SpotMe Mobile App
System-Requirements Documentation

FUNCTIONAL REQUIREMENTS

Feature 1: Primary UI/UX Elements		**Priority: High**
Description: 1.1 The main display of the app a user will see upon opening the app shall consist of a full-screen map view served through Google Maps tiles. ● 1.1.1 User interaction with the Maps elements as well as map-specific overlays, will be described in more detail in **Feature 2.** 1.2 A toolbar aligned to the top-right side of the user's display shall present the minimal options for app interaction in the form of vertically stacked, small, circular UI widgets: ● 1.2.1 The first (topmost) button shall have an icon of a compass that rotates based on the compass rotation of the map. Tapping this button shall realign the map view to have it facing north towards the top of the user's display. ● 1.2.2 The second button shall have an icon with a simple car symbol and will be used to save or return a user's parked car location. Tapping this button will perform an action detailed in **Feature 4**. ○ 1.2.2.1 If the user has not yet allowed app access to their location, the app shall alert them with a small menu that prompts them to give access through their device's settings. ● 1.2.3 The third button shall have a simple "person" icon. Tapping this button will bring users to the accounts screen detailed in **Feature 5**. ● 1.2.4 The fourth (bottommost) button shall have a simple arrow in a circle icon. Tapping this button will refresh the user's client with up to date server information regarding occupancy (see section 3.3)		
Inputs: ● Physical user interactions (button tapping) ● Location access permission	**Outputs:** ● Fullscreen Google Maps view of campus. ● [Pressing 1.2.1] Maps view is re-oriented to have North facing the top of the user's display. ● [Pressing 1.2.2] *see **Feature 4***; location access permission alert if not yet allowed by user. ● [Pressing 1.2.3] *see **Feature 5***.	**Processes:** 1. User opens the app. 2. Fullscreen map view of campus is displayed. Toolbar widgets appear in the top-right of the user's display. 3. (Map interactions, see **Feature 2**) 4. User taps a toolbar widget. An action described in the **Outputs** section takes place depending on which widget is pressed. 5. User closes the app.
Dependencies: Google Maps Navigation SDK, device location tools, device permissions access, **Feature 2**		

Feature 2: Map Presentation and Interaction	**Priority: High**
Description: 2.1 The app shall display a full screen view through Google Maps that is initially centralized on CSUF's campus. ● 2.1.1 Users shall have the ability to zoom in or out of the map via two-finger pinching and expanding	

motions.

- 2.1.2 Users shall have the ability to rotate the view using two fingers in a twisting motion.
- 2.1.3 Tapping the compass button (1.2.1) shall reset the map orientation such that north points to the top of the user's display.

2.2 The app shall overlay color-coded icons in the shape of a pin over distinct parking lots to quickly inform users of the current congestion status of a parking lot (more detail in 2.2.2)

- 2.2.1 A pin marker over a parking lot shall have a circular icon with text that indicates a specific lot's name.
- 2.2.2 The color coding is as follows: red shall suggest intense congestion, yellow shall suggest medium congestion, and green shall suggest mild congestion. Calculation of congestion will be detailed in **Feature 3**.
- 2.2.3 Tapping the marker shall expand the pin into a small informational menu element with more details such as permit requirements, current occupancy as a fraction of taken spots over total spots, and a histogram of typical occupancy (more detail in 3.4).
    - 2.2.3.1 The histogram figure shall, by default, show the average congestion level by hour of the current day of the week. Angle bracket icons to the left and right of the figure shall allow users to switch between days of the week.

2.3 The app shall draw an overlay of the individual parking spots within each lot consisting of simple rectangular bars atop each spot depending on the user's zoom level on the map.

- 2.3.1 Each spot shall be color coded according to their status: green indicates unoccupied, yellow indicates soft reserved, and red indicates taken. Data for each spot shall be provided via sensors.
- 2.3.2 This overlay shall only be drawn if the user's current zoom level is within 3 levels of zoom from maximum and only for the bars that will be visible on the user's display. Note: A "level of zoom" is defined as the approximate level of a user's view magnification where 0 is a view of the entire world and 19 is the maximum level of zoom for the CSU Fresno locale.
- 2.3.3 Tapping a green (open) bar shall allow users to soft reserve the spot. Further detail shall be provided in **Feature 7**.

Inputs:	**Outputs:**	**Processes:**
• Google Maps Tiles • Physical user interactions • Congestion levels via **Feature 3** • Per-lot permit requirement information • User's current zoom level • Current day of the week	• Fullscreen display of Google Maps view initially centered on campus • Zoom and rotation of the Google Maps view • Pins providing quick reference for each lot's congestion level and additional information when tapped • Per-spot box overlay that displays current state (occupied, reserved, free)	1. User is presented map. 2. User uses two-finger motions or taps the compass button. 3. Map display adjusts accordingly. 4. User quickly identifies the approximate congestion of each lot at a glance with the pins. 5. User taps a pin. 6. Pin expands into menu with detailed statistics (2.2.3) 7. User zooms within 3 levels of maximum. 8. Per-spot bar overlay is shown with color code indicating status. 9. User taps a green spot. 10. User can soft reserve that spot (see **Feature 7**).

Dependencies: Google Maps SDK, network access, **Feature 3**

Feature 3: Sensor Data Ingest and Congestion Calculation	**Priority: High**

Description:

3.1 The congestion level shall be calculated based on real-time occupancy data from sensors installed in each individual spot.

- 3.1.1 The congestion level as a percentage will be calculated according to the number of occupied spots over the total number of spots in the lot for use in **3.2**.
- 3.1.2 Data from the sensors shall be transported to the back-end server through the use of an intermediary network gateway supplying HTTP POST requests that include relevant information such as the sensor's ID and its occupancy status.
    - 3.1.2.1 NOTE: As real sensors are an eventuality and are not yet physically available, this data shall merely be simulated for the purpose of furnishing a proof-of-concept. An auxiliary testing system that emulates the gateway making these requests as sensor data is received shall be integrated.

3.2 Users shall be presented with a congestion display indicating the current level of traffic within the parking lot in the form of overlay pins as previously detailed in **Feature 2**.

- 3.2.1 The congestion level shall be visualized using a color-coded system:
    - Green for low congestion (below 65% capacity)
    - Yellow for moderate congestion (65-85% estimated occupancy)
    - Red for high congestion (85-100% estimated occupancy)

3.3 The app shall update congestion data in real-time as it is received from the parking lot sensors.

- 3.3.1 Server-side information shall be updated with every event that can alter occupancy data: a sensor detects that a spot is now occupied/unoccupied, or a user soft reserves a spot.
- 3.3.2 A snapshot of the server-side information will be served to a user's app client when they first open the app, when they tap the refresh button (1.2.4), or soft reserve a spot through **Feature 7**.

3.4 Congestion level history shall be maintained for a maximum of 4 weeks for the purpose of generating a "typical congestion" dataset.

- 3.4.1 Snapshots of congestion levels shall be taken at every X:00 and X:30 time interval.
- 3.4.2 A weekly averages figure shall be calculated for every 30 minute time block. For example, the average congestion at 6:30AM-7:00AM on Mondays for lot X would be the average of the congestion levels snapshot at 6:30AM for lot X over the past 4 Mondays.
- 3.4.3 This information shall be stored server-side for the purpose of generating typical congestion over time histograms for 2.2.3, and for recommending alternate options for users desiring busy lots in **Feature 6**.

Inputs:	**Outputs:**	**Processes:**
• Real-time data of occupied and available parking spots via sensors • Events that alter the current state of parking lot occupancy	• Congestion calculation as a percentage for each lot • Up-to-date information regarding total parking lot occupancy	1. An event causes a parking spot's state to change. 2. The number of (potentially) occupied spots is updated for that lot. 3. The congestion percentage is recalculated. 4. A user opens the app or hits the refresh button. 5. The current congestion percentage, occupancy fraction, and physical spot occupancy data is served to the user's client and updates their markers accordingly.

Dependencies: Sensor data, network access

Feature 4: Saving Parked Location	**Priority: High**

Description:
4.1 Users shall have the ability to save the location of their vehicle's parked location through the UI widget described in 1.2.2.

- 4.1.1 Tapping the car button shall prompt the user to confirm that they wish to save their current location. If the user confirms, their current location will be saved to local space on their device (see **Feature 8** regarding security consideration). A new persistent map marker will be added to the user's map view atop this saved location.
- 4.1.2 If the user already has a location saved with this feature, they will be prompted to either select a "locate" option which centers the map view to the saved location of their vehicle, an "update" option to change the saved location (if the user forgets to remove it with the next option), and a "forget" option that deletes the saved location from their device. Both latter options will force the user to confirm their choice.

Inputs:	**Outputs:**	**Processes:**
● Physical user interactions ● Fine-grain device location data	● A persistent map icon marking the location of their vehicle. ● Alteration of this marker's location if the user selects the "update" option. ● Erasure of this marker if the user selects the "forget" option.	1. User taps the car button in the main map toolbar. 2. User confirms that they wish to save their location. 3. Their current position is saved locally and an icon is added to the map over this location. 4. [repeat, Step 1] User selects the update option. 5. The location changes to their new current position and the icon moves accordingly. 6. [repeat, Step 1] User selects forget option. 7. The saved location data as well as the map icon is deleted.

Dependencies: Fine-grained location data access

Feature 5: Accounts Menu UI/UX	**Priority: Medium**

Description:
5.1 The Accounts Menu shall be accessible from the main map screen through the profile toolbar button described in **Feature 1**.

- 5.1.1 This will be presented as a separate page from the main map screen showing a list of the options available to the user.
    - 5.1.1.1 If the user is not logged in, the menu shall only present options to sign in to an existing account or register a new account (registration process described in 5.2)

        - 5.1.1.2 When the user successfully signs in, the app shall query and locally cache the user's account information from the server database (permits, weekly schedule). The user shall remain signed in to the service until they log out.
    - 5.1.2 If the user is signed in, a greeting with their username will be displayed at the top of the page.

5.2 Users shall be able to register an account using a simple username-password credential schema.

- 5.2.1 The username shall be used as the unique identifier for each user.
- 5.2.2 Multiple users can not share the same username. Attempting to register with an existing username will alert the user that they must change their desired username.
- 5.2.3 Passwords shall minimally be 8 characters long, contain at least one number, one capital letter, and one non-alphanumeric character for the purpose of security.

5.3 Signed in users shall have "Update Account" as the 1st (topmost) option in the accounts page to modify their account.

- 5.3.1 Users can change their username through this menu with a simple textbox input. The app will query the server to ensure that the new username is not in use before changing it.
- 5.3.2 Users can change their password through this menu with a simple textbox input. They will only be allowed to confirm their choice if their new password fulfills 5.2.3.
- 5.3.3 In either case, the user will be prompted to enter their current password before a change to their account credentials is allowed.

5.4 Signed in users shall have "Your Permits" as the 2nd option in the accounts page to specify their permit privileges.

- 5.4.1 Green, Yellow, Black, and Handicap permit options will be presented in a checklist that a user simply fills according to which permits are applicable to them; this information is saved locally and synced to the database when they leave this menu. By default, a user will have NO registered permits.

5.5 Signed in users shall have a "Weekly Schedule" as the 3rd option in the accounts page. See **Feature 6** for full detail of operation.

- 5.5.1 A user's weekly schedule data will be synced to the database when they leave this menu for the purpose of syncing across devices.

5.6 Signed in users shall have "Log Out" as the 4th option in the accounts page. This will simply log the user out of their current session and delete their account information from local storage.

5.7 Signed in users shall have "Delete Account" as the 5th (bottom) option in the accounts page.

- 5.7.1 This option will be displayed at the very bottom of the user's display in the accounts menu as to minimize incidental selection when intending to choose the other options.
- 5.7.2 Users will be alerted that they are attempting to delete their account. They will be notified that this includes deleting their weekly schedule and permit information. A "confirm" option will be presented at the bottom of this alert.
- 5.7.3 If the user confirms deletion, they are signed out, and their account information is deleted from database records.

Inputs:	**Outputs:**	**Processes:**
● Physical user interactions ● "Logged in" status ● User login credentials (username and password) ● User's permits	● Available account options displayed as a list. ● Ability to create an account with a username and password. ● Access to permits and weekly schedule editor. ● The ability to log out. ● The ability to delete one's account.	1. User taps the accounts icon, which navigates to the Accounts Menu. 2. If the user is not logged in, the app displays login or registration options. 3. User creates account or logs in to an existing one. 4. The app retrieves and caches the user's profile information locally.

		5. The user updates their permits and/or weekly schedule which is synced to the backend database. 6. The user taps the log out option and confirms. 7. The user is logged out and the app returns to a "guest" mode. 8. [Alternately] User taps the delete button and confirms. 9. They are logged out and their account information is deleted locally and serverside.
Dependencies: MongoDB, network connection		

Feature 6: Weekly Parking Schedule UI/UX	**Priority: Medium**

Description:

6.1 Users signed into an account shall have access to a weekly parking scheduler that allows them to input their anticipated parking schedule over the course of each week through a table.

- 6.1.1 The table shall consist of rows that correspond to 30-minute time blocks beginning and ending at midnight of each day. The columns shall correspond to days of the week from Sunday to Saturday.
- 6.1.2 User defined time blocks shall be marked with the user's desired lot for that time block and a color code schema defined in 6.4.

6.2 Users shall be able to create time blocks by long pressing an empty cell and continuing to swipe to select adjacent empty cells.

- 6.2.1 Attempting to swipe over an already occupied time cell will have no effect; the selection will simply stop expanding. Users must manually resize custom time blocks via 6.3.
- 6.2.2 Newly created blocks will have no desired lot by default. This must be defined by the user through the method detailed in 6.3.1

6.3 Users shall be able to modify or delete time blocks by tapping or long pressing them.

- 6.3.1 Tapping a time block will prompt a user with the options to update the desired parking space for that time block from a list of parking lots, or to cancel this action.
    - 6.3.1.1 The options may have yellow or red warning symbols next to them indicating high typical occupancy or critical issues, respectively (see 6.4). Users shall still be able to select these options, however, as either typical congestion lowers or the user updates their permits (see 6.5).
- 6.3.2 Long pressing a time block will prompt a user with the options to resize the block or delete it outright.
    - 6.3.2.1 Selecting resize will highlight the block and add interactable widgets to the top and bottom of the cell that can be dragged to expand/compress the block in either direction.
    - 6.3.2.2 Selecting delete will prompt to confirm. If the user confirms, the cells will be freed.

6.4 After creating a block and specifying a parking lot, the block shall be colored according to factors such as congestion levels and permit requirements.

- 6.4.1 Green indicates that the lot typically experiences an average of <85% occupancy at the *starting time* of a block (see 3.4). The app shall only consider the starting time of a block because it can be assumed a user will not move their vehicle for the entirety of the block once they have found a spot.

- 6.4.2 Yellow indicates that the lot typically experiences >85% occupancy at the starting time of a block.
- 6.4.3 Red indicates a critical consideration that could prevent the user from parking in a lot outright. Early development will only factor in permits that the user does not have listed on their account that are required for a desired lot for a certain time frame.

6.5 The app shall change the color coding of each block depending on up-to-date factors.

- 6.5.1 The first time a user opens their weekly scheduler table in a 7-day interval between 12:00AM Sunday and Saturday 11:59PM, the app shall make a query to get the current typical occupancy data from the server for the starting time of the occupied time blocks. The color of each block switches between green and yellow if either typical congestion is below or above 85%, respectively.
    - 6.5.1.1 After the first query of the week, the traffic occupancy data is cached or updated. This information is overwritten with each subsequent initial query of the week.
- 6.5.2 Each time the user opens their weekly scheduler table, the app shall query its locally saved information regarding a user's permits. If previously red blocks can be validated with any new permits found by this query, the block switches to green or yellow depending on congestion (see previous).

Inputs:	**Outputs:**	**Processes:**
● Physical user interactions ● Typical occupancy data calculated via 3.4 ● User's permit information ● Current date and time	● User creates and saves a tabulated weekly-basis parking schedule ● User can check their table for changes in expected congestion and can alter their desired lots accordingly ● User is notified about permit restrictions that will require alternate options or updating their permits in their account	1. User selects the weekly scheduler option in the Accounts menu. 2. User creates time blocks with desired lots as specified in 6.2 and 6.3 3. User is advised of considerations such as congestion and permit requirements. 4. User modifies their table accordingly, and/or updates their permits. 5. User reopens menu down the line. 6. App queries updated congestion and permit information (if first time in a week); adjusts table accordingly 7. Repeat 3-6.

Dependencies: Feature 3, network access

Feature 7: Soft Vacancy Reservation and Navigation	**Priority: Low**

Description:

7.1 The soft reservation feature shall allow users to temporarily "reserve" and receive navigation directions to a currently vacant parking spot by tapping a green block presented via 2.3.

- 7.1.1 Tapping a green spot will prompt the user with a button to "mark for occupation". Tapping this button will alert the user that the reservation period will only last 10 minutes before prompting confirmation. Upon confirmation, the app shall inform the server that the spot is soft reserved (7.1.3) and the map display is switched to navigation mode via the Google Maps Navigation SDK.

- 7.1.1.1 If multiple users attempt to soft reserve a spot simultaneously, priority will be given to the user whose request is received first. The other user(s) will be notified with an error that the spot is no longer unreserved and will have to locate another.
- 7.1.1.2 If the user does not specify that they have a permit required by the desired spot or is signed out, a warning in red advising them of the permit requirement will be displayed below the button. They will still be able to reserve the spot as this will allow anonymous users with required permits or users who have yet to add their permits to still utilize the feature.

- 7.1.2 Users will be able to reserve a spot through the app for a maximum of 10 minutes while they navigate to the lot. Once the reservation expires, the spot shall automatically become available again if the sensor for that spot does not sense a vehicle within the time window.
- 7.1.3 A reserved spot will be marked as yellow in the map overlay to other users during the reservation window. They will not be able to soft reserve this spot in the app.
  - 7.1.3.1 Note: this does not guarantee the spot for the user, as any vehicle can occupy the spot if they physically arrive before the user. Handling of this scenario will be detailed in 7.2.

7.2 Sensor status will be polled every 10 seconds during the reservation period. If the sensor detects that a vehicle has occupied the soft reserved spot, the app will notify the user and ask if it was them who parked; the options “yes” and “no” will be provided in the notification.

- 7.2.1 If the user selects “yes”, then navigation is terminated. The user is then prompted with the option to save their vehicle’s location with the same process detailed in **Feature 4**.
- 7.2.2 If the user selects no, then the app queries, soft reserves for the remaining reservation period, and redirects to the closest available spot factoring in distance by coordinates. Spots requiring permits the user does not have registered will be ignored in this calculation.
  - 7.2.2.1 This is repeated every time the soft reserved spot is taken until either the user exits navigation manually or their reservation period expires.

Inputs:	**Outputs:**	**Processes:**
● Updated vacancy data ● Spot location by coordinates ● Navigation data provided by Google Maps ● Device location permissions	● User soft reserves a spot, barring other users from reserving that spot ● User receives navigation to reserved spot, or is navigated to other vacant spots ● User successfully finds a parking in or near the desired spot	1. User confirms reservation for a vacant green spot. If reservation fails, the user is notified to try another spot. 2. App notifies the server of the reservation; any later requests to reserve the spot are ignored. 3. User receives navigation to the spot. 4. If the spot becomes occupied, an alternate spot is found and the user is redirected. 5. The user parks in a vacant spot and terminates soft reservation.

Dependencies: Device location permission, Google Maps Navigation SDK, **Feature 4**

END OF FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENTS

1. Quality Attributes

Feature 8: User Privacy and Security	**Priority: High**

Description:
8.1 The system shall guarantee the user data privacy and security by limiting what information is necessary for operation and minimizing personal, trackable data from reaching the back-end server.

- 8.1.1 No personally identifying user data shall be required for core functionality. Users are free to utilize the features that do not require an account, and the account-required features optionally store minimally sensitive data such as a user's permits and (largely tentative) weekly parking schedule for the purpose of cross-device syncing.
- 8.1.2 Fine-grained location data, such as that accessed via **Feature 4**, shall never be stored in the database; external access to device location should be limited to the Google Navigation SDK.
- 8.1.3 Server-side user data shall be completely removable by a user by deleting their account. No leftover information of a user's account should be present after a user selects this option.

Feature 9: Account Security	**Priority: High**

Description:
9.1 The system shall ensure account security through password protection requirements and limitations on what server-side information is accessible to the user and how.

- 9.1.1 Account access shall be restricted by strong password requirements as detailed in 5.2.2. Alongside server-side security implementation (9.2.1) This should prevent easy guessing of passwords via brute force.

9.2 User passwords shall never be stored as plaintext.

- 9.2.1 Passwords shall be used to generate a hash string for long-term storage server side.

9.3 Users shall never have direct access to information stored by the server-side database.

- 9.3.1 All access to information by a user shall be handled through requests made to the back-end proxy server via the user's app client.
- 9.3.2 Only the server itself will have the direct ability to access and manipulate user data within the database. Users shall not have the ability to query information directly from the database.

2. System Performance

Feature 10: General UI/UX Responsiveness	**Priority: Medium**

Description:
10.1 In expected conditions (detailed in **Feature 13**) the system should provide a responsive UI that minimizes the amount of time spent waiting for primary display elements (**Feature 1**) to load in and become usable.

- 10.1.1, the amount of time a user spends waiting for the primary map display to render upon opening the app should not exceed 10 seconds.
- 10.1.2 The amount of time spent waiting for the UI toolbar to render and become usable upon opening the app should not exceed 10 seconds (shared with map rendering).
- 10.1.3 Overlay pins will be dependent on occupancy query speed (**Feature 11**) but should not take longer than 10 seconds to render once the information has been received.

- 10.1.4 Overlay box markers for per-spot querying will be dependent on occupancy query speed (**Feature 11**) but should not take longer than 10 seconds to render the "visible" boxes once the information has been received.

10.2 The system, within hardware considerations laid out in **Feature 13**, should provide a consistently smooth display experience.

- 10.2.1 The app should display map manipulation (rotating, panning, zooming) and animations at a minimum of 60 frames per second, but otherwise unlimited except by the refresh rate of the user's display.
- 10.2.2 Considerations such as limiting the complexity of UI icons and the number of dynamic elements shall be made to achieve this target; for example, overlay elements such as pins and spot markers will only be rendered if they are visible to the user's display.

Feature 11: Data Query Efficiency	**Priority: Medium**

Description:

11.1 Under the expected conditions laid out in **Feature 13**, the system should be able to retrieve server-stored information required for core functionality in a timely manner.

- 11.1.1 Real-time metrics such as occupancy data shall only be served on a limited basis and only via specified events (opening the app, tapping the refresh button in 1.2.4). This data should take no longer than 10 seconds from the request to be fully retrieved.

11.2 In cases where real-time data is unavailable, the app shall display cached data from the last successful query.

- 11.2.1 A copy of the most up-to-date query results shall be stored locally in a cache and overwritten upon each successive query to the server. This shall be used for circumstances where server access may not be available in the future.

3. Constraints

Feature 12: API Query Limitation	**Priority: Medium**

Description:

12.1 The system should not exceed a target number of queries for each third party API service in use: the Google Maps SDK for map tiles and custom overlays present in **Features 1** and **2,** and the Navigation SDK for use in **Feature 7**. The limits will be defined in terms of total API requests to each service in a one-month time frame.

- 12.1.1 The target limit for the Maps SDK shall be 100,000 request units as this falls under the "no charge" bracket of the Maps SDK API request pricing.
    - 12.1.1.1 The Google Maps SDK under the Mobile Native Dynamic Maps SKU defines a "map load" request unit as a single Google Maps object. These are typically instantiated upon initial rendering of a Google Maps instance and actions that introduce new areas to the view such as panning through different areas or zooming out.
- 12.1.2 The target limit for the Navigation SDK shall be 1,000 request units as this falls under the "no charge" bracket of the Navigation SDK API request pricing.
    - 12.1.2.1 The Google Navigation SDK defines a navigation request unit as a single destination request. In **Feature 7**, this would correspond to each new spot the user is directed or redirected to.

Feature 13: Server and Client Hardware Considerations	**Priority: Low**

Description:

13.1 The system's user app should be optimized to the point where client-side operations should complete with expected performance metrics laid out in **Features 10** and **11** in typical network conditions and on entry level modern hardware or previous generation flagship devices.

- 13.1.1 The system should be able to achieve the UX performance metrics laid out in **Feature 10** on cellular devices that are still actively supported by their manufacturers. For example, this would include the Samsung Galaxy A01 which contains an octa-core processor with 3GB of RAM and the iPhone XR which contains a hexa-core processor and 3GB of memory which were released 2020 and 2018, respectively.
- 13.1.2 It shall be expected that users will minimally have access to a stable 4G LTE connection or better supporting a minimum 5Mbps connection in a worst case scenario. The app should achieve the target performance metrics in **Feature 11** while operating within this limit.

13.2 The system's backend server software should provide services necessary for core functionality (namely, **Feature 3**) while achieving query efficiency targets laid out in **Feature 11**.

- 13.2.1 Initial development shall test performance on a Google Cloud Compute Engine utilizing free-tier specification: an e2-micro configuration with 2 vCPU processors on a shared physical core, up to 1GB of total RAM, and up to 30GB of persistent storage.
- 13.2.2 If it is determined that the system's backend can not keep up with the amount of processing required to achieve desired performance metrics, optimizations to the software should be prioritized first. Upgrading the compute engine to a higher-spec configuration should only be considered if all practical optimization efforts have been exhausted as this will incur additional costs associated with the Google Cloud hosting Service.