TeleTag-Requirements - 487W

Cameron Bussom, Love-Divine Onwulata, Mahima Abraham, Greg Salisbury

September 10 2023

1 Team Info and Policies

1.1 Roles of Team Members:

- Cameron Bussom - Front End and Bluetooth
- Mahima Abraham - Front End
- Love-Divine Onwulata - Bluetooth
- Greg Salisbury - Back End

1.2 Project Artifacts

- Git Repo - Click to go to the Git Repo
- Proposal PDF - Click to go to the Proposal PDF

1.3 Communication channels

- Team will meet either in person or on zoom to work on the project.
- Using Github to organize all of our documents and code.
- Use Monday.com to keep track of due dates, who is working on what, and tasks to be done.
- Use discord to communicate about the project and develop a knowledge base for collaboration when necessary.

2 Product Description

- TeleTag is a mobile app that people can download to play tag with their friends. The game will have different rooms that players can join to play tag with each other. Their phones' Bluetooth will help them find other players and once they are in range their tag button will be enabled. One the player is out of the range to tag another player, their button will be

disabled. There will be a leader board that displays the player's rankings, wins, and losses.

2.1 Four Major Features

- Detecting Bluetooth devices
- Connecting to a Bluetooth device
- Detecting proximity of Bluetooth devices
- Sending data through Bluetooth devices
- Recognizing devices that are in the current game

2.2 Two Stretch Goals

- Different Game Modes
  - Games that have conditions that once met end the game. i.e Elimination mode, Multiple taggers, etc
- Map System
  - A service that can track all players in proximity to the user

3 Use Cases

3.1 Use Case 1: User Starts a Game

- Actor: User
- Overview: A user wants to play a game with n other users. The user connects all mobile devices via Bluetooth
- Trigger: A user and n other users want to play TeleTag.
- Precondition: n users want to play a game of TeleTag.
- Post-condition: N users phones have the data and Bluetooth connections of n-1 users
- Steps:
  1. N users gather to play a game of TeleTag
  2. One user presses the host game button
  3. The users not hosting, gather in range of the host user's device's Bluetooth range
  4. The phones connect via Bluetooth
  5. The of of all the users phones are sent to each device

- Extensions
  - One user decides to not play, and does not connect to the game
  - Another user decides to play as well, and connects to the other phones
- Exceptions:
  - One or more user's device is not connecting to the other phones correctly and is kicked from the lobby

3.2 Use Case 2: User Leaves a Game

- Actor: User
- Overview: A user wants to leave the current game permanently. The user clicks on the 'Leave Game' button. The user's device no longer transmits data to other devices.
- Trigger: A user wants to disconnect from the game
- Precondition: A user wants to stop playing the current game of TeleTag.
- Post-condition: The user's device no longer transmits data to the other devices in the game
- Steps:
  1. The user stops running
  2. The user presses the 'Leave Game' button
  3. The device transmits to the other devices that it is leaving the game
  4. The device stops transmitting data
  5. The database removes the player from the player database
  6. The devices of the other players no longer transmit to the former player's device
- Extension
  - The user's device stops working or dies and is removed from the player database
- Exception:
  - The user's device fails to update the player database, and the other users phones continue to send data

3.3 Use Case 3: The Host Ends the Game

- Actor: User
- Overview: The host decides that the game should end and presses the 'End Game' button. The host's device sends a signal for the other devices to disconnect from the game and stop transmitting data.
- Trigger: The host presses the 'End Game' button
- Precondition: The host user wants to end the game
- Post-condition: The users' devices stop transmitting data to one another
- Steps:
  1. The host presses the 'End Game' button
  2. The host device transmits to the other devices that the game has ended and instructs them to stop transmitting data to one another
  3. The other devices receive the instruction to end the game and stop transmitting data
  4. The player database removes all the player
  5. The game ends
- Extension
  - All the users' devices stops working or dies and are removed from the player database, ending the game
  - The game goes on for too long, and the player database removes all the players and ends the game
- Exception:
  - The devices are not updated on the game's status and continue to transmit data as if still in the game

3.4 Use Case 4: A User is 'Tagged'

- Actor: User
- Overview: The user that has the tag gets in range of another user. The tagger then presses the 'Tag' button, transferring the ability to tag to the other player. The ability to tag then goes on cool-down for a set time, preventing the new tagger from tagging another player until the cool-down is finished.
- Trigger: The tagger, in range of another player, hits the tag button
- Precondition: The tagger is in tagging range of a player

- Post-condition: The tag has been transferred from the user to the other player
- Steps:
  1. The tagger gets in range of a player they desire to tag
  2. The user presses the 'Tag' button
  3. The tag is transferred to the player
  4. The player receiving the tag is notified that they have been tagged
  5. The tag then goes on cool-down for a set time
- Extension
  - There are multiple players in range, and the user has to select which to tag
  - The tagger presses the 'Tag' button, but the tag is in cool-down resulting in nothing happening
- Exception:
  - The tag is not in cool-down, and another player is in range, but the device does not transmit the tag

4 Non-Functional Requirements

1. Usability: An easy and efficient game that players can easily join the game and connect with other users to play the game this will allow for anyone to be able to play TeleTag
2. Scalability: Multiple players are able to join at once and choose if they want to join a current game or start a new one. The limit of players will be salable to however many players are in the same vicinity of each other.
3. Security/Privacy: Database that stores players' information that cannot be accessed by anybody but the player themselves. This information will not have any person information only game information so nothing personal will be accessed by TeleTag.

5 External Requirements

5.1 Tools Being Used and Why:

- Android Studio - A great Android app developer that has different tools and features, that is constantly updated, which can be used to develop a good apps. Also the Android emulator helps us to see what the app will look like as we design it without having to use an actual android device.

- Kotlin - Google's preferred programming language for Android app development, Kotlin helps us code more efficiently, is easy to pick up, and provides code safety.
- SQLite - Is an open-source database engine that is already installed on Android devices. We can use SQLite to easily create a database to store player information. Due to our familiarity with MySQL, and since this software is recommended by Android, it seems like the obvious choice for the project.

5.2 Team Roles and Justification

- Cameron Bussom - Front End and Bluetooth
  - A front end developer is needed in order to be able to create different functions for the players in the game such as the history tab and tag button. The role of developing the Bluetooth is needed in order to connect the players' phones and to determine the range of each player. Cameron has experience creating many different types of front ends and has an interest in Bluetooth technology which would make him perfect for the task
- Mahima Abraham - Front End
  - Another front end developer is needed for the project because there are multiple aspects needed for the game to look user friendly and to display different information such as the game tab and tying the database to the game. Mahima's experience in front-end work and interest in learning more front-end skills can help in making the game more enjoyable and the development of it faster.
- Love-Divine Onwulata - Bluetooth
  - Another Bluetooth developer is needed because it is one of the major aspects of our product. It is needed to find other devices, detect the range of players, and to tag other players. Having two people assigned to this tasks will help the Bluetooth function of our app to work properly and efficiently. Love's interest in learning new technologies and his experience in Java, a language similar to Kotlin, the language being used, makes him perfect for this task.
- Greg Salisbury - Back End
  - A database needs to be developed to satisfy the objective of creating a leader board for the game, as well as a history of who got tagged by who, and so any required data can be accessed from a central location. Being able to store information during the game makes the game more enjoyable for the players, who can be more focused on the game itself, and not remembering who's winning or losing.

Greg's personal interest in back-end development and background in MySQL should make the development of the database faster and more efficient.

6 Milestones

6.1 Front End Tasks:

1. Set Up History Tab (9/13/23)
2. Create Game Tab (9/13/23)
3. Tie Database to History Tab (9/29/23)
4. Improve the layout of the application (10/2/23)
5. Add Tag Button (9/13/23)

6.2 Back End Tasks:

1. Setup Database (9/22/23)
2. Create columns for player (9/22/23)
3. Create multiple game rooms (10/16/23)

6.3 Bluetooth

1. Find Other devices using Bluetooth (9/22/23)
2. Make sure the player is in range to tag other players (9/29/23)
3. Find how to send data through Bluetooth (10/2/23)

7 3 Major Risks

1. Bluetooth - Risk of not being able to use Bluetooth to send data, detect range, and to connect players.
2. Database - Having trouble with storing the players' information in the database and then not having a leader board that players can use for rankings.
3. Software - Risk of not having enough time to develop our app into a polished piece of software due to lack of knowledge of tools being used, or slow implementation of major parts of the project.

8 External Feedback

- External feedback is most crucial after the game is developed and ready to be used by others to test the quality of the game and make sure the features are user friendly and intuitive to use.

9 Software Architecture

9.1 Major Software Components and the Functionality

1. User Interface
    - This component provides the user interface for the app, allowing users to interact with it. It includes screens, buttons, and other elements that enable users to perform actions such as tagging devices and configuring settings.
2. Bluetooth Communication
    - This component manages Bluetooth communication with other devices. It scans for nearby Bluetooth devices, and establishes a connection with them if they are registered. Once the connection is established, the devices exchange data.
3. Device Tagging Logic
    - This component is responsible for determining when to tag devices based on their proximity. It defines the range two users have to be in for a tag to transfer. It also detects when a device gets in range, then triggers the tagging process.
4. Tagging Management
    - This component is responsible for managing the tags associated with each device. It stores info about the tagged devices, as well as updates which devices are tagged when needed.
5. Notifications
    - This component handles notifications to inform users about tagged devices entering or leaving the specified range. It pushes notifications to the user when they are tagged
6. Device Information
    - This component collects and displays information about connected devices, such as Device identification, Device Status, and logs related to tagged devices.

7. Data Storage

    - This component is responsible for storing data related to tagged devices and app settings. It could use local storage or cloud-based solutions.

9.2 Interfaces Between Components

1. Bluetooth to User Interface

    - The Bluetooth communication layer interfaces with the User Interface to display nearby devices, initiate pairing, and manage Bluetooth settings. The Bluetooth sends updates to the User Interface about the status of device connections and Bluetooth-related notifications.

2. User Interface to Bluetooth

    - The UI interfaces with the Bluetooth communication layer to relay user input related to multiplayer actions, such as initiating connections or selecting opponents.

3. Bluetooth to Database

    - The Bluetooth communication layer may interact with the storage component to save and retrieve game data that needs to be synchronized between devices.

4. User Interface to Front-End(other components)

    - The UI sends user input, joining a game room, pressing the tag button, etc., to the game logic(Front-End) for processing.Game logic updates the UI with the current state of the game, including player scores, character status, and level progress.

5. User Interface to Back-End

    - The UI sends requests to the back-end for tasks like detecting other devices via Bluetooth, fetching leader boards, or synchronizing player progress.The UI can initiate user authentication and account-related actions with the back-end.

6. Back-End to the Front-End(Game Logic)

    - The back-end sends game-related data to the game logic, including player profiles, scores, and game state from remote players. The back-end updates the game on player's actions, like a new person has been tagged, a game has ended, etc., so the game can say synchronized.

7. Front-End to Database

    - The front-end interacts with the database to retrieve essential data, such as player profiles, game progress, high scores, and achievements. The front-end displays the retrieved data from the database through leader boards, user profiles, etc.

8. Back-End to Database

    - The back-end component is responsible for handling game-related data on the server side, including player accounts, game state synchronization, and real-time game play data. It ensures that data is appropriately synchronized between all connected players.The back-end communicates with the database to update and retrieve leader board data.

9.3 Data The System Stores

- Player's Data:
    - Our database will mainly store the players' progress within the game, their statistics, such as wins, losses, etc,. It will also store players' current status such as if they are currently in a game, if they just completed a game, if they just won a game, etc. This data will all be stored in a leader board that players can access to see where they stand among others.

- Game Data:
    - Stores information about active game rooms, including players involved and game settings. Records of completed games, including player actions, scores, and outcomes. Also stores configuration details for each game, such as time limits, number of players, and game play rules.

9.4 Particular Assumptions

- The game will be designed for Android devices running Android 5.0 or higher, to leverage newer Bluetooth and Android features. Older Android versions may have limited support.
- The maximum number of players in a game is assumed to be 10, based on typical Bluetooth limitations. The game design and architecture should account for up to 10 connected players.
- Players are assumed to remain within Bluetooth range (up to 30 feet) of each other for the duration of a game. Longer distances may lead to connectivity issues.

- Fast, reliable internet connectivity will be required for features like leader boards, cloud storage, or account management. The architecture should handle intermittent connectivity gracefully.
- The game will use Bluetooth Classic for connectivity. Bluetooth Low Energy may offer benefits but has limitations on Android.
- Typical mobile hardware limitations like processing power, memory, battery life must be considered. Performance optimization will be important.
- The architecture should isolate device-specific dependencies to allow porting the game to other platforms like iOS in the future. Business logic should be platform agnostic.
- Security considerations like authentication, authorization, and data encryption will be required, especially if cloud services are integrated.
- Offline modes may be necessary to allow gameplay with no internet connectivity. The architecture should support both online and offline modes.

9.5 Alternatives for Software Architecture

1. SQL Lite

   - An alternate for SQL Lite is Realm which also supports Kotlin. It is a mobile database designed for Android and other mobile platforms. It offers real-time data synchronization and object-oriented data modeling. It's known for its simplicity and performance.
     - Advantages:
       (a) Realm uses an object-oriented data model, which aligns well with game development, making it easy to work with game entities, player profiles, and game state.
       (b) Realm is known for its speed and efficiency, which is crucial for games that require real-time or near-real-time interactions, especially in Bluetooth multiplayer scenarios.
       (c) Realm offers real-time data synchronization capabilities, which can be useful for keeping game data in sync between devices connected via Bluetooth.
       (d) Realm provides a straightforward and intuitive API for data storage and retrieval, which can simplify the development process.
     - Disadvantages:
       (a) Our game relies on local Bluetooth interactions and doesn't require extensive real-time synchronization between devices so Realm's real-time features might introduce unnecessary complexity.

(b) Realm databases can become relatively large over time, so you should be mindful of storage requirements for your game, especially if you plan to support long gaming sessions.

(c) Realm lacks support for SQL queries, which may limit your ability to perform advanced and complex database queries.

(d) Can make handling complex data schema changes or migrations more challenging compared to traditional SQL databases due to Realm's simplicity and efficiency.

2. Kotlin

- An alternate for Kotlin is Java which offers excellent compatibility with Android. Android Studio has robust support for Java. It provides a wide range of tools, debugging capabilities, and integration with Java libraries.
- Advantages:
  (a) If there are existing code bases or libraries written in Java that needs to be used, using Java for the TeleTag game can simplify integration.
  (b) Java provides the Android Bluetooth API, which allows you to work directly with Bluetooth hardware and manage Bluetooth connectivity, making it suitable for building Bluetooth-enabled games.
  (c) Java can be used to write cross-platform code that is not limited to Android. This can come in handy if we want to extend the game to other platforms.
- Disadvantages:
  (a) Java is known for its verbose syntax, which can lead to more lines of code compared to more modern languages like Kotlin. This can make development more time-consuming and error-prone.
  (b) Java lacks some of the modern language features found in Kotlin, such as data classes, extension functions, and smart casting. These features can enhance code readability and maintainability.
  (c) While Java is widely compatible with Android devices, compatibility with the latest Android features and libraries may not be as seamless as with Kotlin.

10 Software Design

10.1 Detailed Definition of Software Components

1. User Interface

   - XML files define the structure and arrangement of UI elements using layout containers such as Relative Layout, Linear Layout, and Constraint Layout. XML files define UI elements like buttons, text views, image views, and custom views.
   - 'MainActivity' or 'GameActivity' classes represent screens or game screens and manage UI elements, navigation, and user interaction. Adapters are used for displaying lists or grids of data. They connect data sources, such as arrays or databases, to UI elements.
   - Styles and themes define the visual appearance and behavior of the app's UI elements and can be customized to create a unique look and feel for your game. Layout variations can be created for different screen sizes and orientations, ensuring your UI looks good on a variety of devices.

2. Bluetooth communication

   - 'BluetoothAdapter' represents the device's Bluetooth adapter and provides methods for enabling or disabling Bluetooth, discovering devices, and obtaining information about the device's Bluetooth capabilities.
   - 'BluetoothDevice' represents a remote Bluetooth device that your app can connect to. You can obtain Bluetooth Device instances through device discovery or by using paired devices.
   - 'BluetoothSocket' is a connection point for Bluetooth communication between two devices. It's used to establish a connection with another device for data transfer.
   - UI components such as buttons, switches, and dialogues are used to control Bluetooth functionality and display connection status.
   - Error and exception handling logic is essential for dealing with potential issues that can occur during Bluetooth communication.

3. Device Tagging Logic

   - Player class Represents a player in the game. It may include properties like player ID, username, score, game status, etc.
   - Player Adapter or Player Manager class: Manages the list of players and their interactions.
   - Tag or Tagging Event class represents a tagging event, including the tagger and the tagged player.

- UI elements like buttons, player avatars, score displays, and game screens for tagging and score keeping.
- Classes or components for managing game states (e.g., pre-game, in-game, post-game) and transitions between states.
- Classes or settings for configuring game parameters, such as game duration, scoring rules, and game mode.
- Classes or components to define and enforce the rules and mechanics of tagging, such as how tagging affects a player's score or game progress.
- Logic for determining when the game is over, calculating final scores, and displaying end-game results.
- Classes and components to store game progress, player profiles, and high scores, if required.

4. Tagging Management

    - Player Manager or Player Controller class: Manages the list of players, their locations, and interactions.
    - Tag or Tagging Event class represents a tagging event, including the tagger, the tagged player, and the timestamp.
    - UI elements like buttons,score displays, and game screens for tagging and score keeping.
    - Classes for handling Bluetooth or network communication to exchange tagging events and game updates between devices.
    - Classes or components to define and enforce the rules and mechanics of tagging, including how tagging affects a player's score or game progress.
    - Classes and components to store game progress, wins, losses, etc.

5. Notifications

    - Notification Manager class manages the creation and delivery of notifications in the app such as the winner of the game, who has been tagged, etc.
    - 'NotificationCompat.Builder' class will be used to build and configure individual notifications. It allows you to set the notification's content, title, icon, and other attributes.
    - Classes for different notification styles, such as BigTextStyle for expanded text or InboxStyle for multiple messages.
    - Classes to define custom actions within notifications, such as "Play Game," "View Leader board," or "Dismiss."
    - Templates or layouts for various types of notifications, including simple text notifications, expandable notifications, and custom notifications.

- Classes or components to manage notification sounds, vibrations, and other alerting behaviors.

6. Device Info
    - A utility class or set of functions for retrieving device-specific information. This class may provide methods for accessing device details.
    - Classes or components for managing permissions related to accessing device information. Android requires explicit permissions to access certain device data, and these permissions must be managed in the 'AndroidManifest.xml'.
    - Classes or functions to obtain screen size and resolution information, which can be important for adapting the game's UI to different devices.
    -
7. Data Storage
    - Player, Game, and other data model classes to represent the game's entities. These classes define the structure of the data to be stored.
    - 'SQLiteOpenHelper' or a custom database helper class to create and manage the game's SQLite database.
    - Classes or constants to define the structure of database tables, including table names, column names, and data types.
    - Classes and methods for serializing game data into formats like JSON or XML and deserializing data for storage and retrieval.

11 Coding Guideline

1. SQLite - Click to go to the SQLite Coding Guidelines
    - It is an excellent coding guideline to use because it provides platform-specific guidance, best practices, and practical examples that help developers create efficient, secure, and robust database-driven Android applications. We plan on enforcing these guidelines by using it for our own code so we can build an app that is reliable and easy to use. Using these guidelines can help each group member understand what the code is doing and can make it easier for each of us to correct the code when necessary.
2. Kotlin - Click to go to the Kotlin Coding Guidelines
    - This is an excellent coding guideline to use because it is created by Google and ensures that the guidelines are reliable and well-vetted. It offers Kotlin coding guidelines that are specifically tailored for Android app development. It also provided great practices and will

helps us avoid common errors and have an efficient and reliable code. We are going to be enforcing these guidelines by using it to build our app so we can avoid as many mistakes as possible when trying to get the app up and running. This guideline will help us understand what the common mistakes are and how we can avoid them. So if one of the group members are to have trouble figuring out how to fix an aspect of the app one of us will know how to using the guideline.

12 Process Description

12.1 Risk Assessment

1. Risk 1 - The Bluetooth not being able to run and detect other devices to connect to.

    - Likelihood of occurring: low
    - Impact if it occurs: high
    - Evidence the estimates are based on: We are still figuring out some aspects of the Bluetooth implementation and it detecting other devices. Once we handle the errors and get it fully up and running it should be able to detect other devices and if it fails the impact would be very high because without devices being able to connect, the game is not playable since it is a multi-player game.
    - Steps taken to reduce the likelihood or impact: We are watching multiple tutorials and grabbing code from other sources, modifying it for our needs and implementing it so we can handle as many errors as possible lowering the risk of it not working. Watching tutorials and grabbing code from different sources is more efficient because we do not have enough knowledge or time on Bluetooth in order to build it from scratch.
    - Plan for detecting the problem: Every time we modify the code to handle different exceptions, to run the Bluetooth, etc., we run the app to see if the changes we made worked or if it is throwing an error. Running the app after each change helps us keep track on where we might have went wrong or where we need to add another function.
    - Mitigation Plan: If the issue occurs we would go through the code and see where the error might be occurring. We would add error handling, if applicable, to ensure the Bluetooth connection can run smoothly. We would also debug the code to catch any errors that might have been missed and improve the Bluetooth connection.

2. Risk 2 - Database Not Being Able to Store Player's Data

    - Likelihood of occurring: medium
    - Impact if it occurs: high
    - Evidence the estimates are based on: Right now we do not have information to store in our database because we have not gotten the Bluetooth fully functioning just yet. Once we get that working we would be able to test it out between two Android devices and see if the database can store player's data such as one's wins, losses, etc. Once we have the Bluetooth is up and running we still have a chance of the database not being able to store information due to not handling different errors and other aspects of the code.
    - Steps taken to reduce the likelihood or impact: To reduce the impact or likelihood we can design a well-structured database schema that efficiently stores player information. Normalize the database to reduce data redundancy and improve data consistency. We can also implement strict data validation to prevent the entry of incorrect or malicious data into the database which helps maintain data integrity and security. Also can implement robust error handling and reporting mechanisms in your code to detect and address database-related issues promptly.
    - Plan for detecting the problem: Implement automated tests, including unit tests and integration tests, that specifically target database interactions. These tests should simulate common scenarios and potential issues, such as failed inserts, and record the results.
    - Mitigation Plan: Regularly back up the database in case any of the information already stored in the database gets lost. Thoroughly investigate the cause of the database failure, including examining logs and error messages. Identify the root cause of the issue to prevent its recurrence. Conduct regular database maintenance and optimization to ensure the database is in a healthy state and free from potential issues.

3. Risk 3 - Technical Issues - Compatibility With Different Android Phones

    - Likelihood of occurring: low
    - Impact if it occurs: high
    - Evidence the estimates are based on: The code we are using to build this phone app in Android Studio is built to be compatible with any Android phones and makes it possible to adapt to new Android Features.
    - Steps taken to reduce the likelihood or impact: The steps we have taken so far to reduce the likelihood or impact is by following the coding guidelines for Kotlin and SQLite which helps us implement

the latest code possible which makes it easier to create an app that is compatible with various Android phones. With the app we have in place right now we have tested it on two different Android phones and as of right now the app runs and both devices which indicates it is compatible with multiple Android phones.

- Plan for detecting the problem: To detect problems we need to add error handling within the program so we can avoid any problems that might come when trying to run the app on different phones. We also run the app every time we make a change to see if there might be a new error due to the changes. That way we can keep track of what errors we need to fix.
- Mitigation Plan: Constantly debug the code to fix any errors that might have been missed. Constantly run the app on different Android phones and see if there are problems and if so what is causing it.

4. Risk 4 - Phone App Not Being User Friendly - User Interface

    - Likelihood of occurring: low
    - Impact if it occurs: high
    - Evidence the estimates are based on: We have added different buttons and game tabs for the app that is currently up and running. Once we have data the different tabs will make it easier for the user to access any data they want to. We are constantly following different tutorials and staying up to date on different feature we can add to the game app to make it more user friendly like a leader board, column for players, multiple game rooms, etc.
    - Steps taken to reduce the likelihood or impact: We run the application to make sure that the different features being added are working properly. We also compare our game to other similar apps out there and see what features they have to make theirs more user friendly and their user interface function properly. Also will add error handling and constant debugging the code any errors can be caught and fixed efficiently and quickly.
    - Plan for detecting the problem: Having other people run our program and get their feedback on what we could add or change for our game. Having others use our game as we are developing it can help us improve it constantly to make our User Interface as efficient as possible.
    - Mitigation Plan: If something were not to run, such as our tag button not working, we would debug the program and see where the error must be. We will stay constantly updated on different features we can add and have a backup plan to fixing the problem an error handling.

5. Risk 5 - Network Connectivity Issues

    - Likelihood of occurring: low
    - Impact if it occurs: medium
    - Evidence the estimates are based on: Our game involves leader boards and other formats to display player data that users need an internet connection to access. So if the user does not have a good internet connection, it will prevent them from accessing different data or their data being added to the database.
    - Steps taken to reduce the likelihood or impact: Right now we do not have anything in place for handling this problem but we would add error handling for network related issues.
    - Plan for detecting the problem: We would run the app with good and poor internet connection to see the differences between the two. This will help us detect any issues we might have with the error handling we have in place.
    - Mitigation Plan: In case this might occur we need to have a backup plan which is maybe having an offline mode that can enable players to still play the game with other users.

6. How has it changed since submitting the Requirements Document: Our five major risks has become more in depth and we know exactly where we are or will have problems when creating this game. We have done a lot of research on the different components of our game which helps us know where we might encounter different problems and its impact on our product.

12.2 Project Schedule

- External Milestone:
    - Alpha Release (November 15) - Core game play working, with basic UI and features. Limited testing.
    - Beta Release (December 1) - Complete game play and features. UI polish. Stability testing.
    - Final Release (December 12) - Fully tested, stable public release with all planned features.

- Internal Milestone:
    1. Milestone 1: (October 21)
        - Internal
        - Task: Implement core Bluetooth connectivity and device discovery
        - Effort Estimate: 5 Days
        - Dependency: None
    2. Milestone 2: (October 28)
        - Internal
        - Task: Basic UI with device pairing
        - Effort Estimate: 5 days
        - Dependency: Milestone 1
    3. Milestone 3: (November 4)
        - Internal
        - Task: Tagging logic
        - Effort Estimate: 5 days
        - Dependency: Milestone 2
    4. Milestone 4: (November 11)
        - Internal
        - Task: Expand game play features
        - Effort Estimate: 5 days
        - Dependency: Milestone 3

5. Milestone 5: (November 18)
   - Internal
   - Task: Leader boards, notifications, polish UI
   - Effort Estimate: 5 days
   - Dependency: Milestone 4
6. Milestone 6:
   - Internal
   - Task: Extensive testing and bug fixing
   - Effort Estimate: 5 days
   - Dependency: Milestone 5

12.3 Team Structure - Updated

- Our team structure is divided to three categories:

1. Bluetooth: Cameron and Love is working on getting the Bluetooth up and running so devices are able to detect other devices and can enable users to join different game rooms. They are responsible for handling any errors that might come of Bluetooth such as connectivity issues, compatibility issues, etc.
2. Front-End: Cameron and Mahima are responsible for creating the user interface and making the game more user friendly by adding a tag button, history tab, game tab, etc. They also have the role of connecting the database to the history tab which we can use to create the leader board that users can use to track their progress. Mahima is also working on finding more features to add to the game such as dark mode vs light mode, different orientation, etc.
3. Back-End: Greg is responsible for using SQLite to develop a database than can be used to store different player data which is needed to create a leader board. He is responsible for making sure the database is being constantly updated with the current stats of players, have error handling in place, and make sure the database can be connected to the history tab.

12.4 Documentation Plan

1. Our project will contain a user guide that players can use to understand how to play the game. It will explain that they need to connect to other devices using Bluetooth, how they have to be in range of other players, how to access the leader board, and any other relevant information.

2. Our project will have a help menu, if we have time, that users can use if they do not understand how to do something. It will provide step by step instructions on connecting to other devices, how to access their stats, how to join multiple game rooms, etc.

12.5 Test-Automation and CI

- Test-Automation Structure of Choice:
  - We chose JUnit for the test-automation for our TeleTag phone app.
- Reason(s) for Choosing JUnit:
  1. It is a well-integrated structure with Android development ecosystem. It provides tools and APIs for testing Android applications and we can write and run JUnit tests directly into Android Studio, which makes it easier to run tests. There are also different annotations such as '@Test' and '@Before' that makes it easier to define and organize our test methods. It can help us specify which methods are test cases and to set up and tear down resources for your tests.
  2. JUnit supports parameterized tests, which is helpful for testing different scenarios and inputs in Android apps. This allows us to run the same test method with various sets of data, which can be valuable for testing a wide range of conditions in your app.
  3. JUnit tests can be easily integrated into your Continuous Integration and Continuous Deployment (CI/CD) pipeline. This allows us to automate the testing of your Android app with each code change, ensuring that new features or bug fixes don't introduce regressions.
  4. JUnit supports Test-Driven Development, which is a software development methodology that involves writing tests before writing the actual code. This can be beneficial in ensuring that our Bluetooth-related functionality works as expected.
- How to Add a New Test to the Code Base:
  - In order to Add a new test to the code base, navigate to the main/java/Test folder in the source folder. Create a new Kotlin class file and name it to the test you want to create. Create a the class, then the function that contains the test. After writing the test, write @Test above the function. Pushing the changes to GitHub will run the test.
- CI service and How Our Project Repository is linked to it
  - The CI service we are going to use is GitHub Actions.

- How our Repository is Linked:
  - Our CI service, GitHub Actions, is seamlessly linked to our repository via the ".github/workflows" directory, where we store YAML configuration files. These files define the automated workflows for building, testing, and deploying our code. GitHub Actions streamlines our development process by triggering actions in response to various events, such as code pushes and pull requests. This integration ensures the consistent and repeatable execution of CI/CD pipelines, improving code quality and deployment efficiency in our software development workflow.
- Justification for the CI service of Choice:
  1. GitHub Actions is tightly integrated with your GitHub repository. This means our CI/CD workflows are triggered automatically whenever changes are pushed to your repository. This ensures that every code change is built, tested, and deployed, helping you catch issues early in the development process.
  2. GitHub Actions supports running jobs in parallel. This can significantly reduce the overall build and test time, which is particularly important for a Bluetooth-based game app where we need to test various scenarios and device configurations.
  3. GitHub Actions allows us to set up matrix builds to test your app across multiple Android versions, device emulators, and configurations. This is valuable for ensuring broad compatibility, especially in the diverse Android ecosystem.
  4. We can incorporate test suites, JUnit in our case, into our CI workflow which will help us verify the functionality of our Bluetooth functions and features of our app.
- 2 CI Services We Considered
  - Travis
    - Pros:
      1. We can set up matrix builds to test your app across various Android versions, device emulators, and configurations. This is essential for ensuring our game app works well on different devices.
      2. We can define our CI/CD pipeline using a 'travis.yml' configuration file. This provides a high degree of customization, allowing us to tailor the build and deployment process to our project's specific requirements.
      3. Travis CI offers parallel job execution, which can significantly reduce build and test times. This is beneficial for Android projects with large codebases or complex Gradle dependencies like our TeleTag game app.

4. Travis CI allows us to securely store environment variables, which is essential for protecting sensitive data like API keys and credentials especially since our app uses Bluetooth.

* Cons:
    1. The free tier of Travis CI for open-source projects may have concurrency limits, which means we can run a limited number of concurrent jobs. This is a limitation because we have frequent updates.
    2. Using the free tier causes us to have limited resources which affects the build and test times, especially because our app is complex with different classes, functions, etc. On Travis CI's free tier, the resources allocated may not be sufficient to maintain fast build times, leading to slower feedback during development and testing.
    3. Travis CI has some limitations when it comes to artifact storage, especially for large game assets or multi-platform builds. Have to explore other solutions for storing and managing our game's build artifacts.
    4. Some game testing frameworks require additional setup or customization when integrating with Travis CI, especially if they have specific requirements that are not directly supported out of the box. For our app, we might need to configure emulator images and device configurations to make sure that Travis CI can access and interact with the devices we are testing our app on.
    5. Our Bluetooth function may require for us to additional setup to make sure that the Travis CI can interact with Bluetooth devices and simulate Bluetooth interactions during tests.

– CI Circle

* Pros:
    1. CircleCI allows you to create custom Docker containers for our Android game app's CI/CD pipeline. We can include specific Bluetooth-related dependencies, such as Bluetooth testing libraries or emulators, in our custom container.
    2. CircleCI allows parallel job execution, which helps reduce the build and test times significantly. This is crucial for a project like ours because we require an extensive amount of testing.
    3. CircleCI provides artifact caching, allowing you to store and reuse build artifacts. This can significantly improve build and test times by reducing redundant downloads and build steps.

4. CircleCI's support for both Linux and macOS build environments enables you to test your app on multiple platforms, helping you identify and fix platform-specific issues.
5. CircleCI allows you to define highly customizable workflows using a YAML configuration file. This flexibility is beneficial for game apps because it allows you to set up a CI/CD pipeline tailored to your specific needs. You can include steps for building, testing, and deploying your game, including Bluetooth-related testing if needed.

* Cons:
  1. CircleCI provides powerful features, these features may introduce some overhead in terms of management and maintenance, especially for smaller projects.
  2. Our app is a big Android game that requires extensive testing, parallel jobs, custom Docker container configurations, causing the free tier of CircleCI not to be enough and that leads to additional costs to run our tests.
  3. The flexibility of CircleCI may require additional effort to troubleshoot issues or complex configurations. Debugging problems within your CI/CD pipeline may take more time.
  4. CircleCI may experience infrastructure issues or downtime, affecting the reliability of your CI/CD pipeline.
  5. CircleCI doesn't inherently provide built-in versioning and rollback features for build artifacts. We may need to implement our own version control and rollback strategies to handle issues that arise after deployment.
  6. Performance of Android emulators used in CircleCI builds may not fully replicate real-world device performance which can cause limitation when testing resource-intensive games.

- Which Tests Will Be Executed in a CI build:

  1. Kotlin Unit Tests
     - These tests, using JUnit, are conducted to verify the correctness of individual functions and methods within our code.
  2. Integration Tests
     - These integration tests is to be conducted on the Bluetooth functionality. This will test the app's ability to connect, send, and receive data. These tests might use emulators or physical Bluetooth devices depending on the tests we run.
  3. UI Test
     - These tests checks if the user interfaces runs the way it is suppose by testing the screens, buttons, menus, and other UI elements. It also makes sure user inputs are handled correctly such as tapping buttons, making selections from menus, etc.

4. End-to-End Tests
   - These tests go a step further than UI tests and focus on user interactions that span multiple activities or screens.
5. Performance and Load Tests
   - These tests help evaluate how well your app performs under different Bluetooth loads. You can assess connection times, data transfer rates, and the app's responsiveness during Bluetooth operations.

- Development Actions That Trigger a CI Build
  - In order to maintain a rigorous standard of code stability and reliability, our objective is to ensure that the introduction of new features or changes to our codebase does not inadvertently disrupt other parts of our software. To achieve this, our CI/CD pipeline is meticulously crafted to encompass comprehensive testing practices. Specifically, upon each code push to our repository, our system systematically executes a battery of tests located within our designated "Tests" folder. This meticulous testing regime is designed to preemptively identify and rectify any issues arising from code changes, guaranteeing the continued robustness and integrity of our software as it evolves.