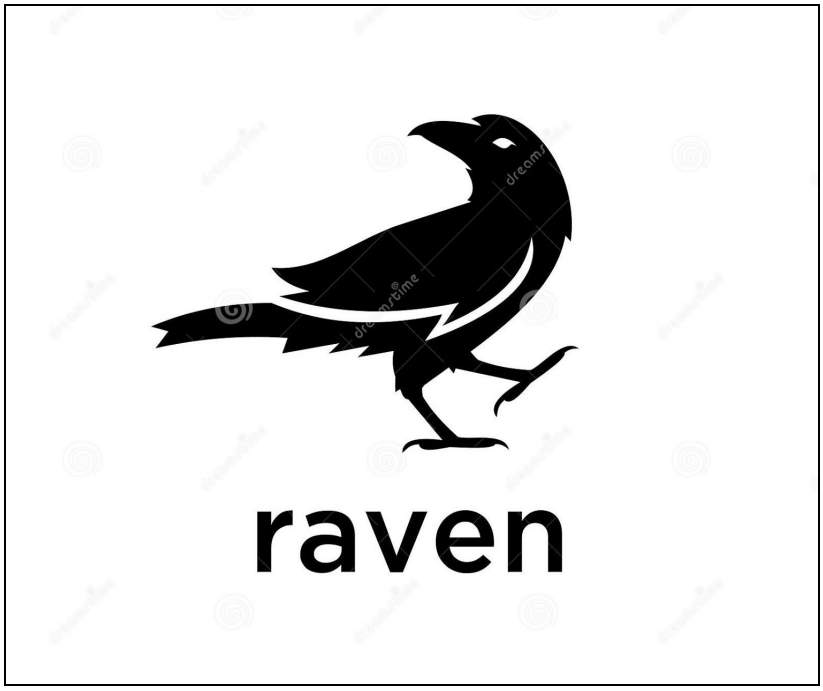
Raven

Delivered to: The University of Texas at Dallas



Created by: Alfredo Meija, Carson Sharp, Mohammed Alotaibi, Mustafa Sadriwala, Natasha Trayers (MASST Inc.)

02/26/2020

Table of Contents

1 Business Landscape

The Raven application is requiring a development team to implement a new mobile app to allow users to post items, alerts, and interact with other users. They have an idea for how the system is all meant to work together, and they have decided on the needed hardware for the development of the infrastructure. What Raven needs is a streamlined solution that ties together their software(application) with their servers following the needed requirements. RVN has been tasked with developing this software within the scope outlined in the Statement of Work document.

2 Introduction of Document

This document will present the solution engineered by RVN through the entire software life cycle. The solution includes two main components that work synchronously: Raven Application, and Raven Databases. Each component is broken down in detail from requirements to development and deployment.

3 Requirements

MASST Inc will work with The University of Texas at Dallas to create a mobile application for users to engage in a mobile student community to communicate important notifications, alerts, and information campus-wide as well as set up a communication system to stimulate a virtual marketplace. This SOW will cover the design, development, and deployment of the Raven application for UTD.

3.1 Functional Requirements

Front-End Elements

1. User shall be able to register an account with UTD SSO
2. User shall be able to login and logout of the app
3. User shall have an interactive walkthrough of steps and functionalities when they login for the first time
4. User shall be able to add tags to the bottom of posts using '#' character
5. User shall be able to select visibility for the news, alerts, and messages that they post
    a. User shall be able to choose to post to friends, campus, or current location
6. User shall be able to add and accept friend requests
7. User shall be able to create friends circles/cliques/"conspiracy"
8. User shall be able to privately message individuals
9. User shall be able to send and accept private message requests to and from users who have not accepted friend requests
10. User shall be able to search through posts using tags.
11. User shall be able to view most recent posts or top posts in social media style feed on the home page
12. User shall be able to sort and filter through posts using Username, Location, Type Of Post, Date, and Popularity
13. User shall be able to like or dislike posts
14. User shall be able to report inappropriate content
15. User shall be able to like or dislike comments
16. User shall be able to view and update their profile information
17. User shall be able to see their previous posts on their profile
18. User shall be able to make posts and categorize them into general posts, items for sale, or alerts
19. User shall be able to make comments on posts
20. User shall be able to save/bookmark items for sale
21. User shall be able to browse items for sale by categories
22. User shall be able to switch views between their profile, private messages, public posts, and marketplace
23. User shall be able to upload photos from camera roll
24. User shall be asked for permission for notifications, location services, and camera usage

Back-End Elements

25. Data is backed up on a constant basis (Utilizing UTD's hosted database system (RDS))
26. Reports will be generated to University officials for complaints of content moderation
27. Server to handle real-time messaging, commenting, and alerts
28. UTD SSO Login integration
29. Cross-Platform native functionality integration for photo, location access, and notifications

3.2 Non-functional Requirements

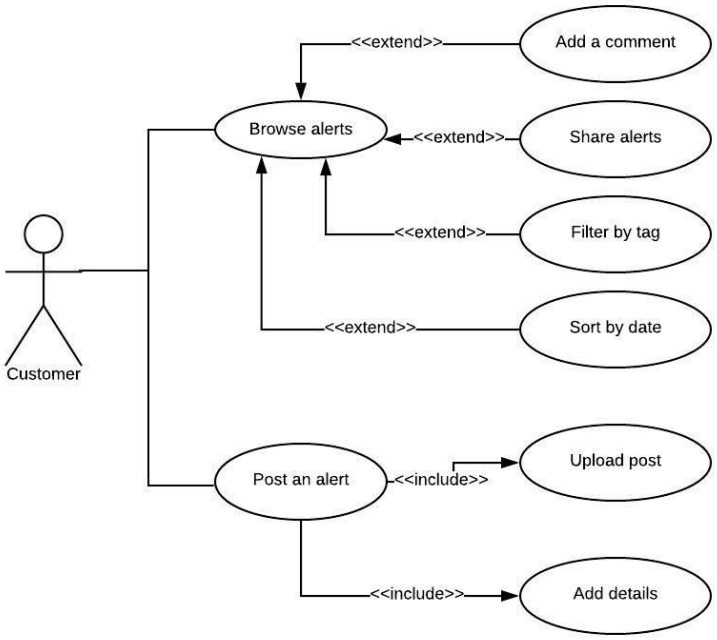
1. App should load within 3 seconds
2. Users should be able to login with UTD SSO within 5 seconds
3. The account should lock after 5 failed attempts
4. Our mean time between failures (MTBF) should be over 24 hours
5. Data should be handled in accordance with Fair Information Principles
6. Maintenance - during the project scope - should respond to issues within 12 hours
7. User should be able to post to the app within 5 seconds
8. Data should be backed up every 6 hours during 8am - 8pm and once between 8pm - 8am
9. The app should not be 'down' for more than 3 hours at a time
10. The app should always be available for usage unless brought 'down' for maintenance or critical bug
11. Server should be able to support up to 50KB of data per post
12. Server should be able to support up to 500KB of data per item for sale
13. 1000 users should be able to simultaneously login into the app

4 Design

Anything not mentioned in Section 3 is considered to be out of scope of the project.
Examples of out-of-scope activities include the following:

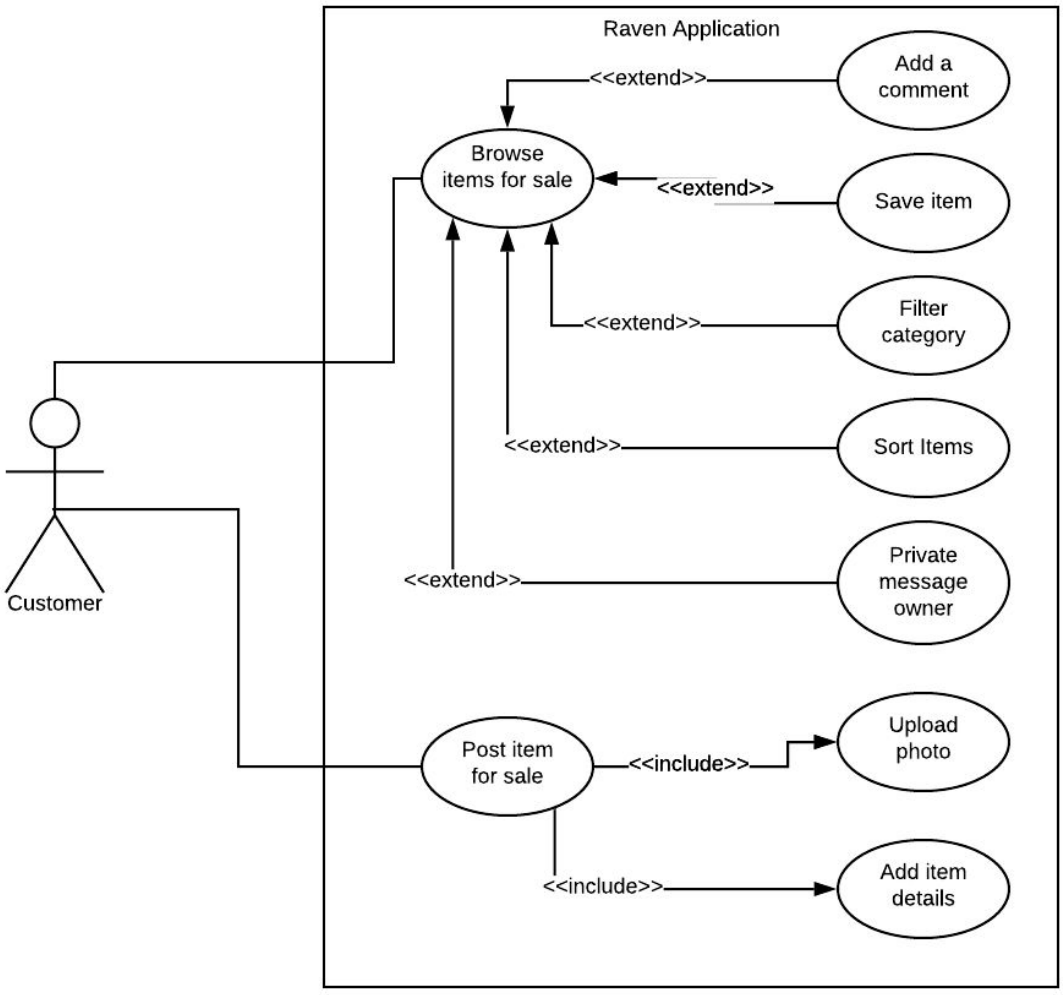
- Photo-editing functionalities for uploaded photos
- Providing services to non-current UTD students
- eCommerce platform for buying and selling goods
- Transporting goods that are for sale
- Calling law enforcement for criminal activity alerts

4.1 Use Case Diagrams



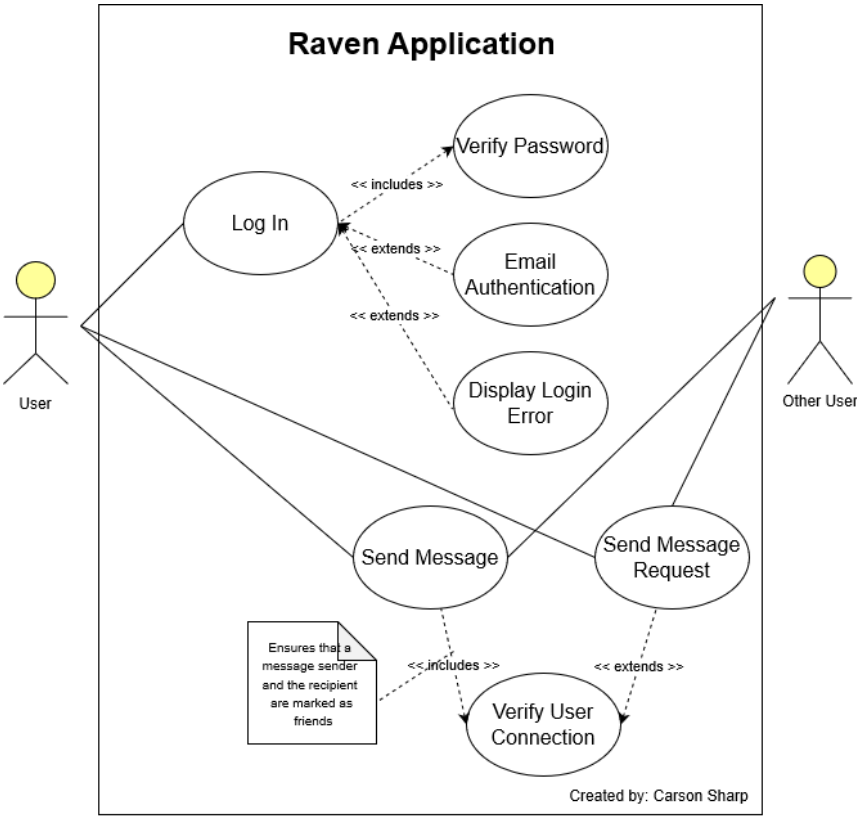
Created by: Mohammed Alotaibi

The use case diagram shown above represents two use cases in the alerts section of the app.



Created by: Natasha Trayers

The use case diagram shown above represents two use cases in the marketplace section of the app. This diagram shows a user browsing items for sale which can be facilitated by the filtering by categories and sorting items functionalities. A user browsing items can also save items to their profile, add a comment and directly message the owner to talk about purchasing. The second use case is posting an item for sale which requires the user to upload a photo and add details for the item.



The use case diagram shown above represents the use case for logging in to the Raven application and the use case for sending a message between two users.

4.1.1 Use Case Descriptions

Post Item for Sale	
Actors	Customer
Description	The customer can post an item for sale on the local marketplace portion of Raven to be viewed by other Raven users. The item for sale will include a picture that the customer must upload and details of the item (including its price, name, and previous use)
Data	Information about item for sale
Stimulus	User command issued by customer
Response	Confirmation that item has been posted
Comments	The item should be available to be viewed on the customer's profile page in the future.

Created by: Mustafa Sadriwala

Send Message	
Actors	Mobile User, Other Mobile User
Description	The user can send a message including keyboard characters and emojis to another user privately. The application verifies that the message sender and recipient are friends on the application before sending the message. The message can include an attachment of a file that is limited to 8 MB.
Data	The content of the user's message
Stimulus	Input from a user that the message is ready to be sent
Response	Confirmation that the message has been delivered, Notification to the recipient that a message has been received
Comments	If the user sending the message has not friended the user receiving the message, a message request is sent in lieu of the message.

Created by: Carson Sharp

Browse item for sale	
Actors	Customer
Description	A customer browses items for sale on Raven's marketplace that are posted by other users. The feed can be filtered and sorted to make the search process easier. Individual items can also be saved for later reference. While browsing, the user can comment on item posts and message the user that posted the item directly from the item detail page.
Data	Item photo and details such as price, date and condition; Owner profile which contains contact information
Stimulus	Command issued by customer
Response	Confirmation that search results are updated
Comments	Items will initially be sorted upon opening the app by most recent and relevance based on previous searches

Created by: Natasha Trayers

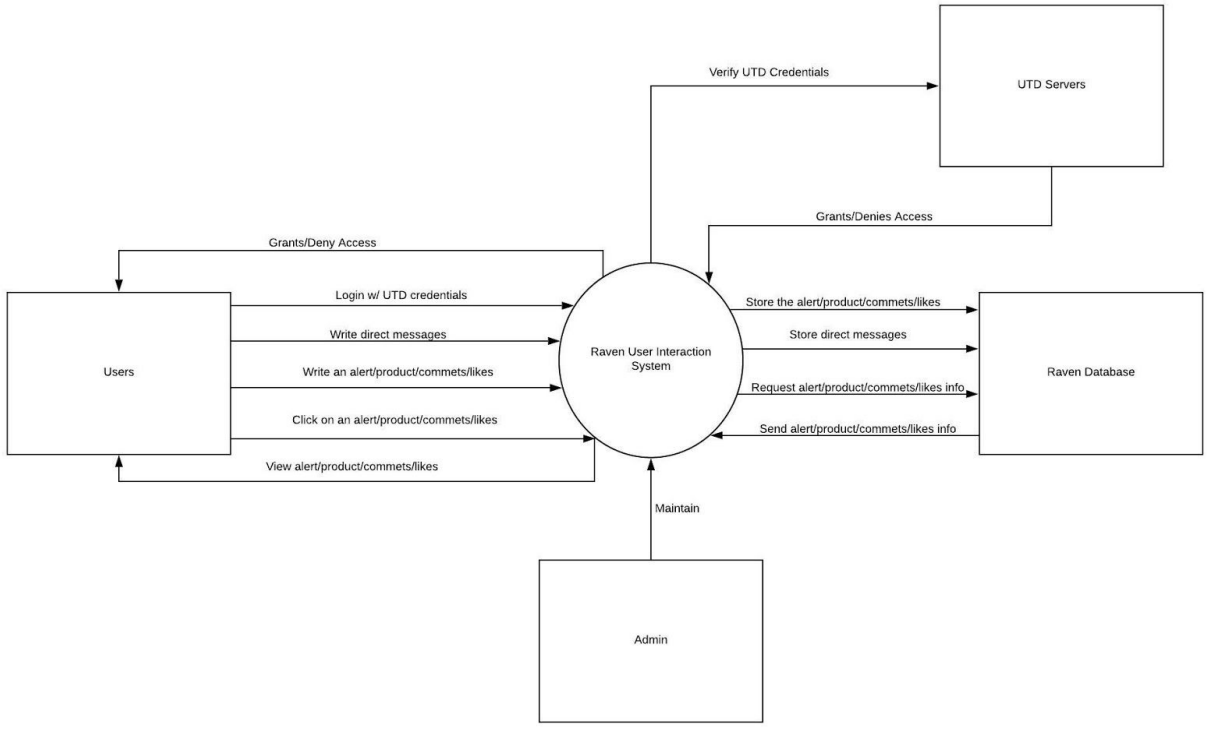
Post an Alert	
Actors	Customer
Description	The customer can post an alert on the alert section of Raven to be viewed by other Raven users. The alert will include a timestamp and tags to help with categorizing and finding the alerts
Data	Information about the nature of the alert
Stimulus	User command issued by customer
Response	Confirmation that the alert has been posted
Comments	The alert should be available to be viewed on the customer's profile page

Created by: Mohammed Alotaibi

Log in	
Actors	Customer
Description	The customer must enter his/her UTD credentials into the Raven application in order to allow access to the application. The Raven application will then communicate with the UTD servers in order to verify the user's email and password. If it was first-time logging in then the Raven application provide email authentication.
Data	User's UTD Credentials
Stimulus	Command issued by customer
Response	Confirmation of a successful/unsuccessful login
Comments	The customer must be a UTD student in order to be given permission to access the application and retrieve customer information by the UTD servers.

Created by: Alfredo Meija

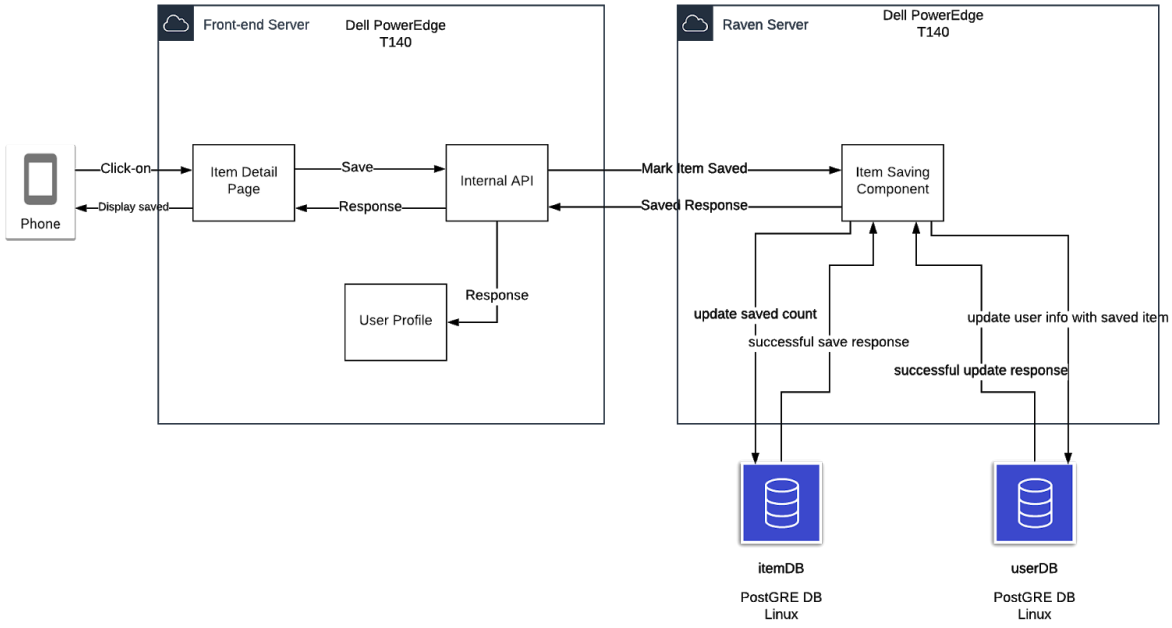
4.2 Context Diagram



Created by: Alfredo Meija, Mohammed Alotaibi

The context diagram provides an overview of the Raven User Interaction process. On the user side, it can include several functions such as logging in, sending and receiving messages, posting and viewing items and posts, writing comments, and more. On the database side, it stores and retrieves the data stored such as images and texts for the posts, comments, likes, and more. Furthermore, the Raven Interaction process includes logging in with UTD credentials and having an admin to maintain the system.

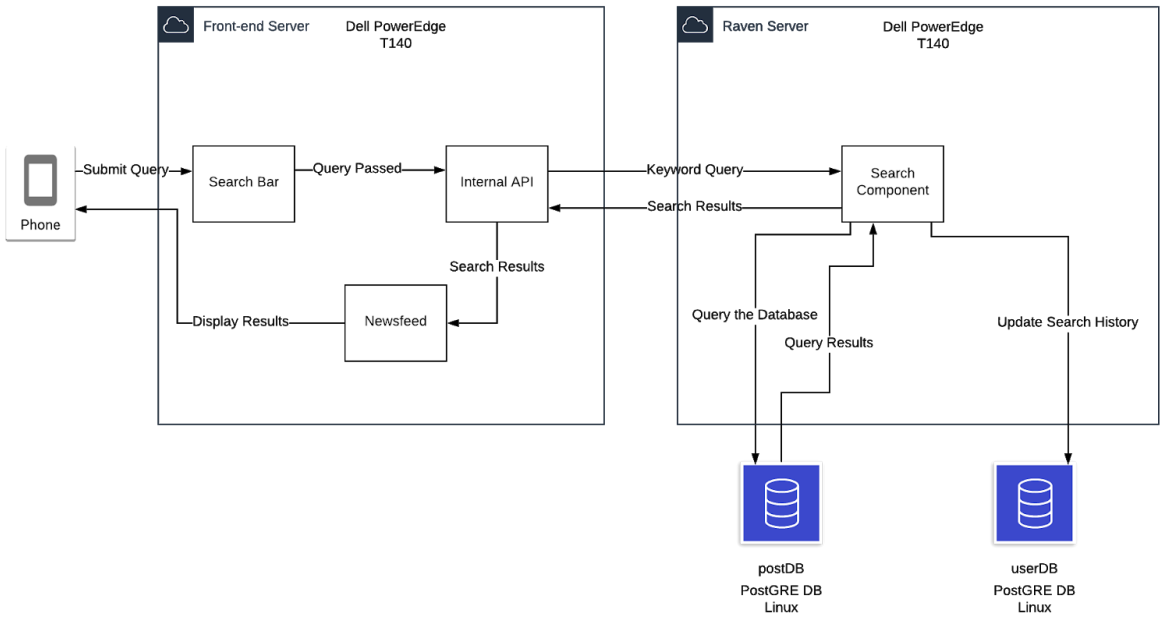
4.3 Mid-level Diagrams



Created by: Mustafa Sadriwala, Carson Sharp, and Natasha Treyers

The above mid-level diagram shows how an item is saved. The user engages the Item Detail Page by clicking on the ‘Save’ feature which calls the internal API and routes to the Raven Server. The internal API calls the Item Saving Component which interacts with both the itemDB and the userDB to update the saved count of the item and the user info, respectively.

Front-end Server: The front-end server handles input from the user, runs the internal API, and displays all output to the users
Raven Server: The Raven Server receives instructions from the Front-end server, manages the databases, and outputs to the Internal API
itemDB: Item database consists of all inventory information for items for sale.
userDB: User database consists of user table, user-item table and user-post table.

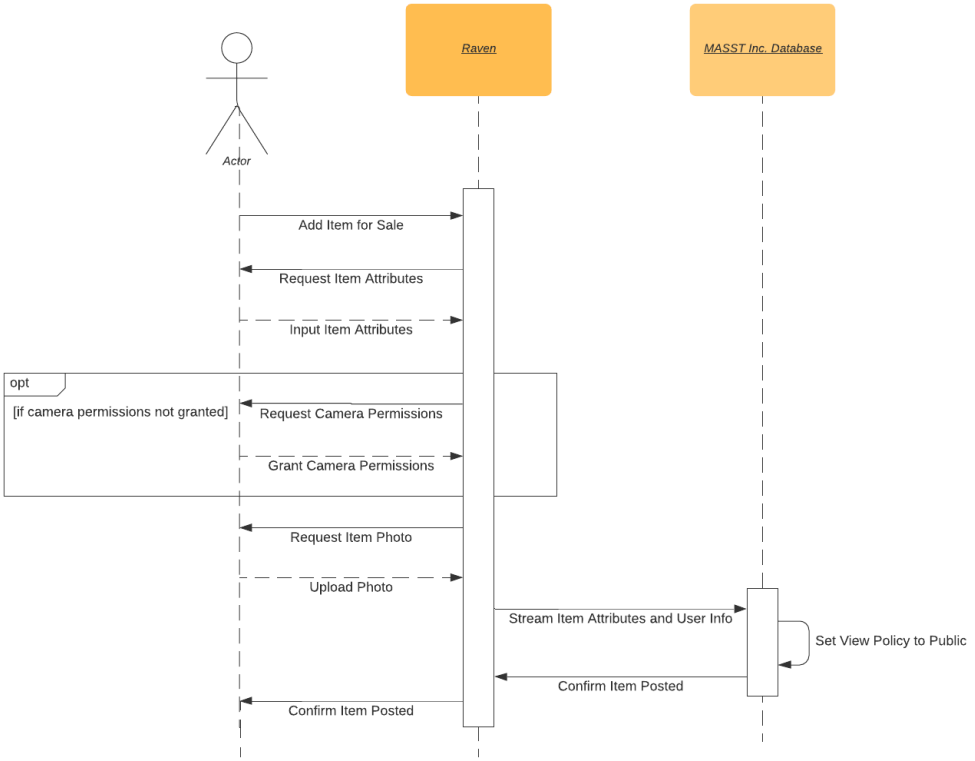


Created by: Mustafa Sadriwala, Carson Sharp, and Natasha Treyers

The above mid-level diagram shows the process of a user submitting a search query for a post. The user first submits a query into the Search Bar which is then passed to the Internal API. The Internal API will call a keyword query on the Search Component in the Raven Server which will update the user's search history in the userDB and query the postDB for the actual results. These query results will then be returned to the Internal API and subsequently passed to the Newsfeed which will be displayed to the user.

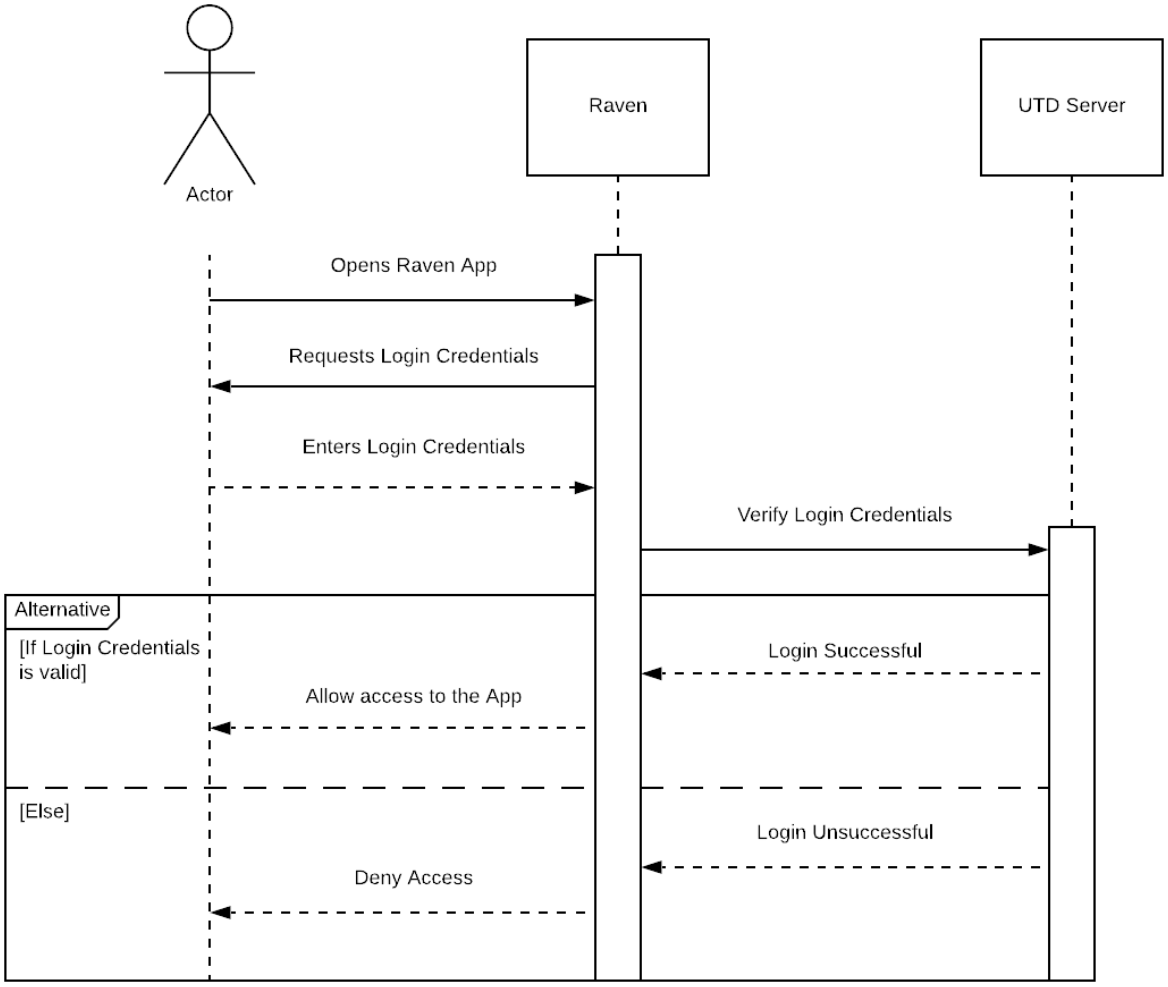
postDB: Post database consists of post information including alerts, news, and general posts.

4.4 Sequence Diagrams



Created by: Mustafa Sadriwala

The sequence diagram above describes the process of a user posting an item for sale and it's photo to Raven for other users to see. The sequence takes into account the scenario when a user may have not yet granted camera permissions, in which case, Raven must ask the user to grant camera permissions.



Created by: Alfredo Meija

The sequence diagram above shows the processes of logging into the Raven application. The Raven application requests the UTD credentials from the customer. Once the input has been received it verifies the customer's information with the UTD servers and responds to the customer by either allowing access to the app or denying access to the app.

5 Coding

The Raven app involves both backend and frontend coding. In the "Coding" section, we will explain our plans and strategies in both ends. We will first talk about our technical language/framework selections and why we made those choices. We will make a few examples of projects components and talk about our solutions and how to achieve our goals in these components. Then, we will state a rough schedule for our coding sprints. Meanwhile, we will also clarify each team member's responsibility in this project.

5.1 Frontend - Mobile App Development

In the choice of frontend development environment, we decided to use React Native as our base to implement the application because React Native has a large and comprehensive documentation online. React Native is open source so we can find various existing components and libraries for us to use to better layout and decorate our mobile app. Also, React Native is cross-platform compliant as it automatically handles JavaScript code suitable for each platform. Looking at it from a saving aspect, React Native is cost effective since we don’t need to use two separate codes for both platforms as both OS can be coded with single programming language. Since we need to build a mobile application that works on different electronic devices with different operating systems, React Native is a good pick. In addition, React Native has a modular architecture allowing for flexible development, reusability, and establishes better coordination with each other to get the updates.

5.2 Middle-end - Server-side Application

We will use Express.JS application to host our back-end server that will connect both the Front-end React Native application and the backend MySQL. We will use HTTP requests to communicate between our front-end and middle-end. We are using Express.JS because it offers a stable, secure, and quick server application that will also constrain the overall number of languages our development team will need to be familiar with.

5.3 Backend - Database Management System

For our database management system, we have elected to use MySQL. Since we have already created requirements specifications and functionalities for the application we can confidently use a relational database without fear of changing our schema design often. MySQL is fast, secure, and incredibly scalable. It fits most of our needs as a small to mid-size business and allows our application the potential to grow quickly beyond one university. It also has good compatibility with React Native and exceptional documentation.

We will also use SQLite for local device storage purposes which is compatible with both MySQL and React Native. SQLite is the simplest solution for local storage as it is an industry standard and meets security and scalability requirements.

5.4 Overall Development Technologies

For development purposes we will be using Android Studio and VSCode for optimal coding and testing. We will need at least one Apple MacBook to use Xcode so that we can emulate and test our mobile app in an iOS environment. We will use Android Studio to emulate an Android environment for testing.

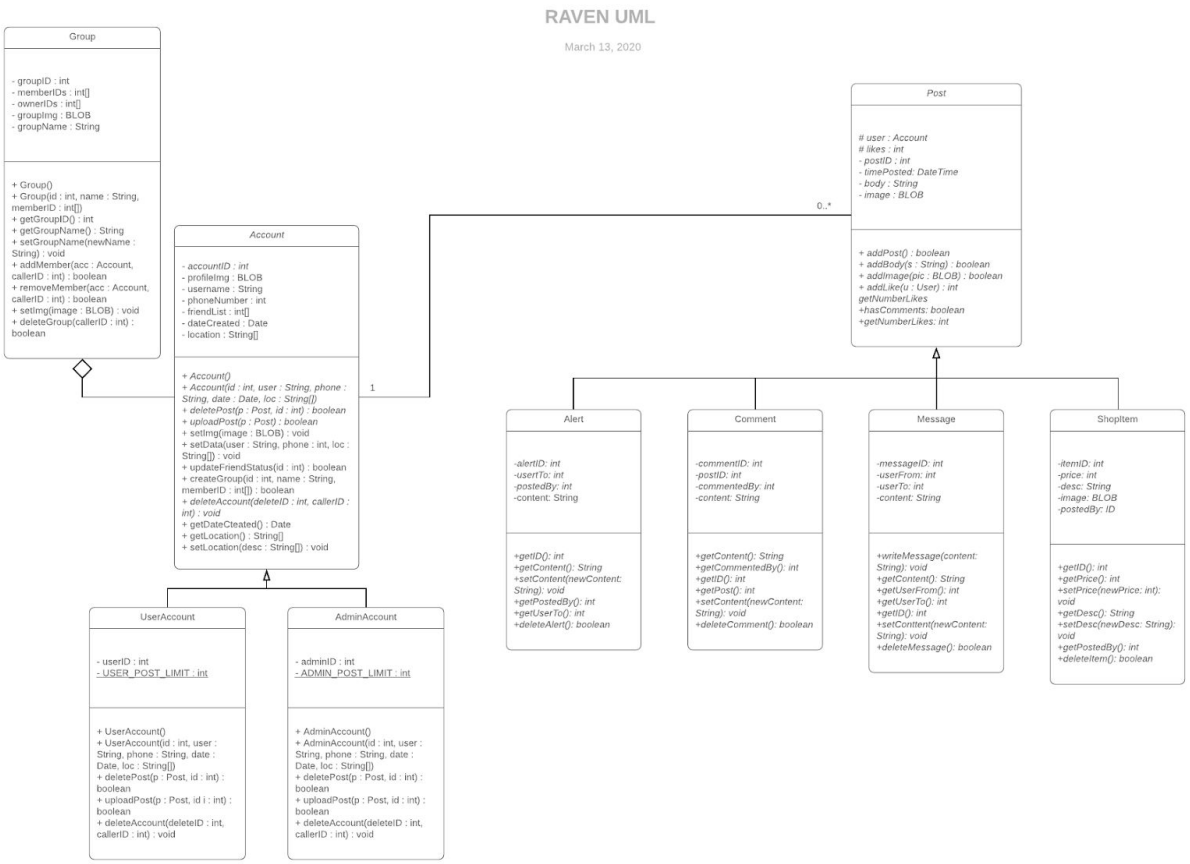
5.5 Coding Process

The formal coding process for the Raven app starts at 8/15/20, the start date of the 2020 fall semester. We will be using the agile methodology with 2 week sprints. We aim to finish the framework by the first sprint, login and sign up by the second sprint, user interface by the third sprint, the filtering and search components by the fourth sprint, location options in the fifth sprint, posting, liking, commenting, on posts by the sixth sprint, and finalize the project by the seventh sprint.

5.6 Work Delegation:

The initial work delegation will be the following: Mustafa and Natasha will tackle the back-end while Alfredo and Mohammed work on the front-end. There will be lots of collaboration so we will not only work in our designated teams. Carson will be in charge of creating and maintaing the databases and their corresponding tables.

UML Diagram:



Created By: Carson Sharpe, Natasha Trayers

6 Testing

6.1 Test Plan

Testing Process

Testing will be broken into phases corresponding to different phases of development:

1. Regression Testing
    a. Unit Testing
    b. Component Testing
    c. System Testing
2. Scenario Testing and Performance Testing
3. Alpha Testing and Beta Testing
4. Acceptance Testing

The phases will be worked progressively. Once one phase is finished, the testing team will move to the next phase. For the first phase, the development team will be conducting the testing as the program is developed incrementally. This ensures after each sprint the changes made to the program does not introduce new bugs. In regression testing, the development team will also do unit testing for the methods and objects just completed in the sprint and will also do component testing with several interacting objects. As development continues and the solution is capable of system testing, the developers will integrate components to create a version of the system and then test the integrated system. After the final sprint, the program will be tested via two teams. One team will create scenarios and test for those scenarios and the other team will do performance tests. As the product is almost ready to release, the testing team will continue with Alpha and Beta testing and finish with acceptance testing.

Requirements Traceability

Testers will be required to sign off on individual tests linking them to the requirements they originate from. If tests can no longer be linked to a requirement, tester should inform team lead to fix any discrepancies such as a changed or removed requirements that can cause a testing failure.

Testing Technologies

Jest

For testing the our React Native Mobile Application we will be using Jest. Jest was developed by Facebook in tandem with React Native and so is the default choice for testing our front-end application. Jest comes with rich documentation, fast implementation and mocking, and parallel execution. It is designed to solve issues within our JavaScript codebase.

Sinon

For testing the Express.JS we will be using Sinon, because of its ability to work with any unit testing framework, Sinon was our first choice. A super clean integration with minimal setup, Sinon turns mocking requests into a simple easy process.

Docker

For testing the MySQL Database we will be using Docker, because of its ability to package units into standardized applications called containers, Docker makes testing individual components and functionality simple. The ease of setup and continuous integration tools make Docker an effective tool for testing updates and retrievals for all Raven Databases.

6.2 Test Cases

Functional Test Cases				
#	**Name**	**Corresponding Requirement #**	**Execution**	**Acceptance Criteria**
F1	Comments on Post Test	19	Input a string as a comment to a post	1. The UI of the post will change to indicate it has a comment 2. All users should be able to see the comment on the post
F2	Categorize Posts on Main Feed Test	18	Click on a button in the main feed that will allow the user to organize posts according to the given options	1. The UI of the main feed should change to indicate the user the posts are being organized. 2. The user should see the posts organized according to the option selected.
F3	Viewing and Updating Profile Information Test	16	1. On the main page click on your profile picture to view your profile information. 2. On the personal profile page click on the edit icon “pencil” to update and change personal information. Then click on the save button at the bottom to	1. The app should change pages and should show the profile page. On that page, the user should be able to see his personal information, profile picture, settings icon, and edit icon. 2. The profile page UI should change to make the user’s personal profile information editable. Once the save button is clicked the information is saved and the

			save information.	information is no longer editable. The updated information should be viewable for those allowed.
F4	Search Through Posts Using Tags	10	1.On the search bar using the "#" sign. 2. You can search relatable terms that helps you find the posts you are looking for.	1. The user should be able to see the posts that are related to the tags they searched.
F5	View Most Recent Post in Social Media Style	11	When you open the app, you can scroll through the posts from the most recent post till the oldest post.	1. The user should be able to view the posts with the most recent posts.
F6	Sort and Filter Through Posts Using Username, Location, Type of Post, Date, and Popularity	12	On the main page, using the search bar a user can sort and filter their search. Specifying either the username, location, type of post, date and popularity.	1. The app must accept the search criteria and show posts based on the filters put on by the user. 2. The user must be able to view the posts based on the filters placed.
F7	Like and Dislike Posts	13	A user is able to press on the thumbs up and thumbs down button on a post.	1. The UI of the post will change to indicate it has a like or a dislike 2. All users should be able to see the like or dislike on the post
F8	Report Inappropriate content	14	If a user finds a post inappropriate, using the help icon, they can report it and have it taken down.	1. The app must have the reporting feature icon next to the post. 2. The user must be able to write exactly what is wrong with the post.
F9	Bookmark an item for sale	20, 22	1. In the items-for-sale feed click on the save button on an arbitrary post 2. go to User Profile 3. View Saved Posts	1. The recently saved item for sale should appear at the top of the list of Saved Posts on the User Profile. 2. Clicking on the save button should change the save icon from an outline to a filled-in shape for that item's post.

				3. The user should be able to move from items-for-sale to user profile.
F10	Categorize items for sale	21	1. Click on search bar in items-for-sale feed 2. Click a category to filter by 3. View items-for-sale feed	1. The items-for-sale feed should only show items tagged under the chosen category 2. All previous sort features should still apply to the items-for-sale
F11	Upload photo	23, 24	1. Make a post or create a new item for sale 2. Click to Upload Picture 3. Choose a picture from phone's gallery 4. Click save	1. The user's selected photo should be displayed besides the newly-created post/item for sale 2. The user should be asked for permission to access photos if this is their first time 3. The photo should be viewable to everyone who can see the post/item for sale
F12	Add/accept friend requests	6	1. Add someone as a friend 2. Accept a friend request	1. The other user should receive a friend request in the notifications tab 2. The other user should appear in the friends list on the profile page
F13	create friends circles	7	1. Add several individual to a friends circle	1. All individuals should receive a notification and be added to a group chat in the chat section
F14	privately message individuals	8, 9	1. Privately message an individual that is already on the friends list 2. Privately message an individual who is not on the friends list.	1. The other user should receive the message in the chats section and get a notification 2. The other user should receive a message request from the individual in the notification section
F15	Register Process	1	1. Attempt registration with non-UTD SSO 2. Register with example UTD SSO and complete walkthrough.	1. Registration fails on the first attempt 2. User is registered on the second attempt.
F16	Log In and	2, 3	1. New account is	1. Walkthrough is given on the

	Log Out Test		registered using a UTD SSO. 2. Walkthrough is completed by input. 3. The user is logged in and out repeatedly.	first successful login attempt 2. All proceeding logins are successful and the walkthroughis never shown.
F17	Post Settings Test	4, 5	1. Post is created on a sample account with sample tags. The post is submitted using the "news" setting. 2. The same post is sent once using the "alert" setting and again using the "message" setting.	1. Posts contain all submitted tags and are only visible to the appropriate accounts.

Mohammed Alotaibi:F4,F5,F6,F7,F8
Alfredo Meija: F1, F2, F3
Mustafa Sadriwala: F9, F10, F11
Natasha Trayers: F12, F13, F14
Carson Sharp: F15, F16, F17

Non-Functional Test Cases				
#	**Name**	**Corresponding Requirement #**	**Execution**	**Acceptance Criteria**
NF1	User Post Speed	7	1. Make a post with an empty string and no picture 2. Make a post with a picture of max dimensions and resolution 3. Make a post with a picture of max string length 4. Make a post with both max string and max picture size	1. All posts should be received by database. 2. User should receive confirmation of post being uploaded within 5 seconds of pressing Submit 3. Post should be viewable to other within 5 seconds of original user pressing Submit
NF2	Data Back up	8	1. Create some new data at 8pm (post, item for sale, or user profile) 2. Create some new data at 8am 3. Create some new data at 2pm	1. Data uploaded between 8am - 2pm should be viewable on back up hardware by 2pm 2. Data uploaded between 2pm - 8pm should be viewable on back up hardware by 8pm 3. Data uploaded between 8pm - 8am should be viewable by 8am
NF3	Downtime Test	9	1. Run the application, server, and database continuously for at least 100 days and track failures and the time needed to restore the system	1. The whole system should be fully restored within 3 hours of the crash of the system.
NF5	Scalability	13	Simulate 1000 virtual users logging into the app concurrently	The application should be accessible with minimal delay in response time. Maintain an average transaction time of five seconds at 1000 simultaneous users.

NF6	App Load in 3 Seconds	1	When activating the app, the home screen must be shown and ready for usage in around 3 seconds or less.	The user will be able to access the app in 3 seconds from opening it.
NF7	Login with UTD SSO within 5 Seconds	2	The app will be using UTD SSO to login, the user must be in the app in less than 5 seconds.	The user will be able to use the UTD SSO to login and will be logged in within 5 seconds.
NF8	Lock Out Test	3	1. Attempt to log in to a sample account using an incorrect password repeatedly. (Can test case sensitivity with password choice)	1. Login process is halted after the fifth automated attempt.
NF9	MTBF Test	4	1. Application is left running with a varying level of requests throughout the day. Activity is simulated for at least a full week. Upper bound of activity is tested.	1. Failures are only detected once every 24 hours on average.

Mohammed Alotaibi:NF6,NF7
Alfredo Meija:NF4
Mustafa Sadriwala: NF1, NF2
Carson Sharp: NF8, NF9
Natasha Trayers: NF5

7 Deployment

7.1 Sprint Deployment

In each sprint, the development team shall complete independent functions that can be tested and deployed directly. The completed parts will be viewed and tested by the sponsor to get permission for the next sprint. The deployed pieces will be hosted on the internal Raven Servers for Front-End and Back-End.

7.2 Project Deployment

By finishing all sprints and milestones, the development team will test the whole project and finally deploy it. The final Raven back-end server will be hosted on a DELL PowerEdge T140. The final application will be built from React Native into native Android and iOS builds. These builds will be provided to the sponsor to view and to make an agreement with the development team representing the completion of the project. Once all processes above have completed, the development team will hand over all documentations, source codes, data, and servers to the client and the Android and iOS builds will then be published respectively on the Google Play Store and Apple App Store.

The documentations included with the project are the Statement of Work, Requirements Documentation, Design Documentation, Software Engineering Documentation and Source Code Documentation. Source codes will include the front-end React Native application and the back-end Express.JS server-side application. The development, design, and testing teams will also be responsible for training employees in the client company’s maintenance team.

After deployment, the project will be considered completed and the development team is no longer responsible for maintenance of the source code. If further requirements are requested by the client then a new contract will be drafted detailing the beginning of a subsequent project.

7.3 Technologies

Our deployment will not require any technologies external to what we have already used. We will deploy both our Front-end and back-end applications onto DELL PowerEdge T140s. Our React Native application will be built into native Android and iOS versions and then deployed respectively onto the Google Play Store and Apple App Store.

7.4 Potential Issues and Resolutions

Bugs associated with the deployment environment could be a potential issue that would not be caught in testing. All tests could be run on a system similar to the users to ensure that their platform is compatible with our system. Inconsistencies could arise between the iOS and Android versions of the application. The deployment process to both platforms should be carefully done in parallel to each other. The installation program may not be built correctly which could lead to a missing or corrupted script being installed. The installation process should be tested and inspected to ensure deployment is accurate and complete. Effective communication between the deployment team and the client over all documentation and maintenance team will be necessary to ensure a lack of communication does not lead to confusion.

8 Maintenance

The objective of maintenance is to ensure correct operation, management, performance enhancement, and fixture of any failures and bugs presented. Maintenance will begin after deployment and MASST Inc will assign this project to the maintenance team under an administrator. This administrator is responsible for scheduling and overseeing all maintenance-related work, managing a maintenance team, and drafting maintenance reports. This will ensure the system is fully functional and working securely at optimal performance.

8.1 Regular Maintenance

For daily operation, the maintenance team will monitor the status of servers and databases to ensure correct operation. The server will be monitored by checking the server's performance and analyzing it for any security threats. If any issue is found, the maintenance team will update the server to enhance performance and security. The database will be monitored by checking the performance, storage levels, security threats, and back-ups of the application. The maintenance team will act accordingly to its administrator either by updating the database software, writing back-ups, flushing out the cache, or flushing out the ports.

In addition, the maintenance team will also monitor the environments the solution is running in to ensure the solution is compatible with any operating system update that may occur. If any incompatibility occurs, the maintenance team will provide an update to the solution to allow for it to run on the mobile operating system. The maintenance team will also monitor the mobile application and find ways to improve the performance of the app especially in the areas lacking performance.

The maintenance team is also tasked with solving any bugs, erros, and any unauthorized access. The maintenance team is going to regularly check for bugs such as security bugs to protect the app from any unauthorized actions and comply with policies. When solving bugs and errors, the maintenance team should also focus on optimization, efficient storage use, and refactoring. The updates made by the

maintenance team must ensure it does not affect other aspects of the program and will closely monitor such updates until deemed successful. If the maintenance team can't solve the issue, it will contact the administrator to contact the development team for assistance. The maintenance team will monitor many bugs through customer feedback and performance of the mobile application.

Finally, the maintenance team will have scheduled system maintenance in order to solve any undetected performance leak and/or bugs. The regular maintenance of the solution is monitoring the application, servers, and databases to solve any issues that may arise such as bugs, security flaws, performance leaks, and inefficient memory usage. Monitoring these objects will consist of a combination of user reports and continuous tests such as performance tests, security tests, etc. Any major update or modification of the application is outside of the scope of the maintenance team. Also any small update or adjustment made by the maintenance team must follow any licenses and policies in place and a small report must be generated to the maintenance administrator.

8.2 Failure Maintenance

The administrator, the development team and MASST Inc management are responsible to deal with any unexpected system failures. The maintenance team is constanting monitoring the servers, databases, and mobile application, and if any unexpected failures occur the team should be aware of it. The main goal is to reduce any financial damage and recover the services as soon as possible. Also, failure tolerance should be the most important aspect for regular maintenance. The Mean Time to Repair (MTTR) must be as low as possible to ensure the Mean Time to Recovery is within 3 hours in accordance to the non-functional requirement. If unable to restore services, the administrator must be able to communicate with the development team and other departments to swiftly restore services. After recovery, a report is generated and given to the development team, maintenance team, and management to ensure that Mean Time to Failure is maximized while keeping the Mean Time to Recover within 3 hours.

8.3 Resources

In terms of human resources, we will need adequate technical knowledge to cover all the bases of the project. For the purposes of front-end maintenance, there should be a few maintenance team members that are knowledgeable on React Native applications and their development and design patterns. Furthermore, there should be someone who has previous experience in database management and ideally mobile database management to allow proper maintenance of the back-end databases. The middle-end, server-side application simply requires a working knowledge of JavaScript to understand the codebase. It would be ideal to also have someone equipped with architectural knowledge of the application that has experience using Express.JS with React Native applications.

In terms of technical resources, the maintenance team will use a suite of programs and tools that will allow for efficient and timely maintenance. These tools include Program Slicers, which can select only

parts of a program affected by a change, and Cross-References, which generate indices of program components. The team will also use Static Analyzers, which allow general viewing and summaries of a program content, Dynamic Analyzers, which allow the maintainer to trace the execution path of a program, Data Flow Analyzers, which allow the maintainer to track all possible data flows of a program, Cross-References, which generate indices of program components, and Dependency Analyzers, which help maintainers analyze and understand the interrelationships between components of a program.

8.4 Potential Issues

Potential issues could be running out of disk space which may require the maintenance team to delete log files. If there is an authentication error, or a host not found error, there will be the need to update configuration data. Modifications to the environment such as updates to the server or operating system may require the maintenance team to reinstall the application. Another potential issue would be if the UTD servers go down and the app isn't able to authenticate a student since it is using SSO with UTD. In this case the maintenance team would need to contact the UTD system administrator. Accumulated bugs could lead to the application crashing so bug fixes must be done in a timely manner. The maintenance team will need to make sure the database doesn't get filled up, and make back-up as there could be a power outage. The application could also lose compatibility with the latest devices and operating systems so the maintenance team must incorporate the latest hardware features.

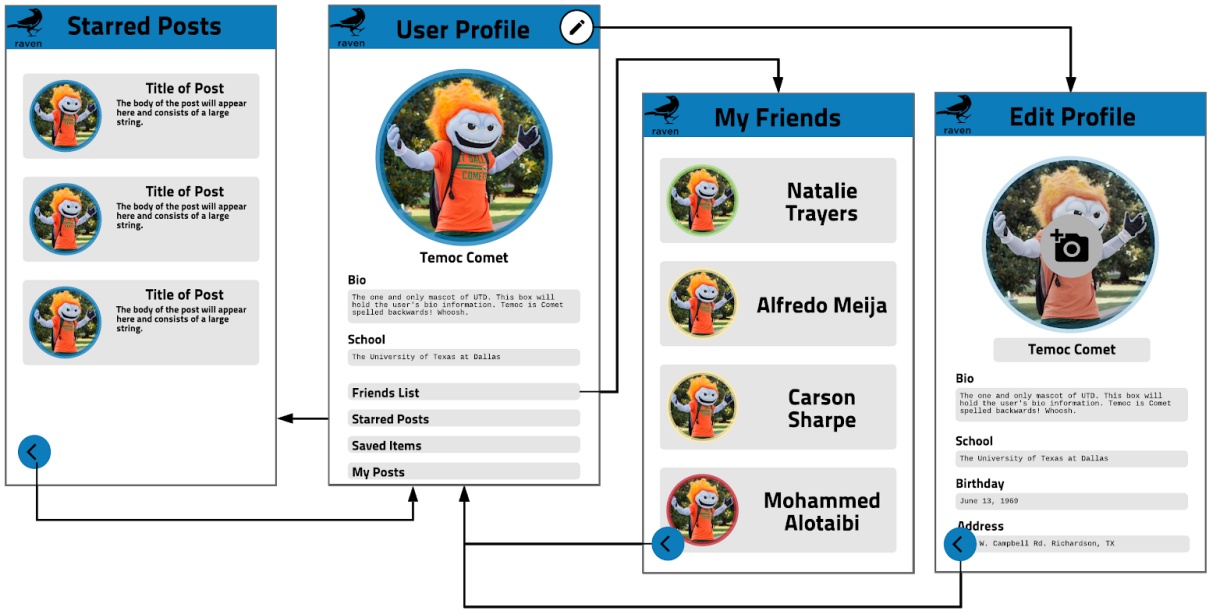
8.5 Transition

After the final sprint, the product will be handed off to the maintenance team. There will be different teams for maintenance as we will need a subset of people who can oversee a project as maintenance maintains multiple projects. The development team will provide the tools to be able to maintain the system, ensure it works, fix bugs, and add very small features. They will also provide documentation on how to check for errors to keep the system maintained and how to respond to specific alerts. They will have instructions on how to patch a system if there is a bug and how to rebuild and redeploy it. Maintenance will first review requirements, designs and understand the code, how it was written and the classes. They will also need to understand the configuration tools and how they work. The development team will show the maintenance team automated testing, what testing tools are used, the test data and how it was deployed.

9 WireFrames

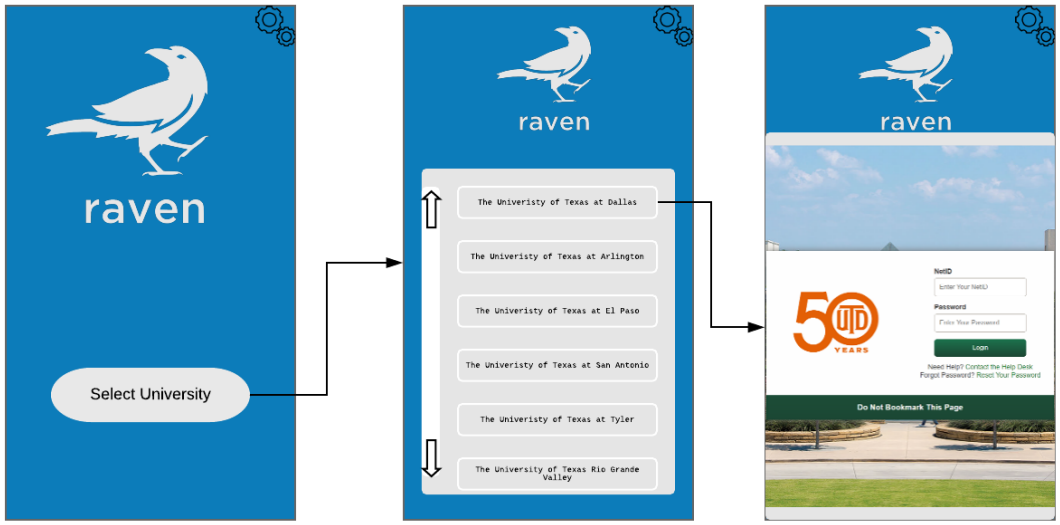
Below are the wireframe diagrams that have been created by MASST Inc. in reference to this project. These are not final designs but rather an interpretation of the requirements and offered as a means to better visualize the final product.

9.1 User Profile



Created By: Mustafa Sadriwala

9.2 Login

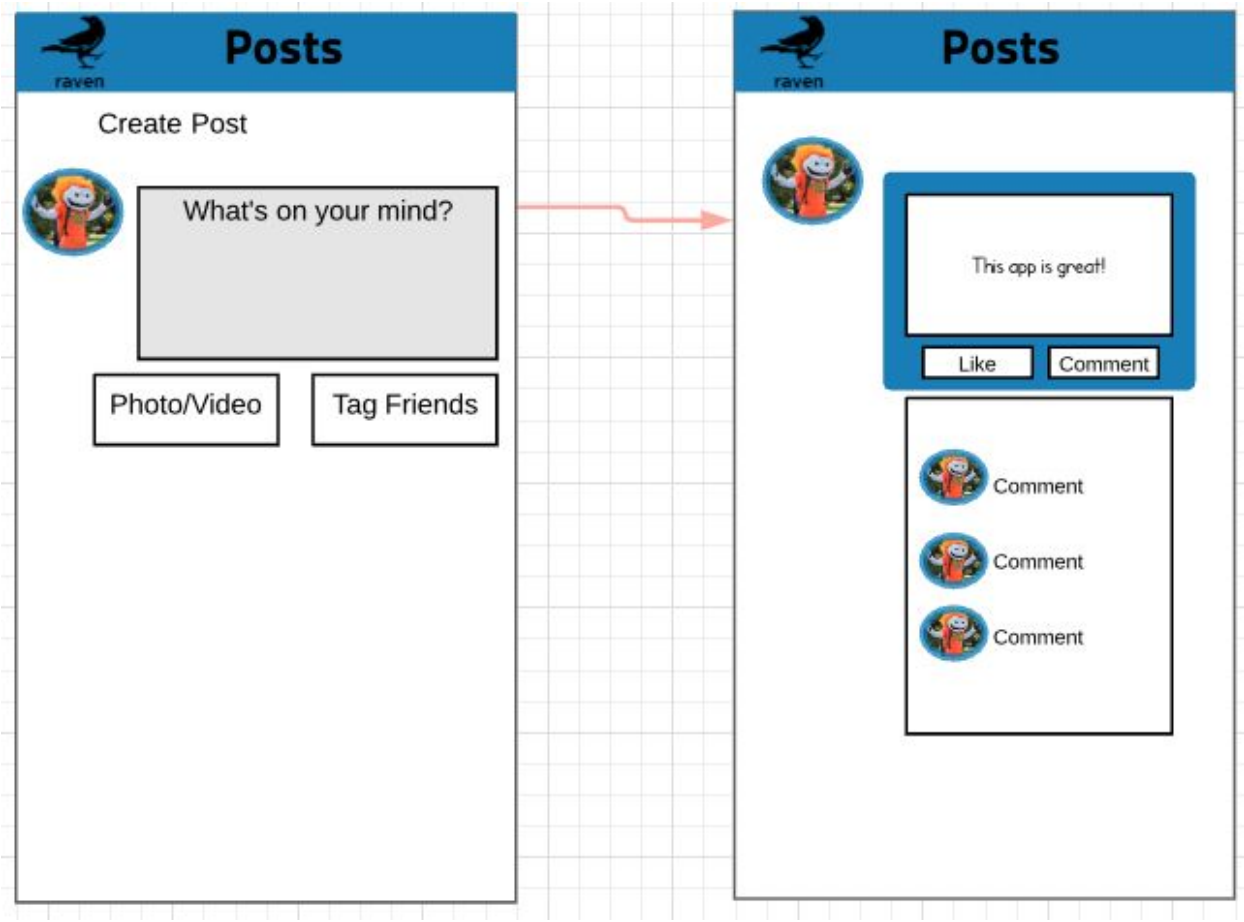


Created By: Alfredo Meija

9.3 Posts Feed

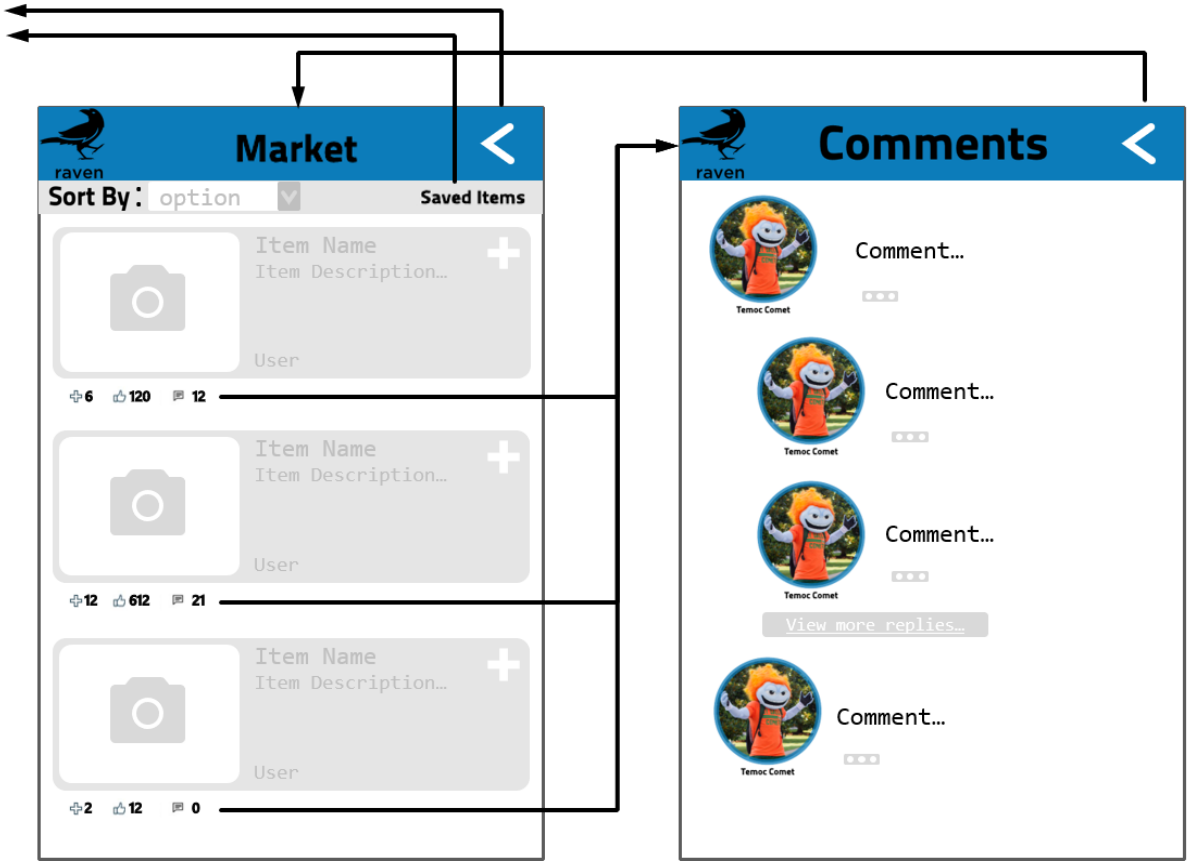
Created by: Natasha Trayers

9.4 Post Creation



Created by: Mohammed Alotaibi

9.5 Items-For-Sale Feed



Created by: Carson Sharp