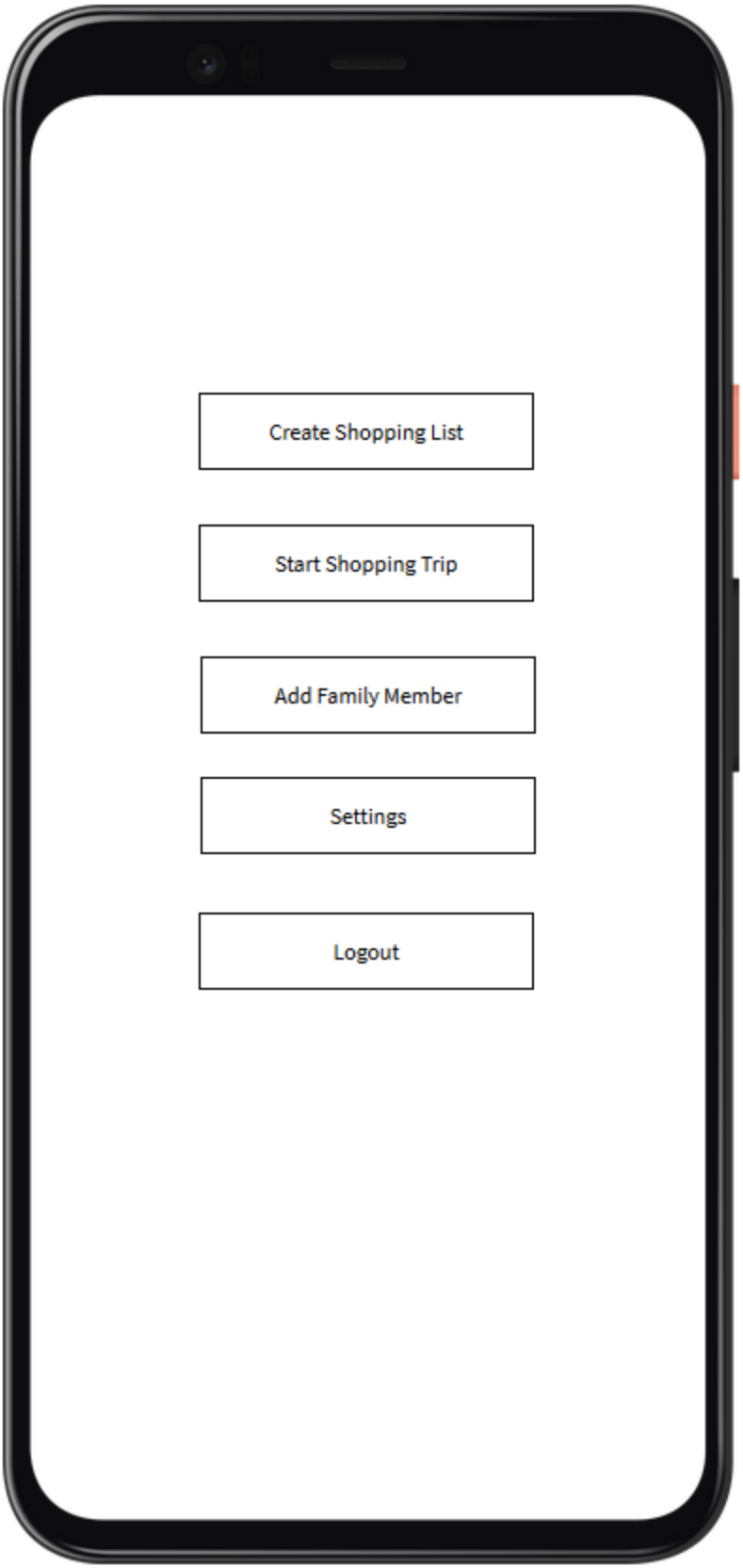
6.4.2 FUNNY MOMENTS

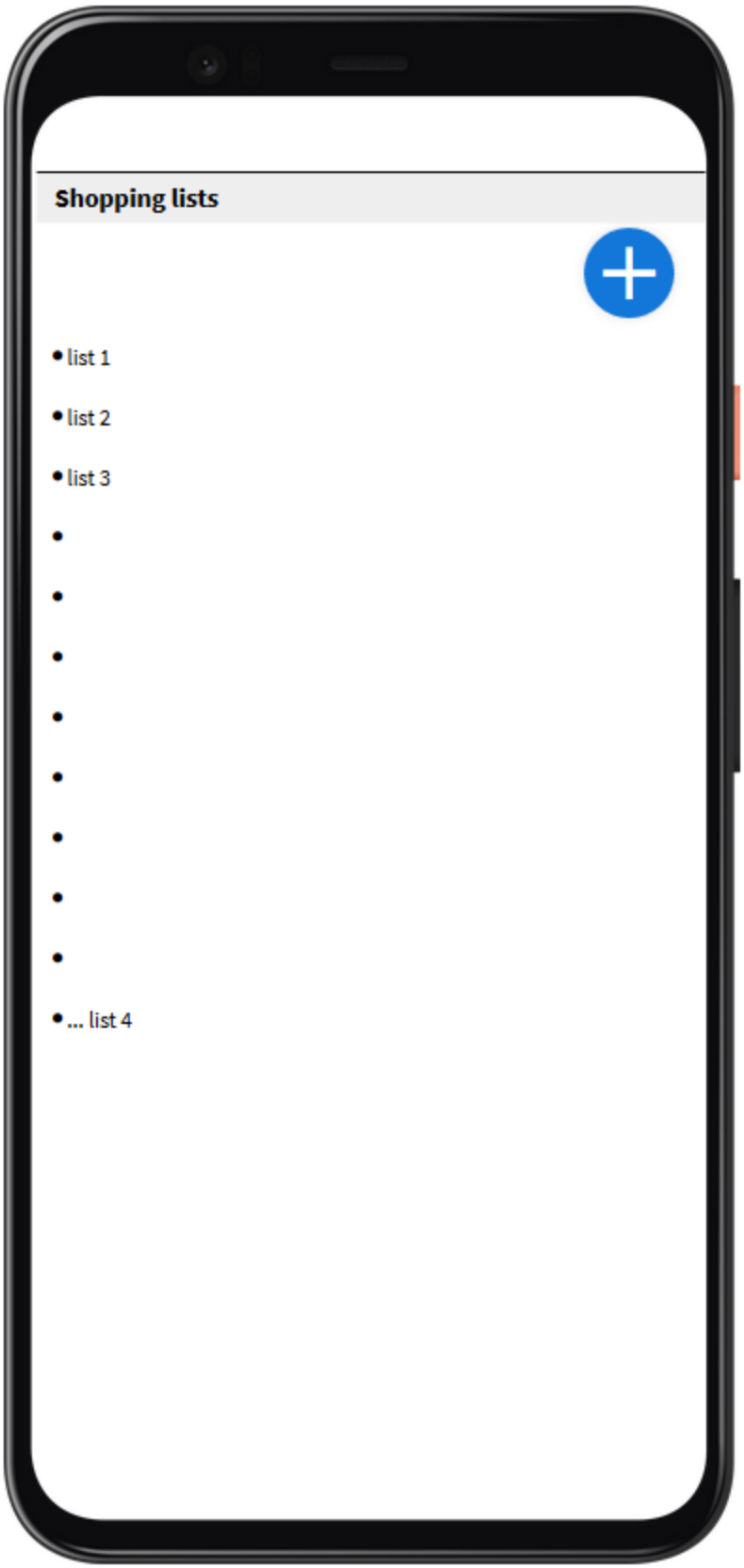
Lots of funny moments occurred during testing due to the nature of software bugs. One such moment was when we were testing our routing methods and our optimized route decided to take us across Lake Michigan to go on an international road trip to Toronto, Canada and then Georgia to pick up a cucumber in Toronto and then a full pear tree in Georgia. Another funny story is when we had to switch over to Bing for our webscraper because Google had flagged our webscraper as a bot and wasn't allowing the webscraper to continue using their search engine.

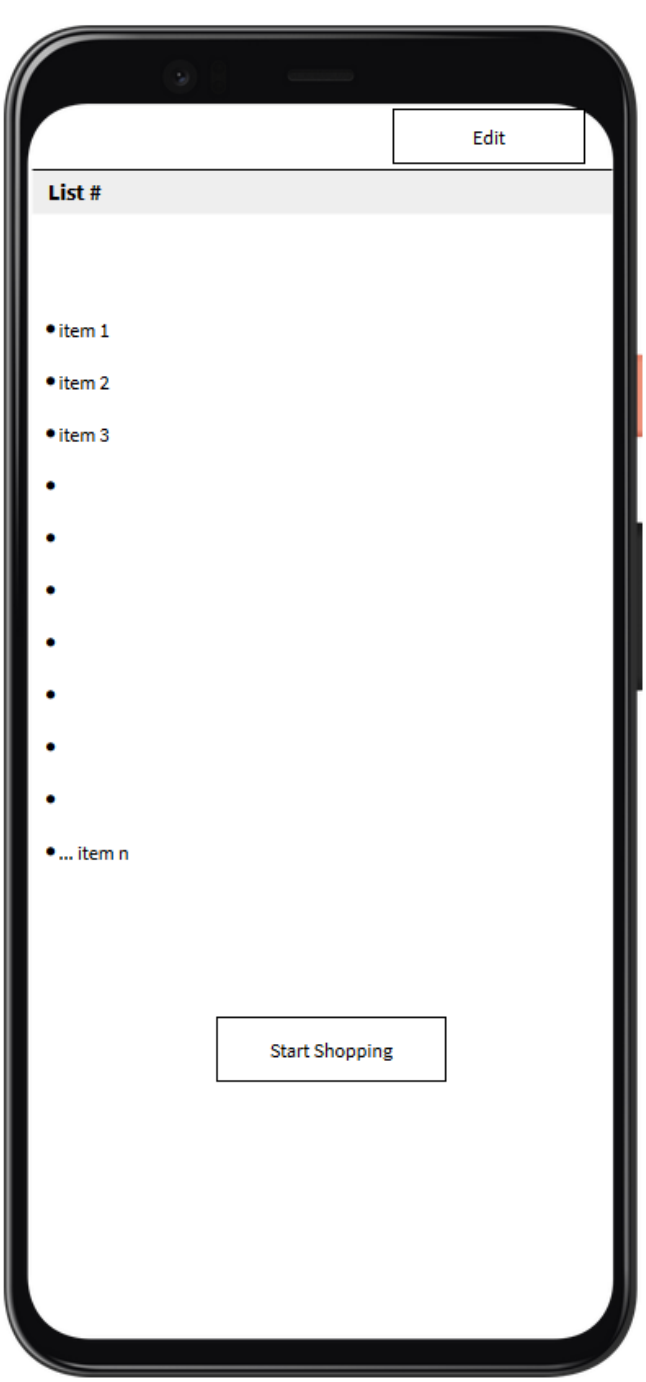
6.5 APPENDIX IV (OLD PHOTOS)

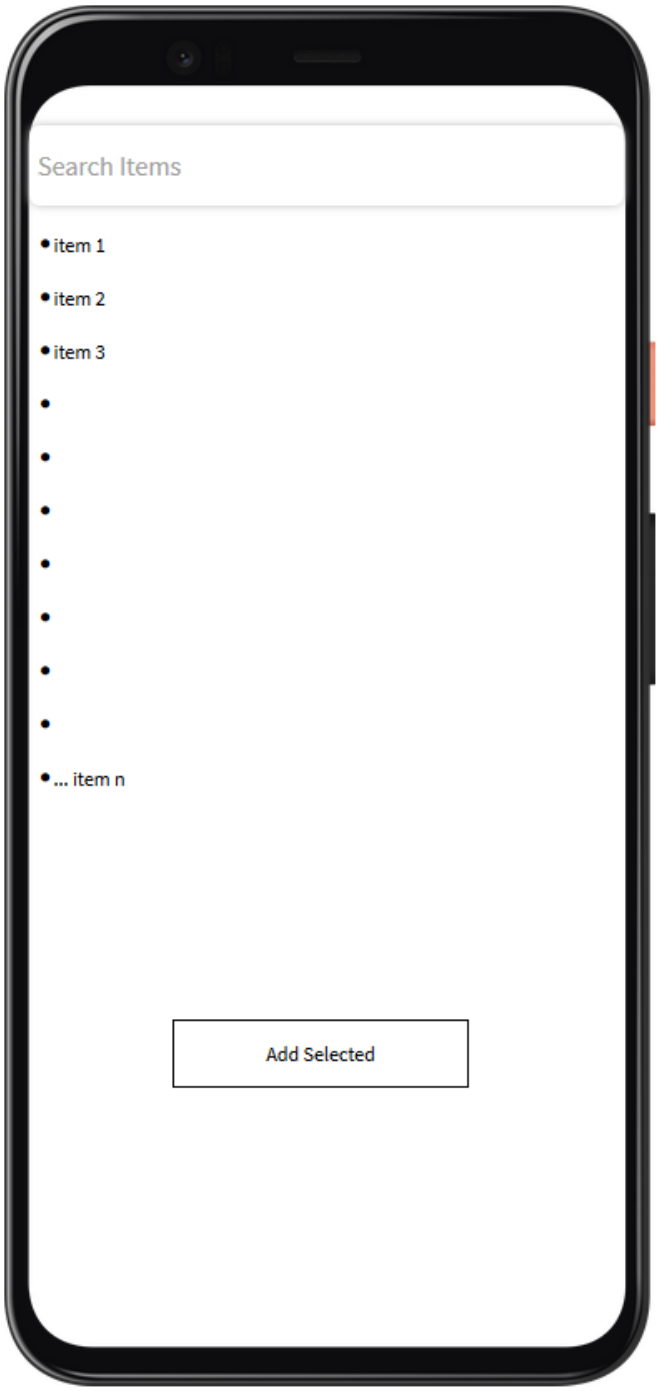


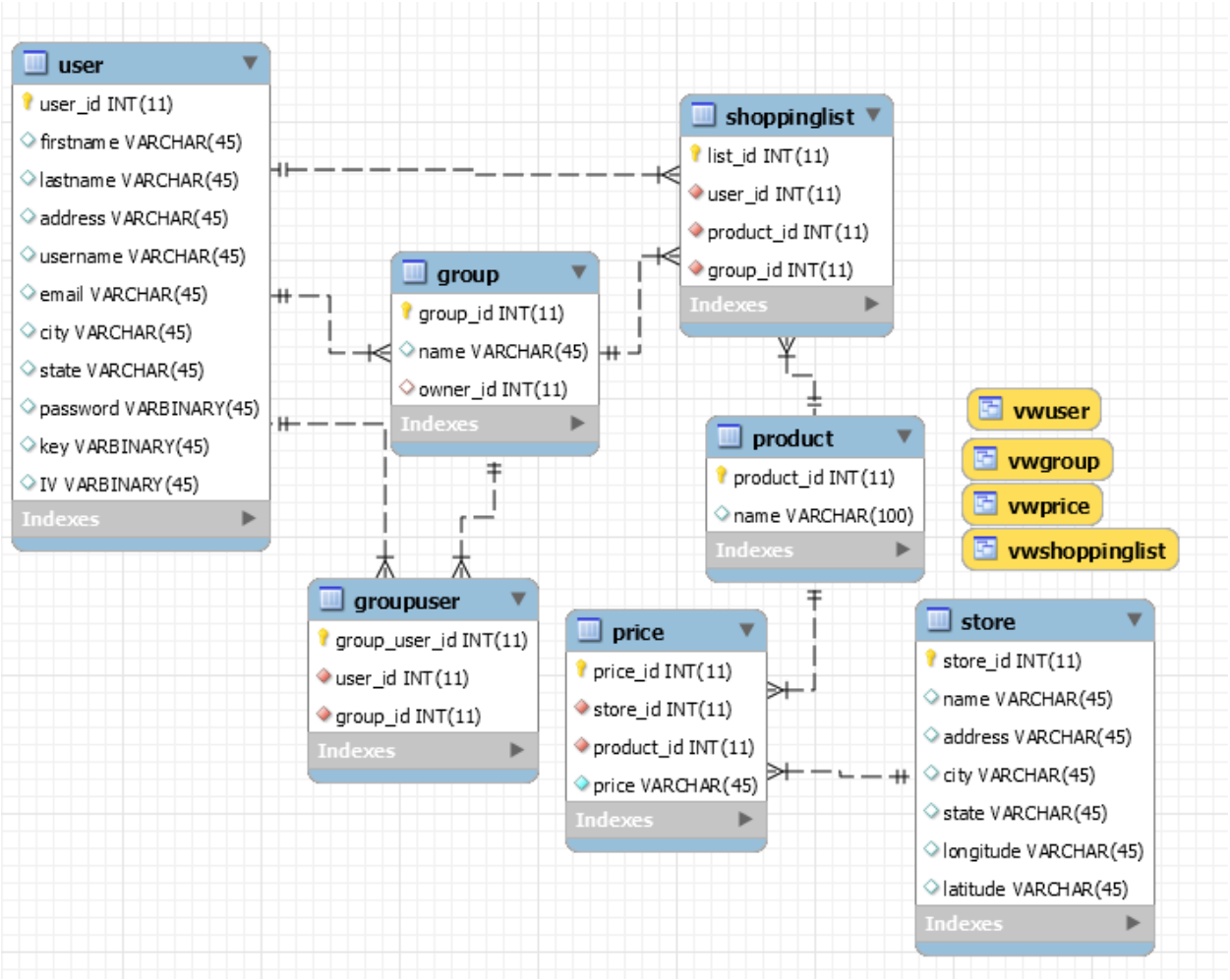

email
username
password
Create Account

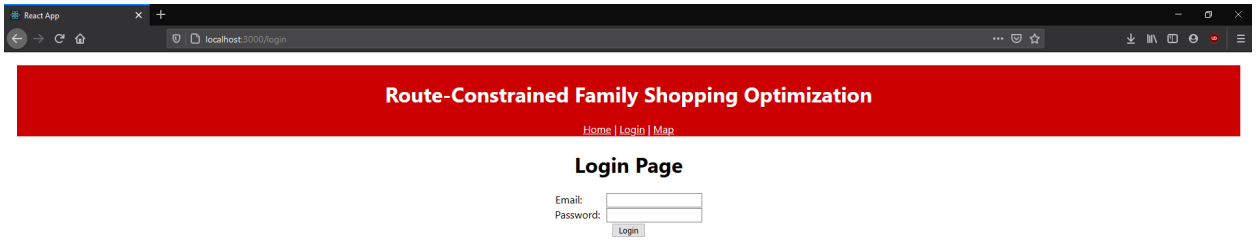

Create Shopping List
Start Shopping Trip
Add Family Member
Settings
Logout


Shopping lists
list 1
list 2
list 3
... list 4


Edit
List #
item 1
item 2
item 3
... item n
Start Shopping


Search Items
item 1
item 2
item 3
... item n
Add Selected


user
user_id INT(11)
firstname VARCHAR(45)
lastname VARCHAR(45)
address VARCHAR(45)
username VARCHAR(45)
email VARCHAR(45)
city VARCHAR(45)
state VARCHAR(45)
password VARBINARY(45)
key VARBINARY(45)
IV VARBINARY(45)
Indexes
shoppinglist
list_id INT(11)
user_id INT(11)
product_id INT(11)
group_id INT(11)
Indexes
group
group_id INT(11)
name VARCHAR(45)
owner_id INT(11)
Indexes
product
product_id INT(11)
name VARCHAR(100)
Indexes
vwuser
vwgroup
vwprice
vwshoppinglist
groupuser
group_user_id INT(11)
user_id INT(11)
group_id INT(11)
Indexes
price
price_id INT(11)
store_id INT(11)
product_id INT(11)
price VARCHAR(45)
Indexes
store
store_id INT(11)
name VARCHAR(45)
address VARCHAR(45)
city VARCHAR(45)
state VARCHAR(45)
longitude VARCHAR(45)
latitude VARCHAR(45)
Indexes


React App
localhost:3000/login
Route-Constrained Family Shopping Optimization
Home | Login | Map
Login Page
Email:
Password:
Login


React App
localhost:3000/map
Greenland
Iceland
Sweden
Finland
Norway
Russia
Canada
Denmark
Ireland
Belarus
Ukraine
France
Kazakhstan
Mongolia
United States
North Atlantic Ocean
Italy
Kyrgyzstan
Portugal
Greece
Turkey
Turkmenistan
China
Morocco
Tunisia
Syria
Afghanistan
Algeria
Libya
Egypt
Pakistan
Nepal
Mexico
Cuba
Saudi Arabia
Oman
India
Bangladesh
Mauritania
Mali
Niger
Chad
Sudan
Yemen
Senegal
Laos
Cambodia
Philippines
Sri Lanka
Maldives
Malaysia
Indonesia
Cameroon
South Sudan
Somalia
Colombia
Ecuador
Peru
Brazil
Angola
Zambia
Tanzania
Bolivia
Zimbabwe
Namibia
Paraguay
Chile
South Africa
Australia
Uruguay
Argentina
South Pacific Ocean
South Atlantic Ocean
Indian Ocean
mapbox
© Mapbox © OpenStreetMap Improve this map

6.4 REFERENCES

A. Chiang, “Distance Indexing on Road Networks”, presented to CS 4440 [PowerPoint Slides]. Available: https://iastate.app.box.com/file/714037198015

“A* Search Algorithm,” *GeeksforGeeks*, 07-Sep-2018. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/. [Accessed: 02-Nov-2020].

Mapbox. [Online]. Available: https://www.mapbox.com/. [Accessed: 02-Nov-2020].

T. L. M. RK. Ahuja, et. al., “Spatial crowdsourcing: a survey,” 01-Jan-1993. [Online]. Available: https://link.springer.com/article/10.1007/s00778-019-00568-7. [Accessed: 03-Nov-2020].