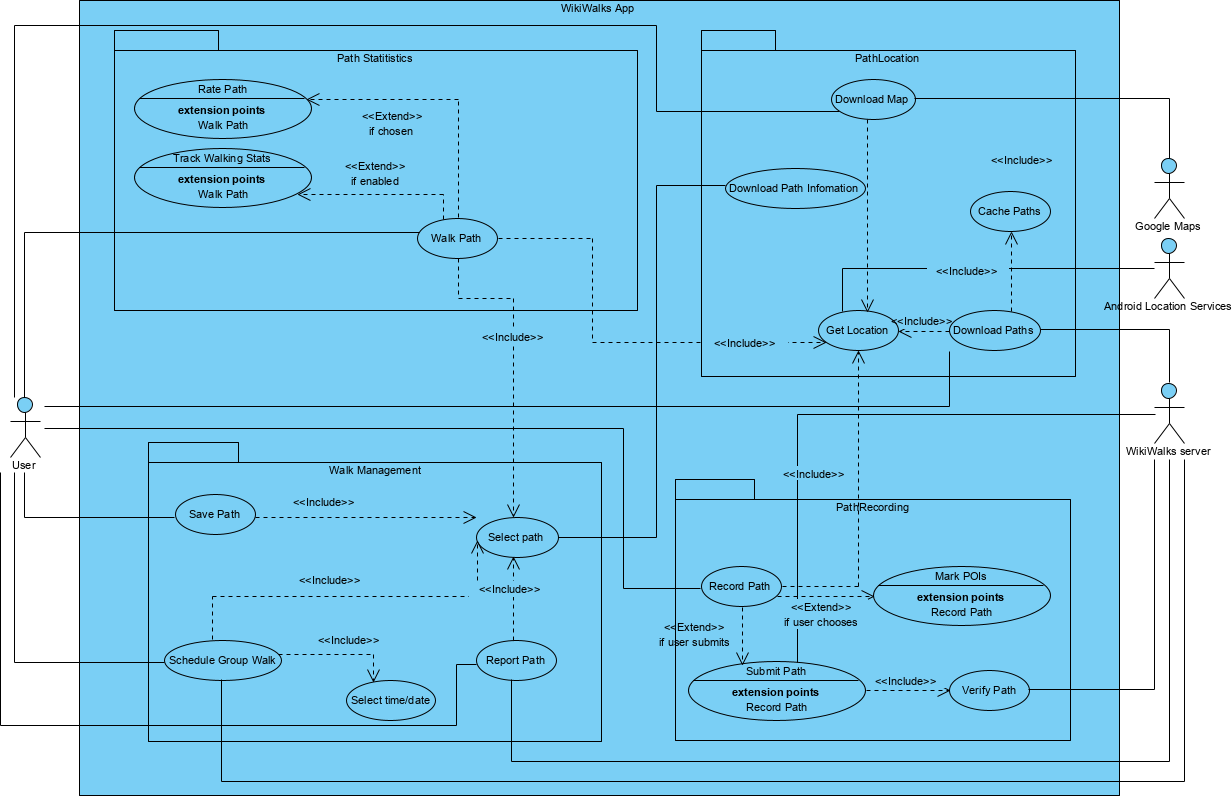
WikiWalks – Requirement Model

# Use Case Model



## User Stories

A user wants to go for a bush walk in their area. They open the WikiWalks app, and the app downloads and displays paths around their location. The user selects a path and follows the line to complete their walk. The user may then choose to rate the path if they’d wish.

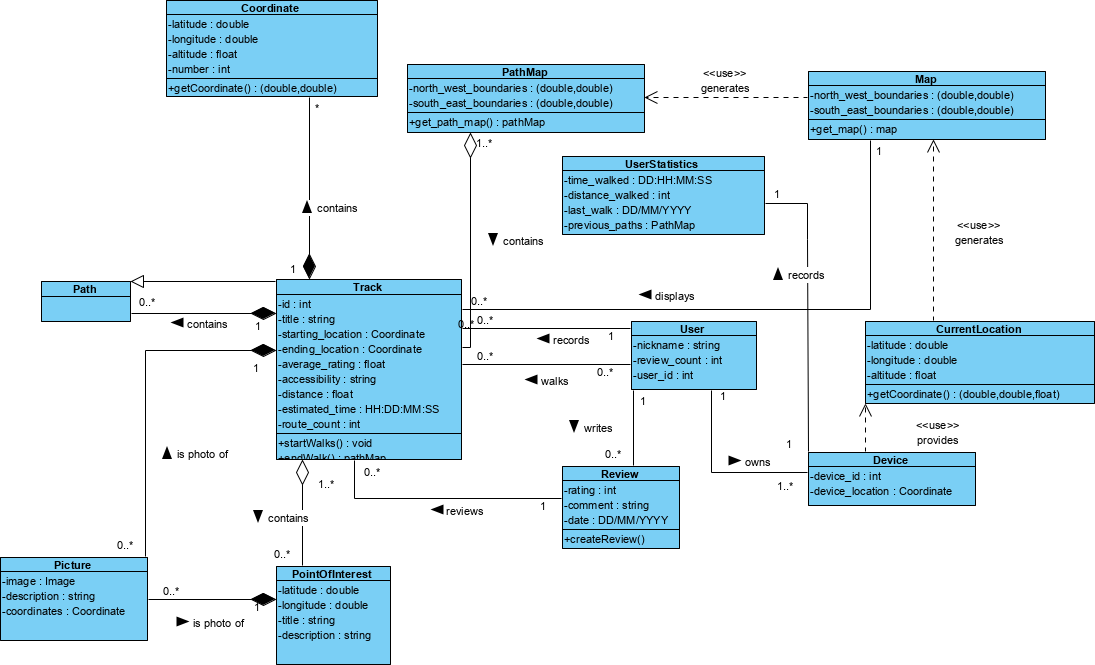
A user has discovered an unmapped path in their area. They open the WikiWalks app and select the option to record a new path. They walk the path, stopping to mark points of interest along the way, and choose to submit the new path to the server. The server verifies it and it becomes available to other users.

A user wants to walk with other users. The user opens the WikiWalks app, and selects the option to schedule a group walk. They select a time and location, before submitting. The server then displays scheduled walks to all other users in the area.

A user wants to save a path for later. They open the app, select a path from the map, and select the option to save. The path will then be bookmarked in the app and can be accessed at a later date.

A user opens the WikiWalks app and sees a marked path but believes there is a problem with it (e.g. doesn’t exist, dangerous, etc.). They select the option to report the path, and the path is displayed as potentially troublesome until it’s manually reviewed or other users vote otherwise.

# Domain Model



# Non-Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Priority | Benefits | Costs |
| Performance – the back-end needs to be quick, and the front-end needs to run smoothly on all devices  App should take less than a second under fast internet to display all paths and information | 1 – slow performance causes a bad first impression | Quicker loading for snappier user experience  Server can take more requests at once without slowdown  Better experience on old or low-end devices | Potentially more expensive server hardware requirements  May require removing or nerfing heavy features to achieve |
| Scalability – the server needs to be able to handle having many paths and many users | 2 – the app could have a potentially endless number of paths submitted from endless users | Less likely to need to make significant upgrades in the future  Easier upgrading  Necessary for best performance | Potentially more expensive server hardware requirements  May require more complex software to handle large amounts of data |
| Backup – in case there is data corruption, backups are important  Database backups should be made at least once a week, keeping the last 10 weeks’ worth | 3 – a failure without backups could corrupt the whole path database, rendering the software useless | Quick restore of the path database and hosting software in case of a failure or corruption  Easier moving between hardware if required | Requires more disk space and takes up system resources, especially as the database gets bigger |
| Ease of use – the app should be simple and easy to navigate | 4 – an easy to understand interface will encourage the user, while a complex confusing one will leave a bad impression | Leaves a good first impression  Users can immediately start using the software and understand all of its features  Untechnical users can use the app with limited trouble | May oversimplify to the point that power-users don’t enjoy it  May limit easily accessible features / increase number of taps  Might be uglier than a more complex, polished UI |
| Reliability – the server should have as little downtime as possible, and the app should retain basic capabilities when there is no server access | 5 – if the server is down, the app is useless, but this is unlikely and thus lower priority | Allows users to use the app at any time without trouble  Offline features are convenient even without server downtime | Potentially more expensive server hardware requirements  May sacrifice features for stability |
| Privacy – the app and server should collect as little information as possible about users | 6 – most users don’t really care, but it’s a nice to have for the segment that do | Requires less data collection and therefore less storage  Widens the potential market of users  Keeps the app less bloated | May require sacrificing features or convenience (e.g. automatic backups and syncing)  May be harder to diagnose bugs without telemetry |

# System-Wide Services

|  |  |
| --- | --- |
| Service | Purpose |
| Getting maps | The system will need to constantly be retrieving maps whenever the viewport changes. |
| Getting paths | The system will need to constantly be retrieving maps whenever the viewport changes. |
| Tracking statistics | If enabled, the system will be tracking the user’s walking statistics locally. |
| Authentication | The system will not be using accounts, but instead device / install IDs that can be backed up and restored from device to device. The system will need to send said ID with each edit or deletion request to ensure there is no vandalism. |

# External Interfaces

|  |  |
| --- | --- |
| Interface | Purpose |
| Android | In order for the app to even install and run, the app must be able to interface with the Android operating system. |
| Google Maps API | The app will be relying on the Google Maps API in order to retrieve map data. |
| Google Play services | The app will pull location data from Google Play services, which provides an API to get the device’s current location, which in turn relies on the device’s sensors. |
| Device’s WiFi and cellular | In order to pull map, location, and path data, an internet connection is required. Due to the nature of the app, cellular and WiFi connections are the most practical, however due to Android handling all this it would theoretically work with ethernet connections as well. |

# Business rules

## Walking Rules

### Walk – Walk1

If the user selects a path and chooses to walk it, then the app should focus and display the path.

### Rate – Walk2

If the user walks a path and chooses to rate it, then the app should allow input of the rating and update the path’s average rating accordingly.

### Report – Walk3

If the user starts walking and something is wrong, then the app should provide a report button to send their concerns.

### Schedule Walk – Walk4

If the user schedules a group walk at a location, then the location should show a marker that there is a group walk scheduled.

## Path Submission Rules

### Record Path – Rec1

If the user wants to add a new path, then the application should record their location until they have completed the path.

### Submit Path – Rec2

If the user chooses to submit a recorded path, then the server should validate that it is a correct path, and add it to the database.

### Delete Path – Rec3

If the user who submitted a path opts to delete the path, then the server should check the device ID and delete the path if it matches.

## Edit Path Rules

### Edit Path – Edit1

If the user who submitted a path wants to update it with better information, then the server should check the device ID and edit the path if it matches.

### Add Picture – Edit2

If a user wants to add a picture to a path or point of interest, then the app should prompt to choose a photo and upload it to the database.

# System Constraints

The project has been limited to 2 programming languages – Python using Flask for the back-end, and Java developed in Android Studio for the front-end. These were chosen due to the team’s experience with each, and must be strictly stuck to in order to avoid confusion.

The app is limited (at least while in development) to the free tier of Google Maps’ API. This should be sufficient for most requests, however other workarounds should be found if a paid version is required for something.

The app must support all Android versions from at least KitKat and upwards, ideally back to Jellybean. This is to ensure maximum compatibility with devices.

The design for the interface should follow general material design guidelines for a consistent, familiar, and easy to understand UI.