Detailed Design Documentation

INTRODUCTION

- introduction to the entire document

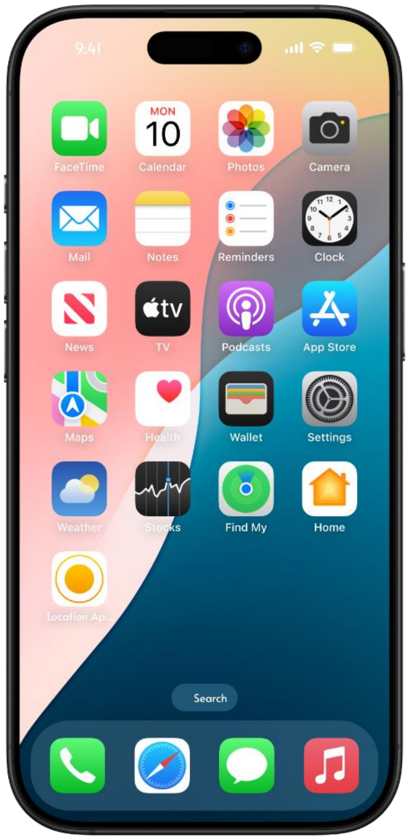
This document provides a detailed design specification for a navigation application that supports real-time tracking, notifications, and location-based monitoring to enhance user navigation and engagement. It serves as a comprehensive guide for developers and stakeholders, detailing the architectural design, rationale, and requirements mappings that ensure the application meets both functional and non-functional needs.

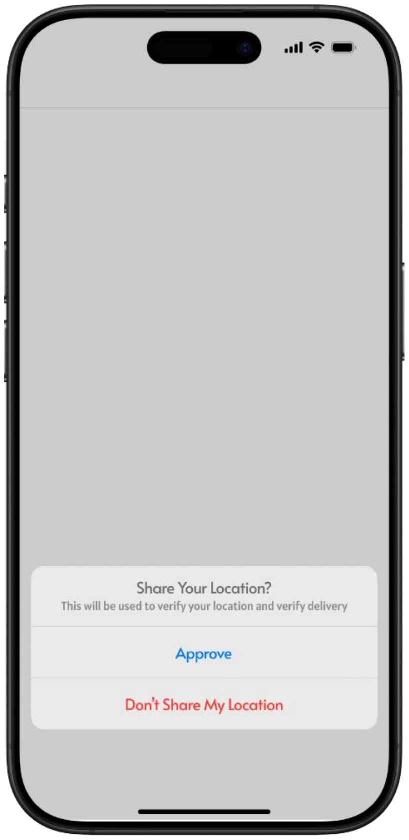
- purpose and scope of the document

The purpose of this document is to outline the structural and behavioral components of the navigation app, demonstrating how each part contributes to the app's functionality and user experience. It covers key technical aspects, design justifications, and system requirements, aiming to create a cohesive resource that informs the development process. This document also emphasizes the app's scalability, reliability, and compliance with relevant engineering standards, setting a foundation for successful implementation and future enhancement.

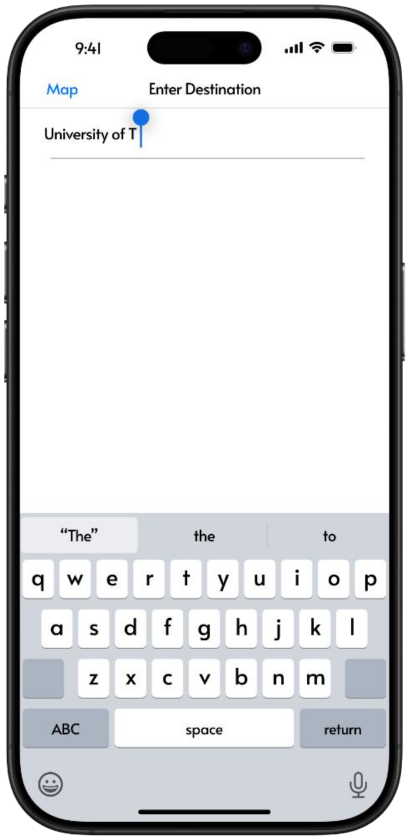
- description of the structure of the document
    - **Graphical User Interface (GUI) Design** – Provides screen designs illustrating the visual layout and user interface of the app.
    - **Static Model** – Contains class diagrams representing the static structure of the application, including core classes and their relationships.
    - **Dynamic Model** – Includes sequence diagrams that show component interactions throughout different phases of user engagement.
    - **Rationale for the Detailed Design Model** – Discusses the role and functionality of each class, with an emphasis on how they support the system's overall performance and responsiveness.
    - **Traceability from Requirements to Detailed Design Model** – Maps functional and non-functional requirements to specific design elements, demonstrating how the app meets its intended objectives.
    - **Configuration Management** – Details procedures and evidence for placing the design model under configuration management, ensuring version control and consistency.
    - **Engineering Standards and Multiple Constraints** – Covers relevant standards (e.g., IEEE SID 1016-1998) and the constraints influencing the app's design.
    - **Additional References** – Lists supporting academic and technical references that reinforce the design methodologies and principles applied in the document.

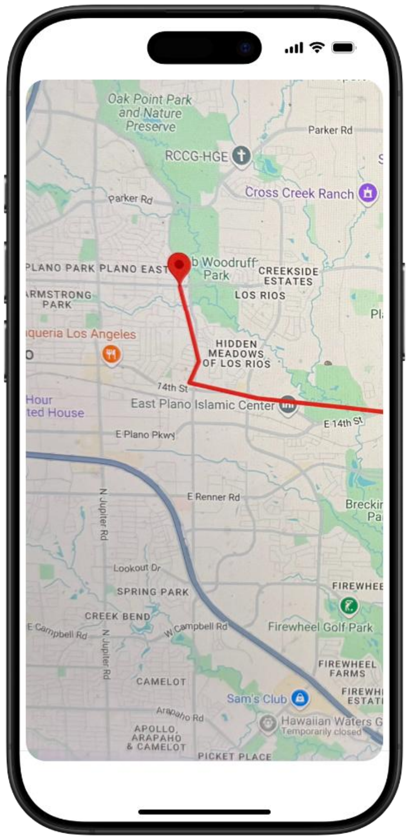
GUI (Graphical User Interface) Design

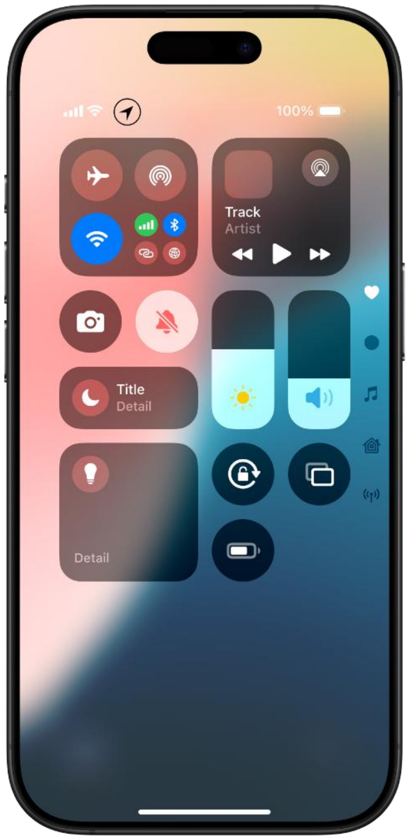


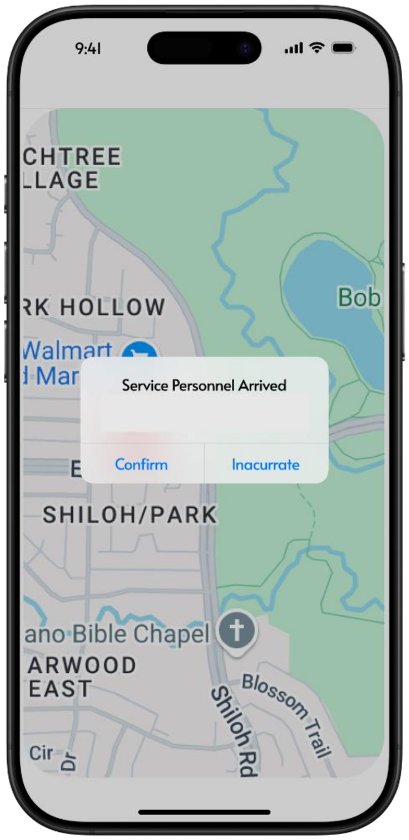

Share Your Location?

This will be used to verify your location and verify delivery



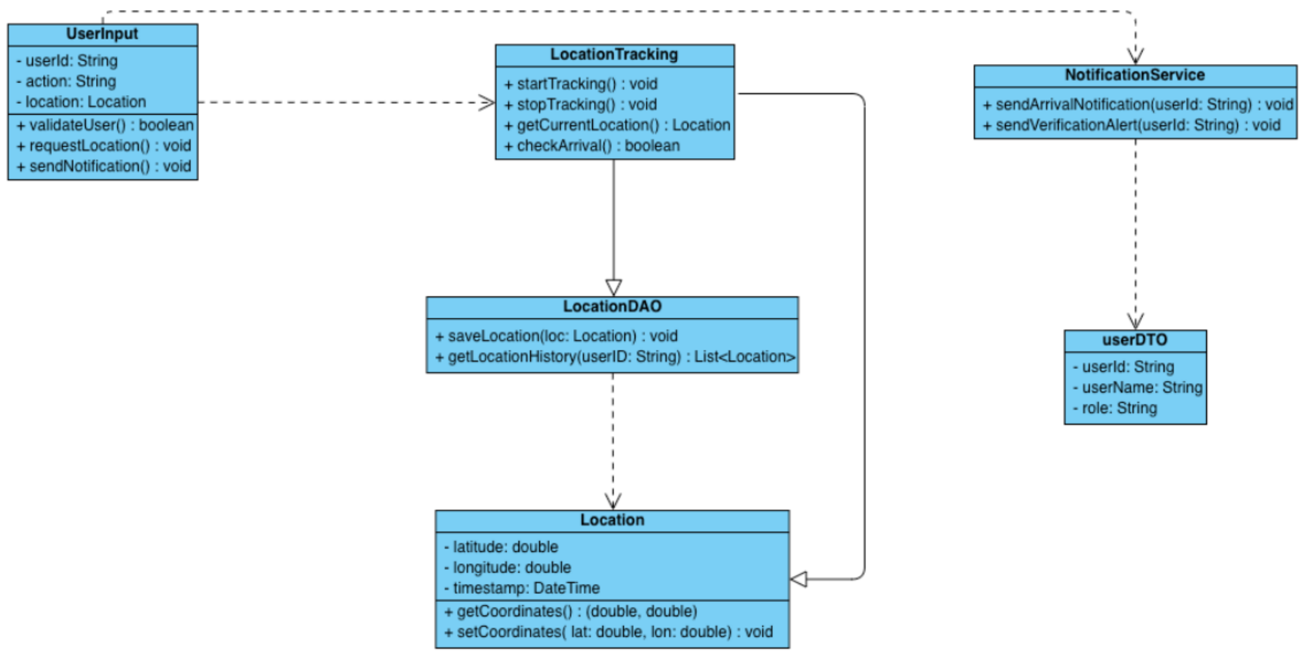





Service Personnel Arrived

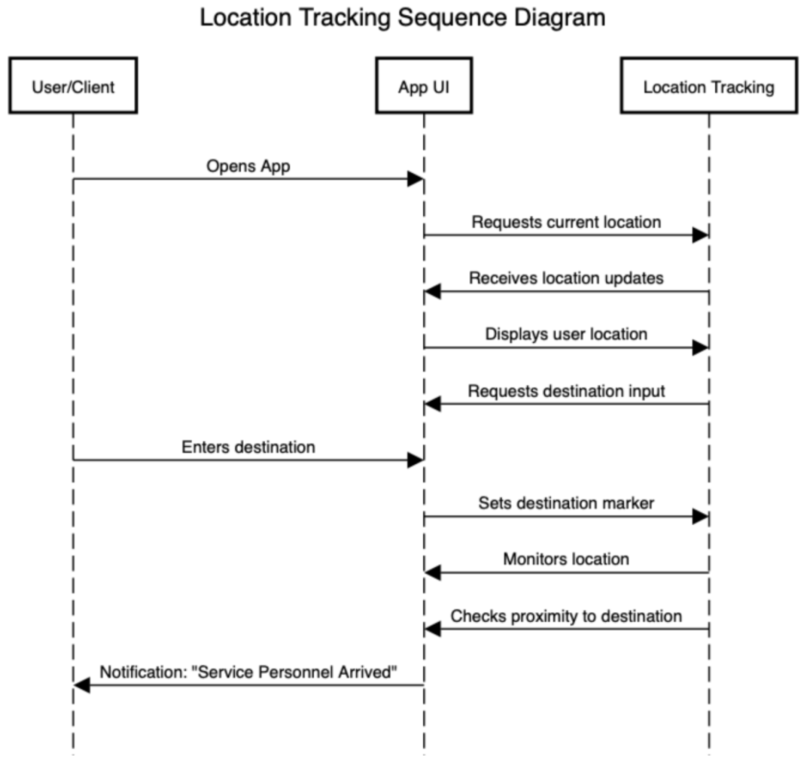
STATIC MODEL

CLASS DIAGRAMS


UserInput
- userId: String
- action: String
- location: Location
+ validateUser() : boolean
+ requestLocation() : void
+ sendNotification() : void
LocationTracking
+ startTracking() : void
+ stopTracking() : void
+ getCurrentLocation() : Location
+ checkArrival() : boolean
NotificationService
+ sendArrivalNotification(userId: String) : void
+ sendVerificationAlert(userId: String) : void
LocationDAO
+ saveLocation(loc: Location) : void
+ getLocationHistory(userID: String) : List<Location>
userDTO
- userId: String
- userName: String
- role: String
Location
- latitude: double
- longitude: double
- timestamp: DateTime
+ getCoordinates() : (double, double)
+ setCoordinates(lat: double, lon: double) : void

DYNAMIC MODEL

SEQUENCE DIAGRAMS


Location Tracking Sequence Diagram
User/Client
App UI
Location Tracking
Opens App
Requests current location
Receives location updates
Displays user location
Requests destination input
Enters destination
Sets destination marker
Monitors location
Checks proximity to destination
Notification: "Service Personnel Arrived"

RATIONALE FOR YOUR DETAILED DESIGN MODEL

Static Model

- The UserInput class centralizes user-related actions, ensuring that every action taken by the user is validated, locations are requested accurately, and the appropriate system responses are initiated.

- The LocationTracking class is critical for managing the actual location tracking. It encapsulates the logic for starting, stopping, and fetching the current location, which ensures accurate and real-time tracking capabilities.

- The NotificationService class ensures that all necessary notifications are sent at the right time. It enhances user experience by providing real-time updates and alerts, which contribute to the overall functionality and reliability of the app.

- The LocationDAO class handles the persistence of location data which ensures that all location data is stored securely and can be retrieved when needed. It allows for historical tracking and analysis, which is crucial for features like route optimization and service verification.

- The Location class represents a single point of location data, encapsulating the essential details required for tracking such as capturing precise coordinates and timestamps. It standardizes how location data is handled across the app and forms the basis of the tracking system.

- The UserDTO class ensures that user information is transported and used consistently across different parts of the application, promoting modularity and reusability.

- The UserInput Class interacts with both the LocationTracking and NotificationService classes, ensuring user commands result in location requests and appropriate notifications.

- The LocationTracking Class communicates with the LocationDAO class to save and fetch location data, which guarantees that the data is consistently stored and retrievable.

- The NotificationService Class uses the UserDTO class to access and notify user-specific information, which confirms that notifications are correctly targeted.

- The LocationTracking and LocationDAO classes interact with the Location class for precise location data management, ensuring that all location-related operations have access to standardized data points.

Dynamic Model

- The sequence diagram maps each step, from the user opening the app to receiving location-based notifications, providing a clear visualization of the user journey.
- The diagram shows how the app continuously requests, updates, and displays location, ensuring the users' real-time location is accurately tracked.
- It shows how the location tracking component responds to proximity checks and user inputs, ensuring it notifies the user on time when they are near their destination.
- The layout makes it easy to expand by adding steps for new features, like additional notifications or navigation options.
- The diagram enhances the experience with proximity-based notifications, keeping the user engaged and informed throughout the journey.
- The diagram has different components:
  - App UI: Manages user inputs, displays location, and interacts with location tracking to set markers and update the interface.
  - User/Client: Provides inputs (e.g., starting location, destination) and receives location feedback via the app UI.
  - Location Tracking: Manages real-time tracking, and proximity checks, and communicates with the app UI to provide accurate location information.

TRACEABILITY FROM REQUIREMENTS TO DETAILED DESIGN MODEL

Functional Requirement Mappings

1. Customer Verifies Service Personnel's Identity
   - Requirement: Verifies personnel identity using ID or biometric data, sends notification to the customer confirming personnel's identity, and uses GPS to validate personnel's location.
   - Design Mapping:
     - *UserInput Class*: The validateUser() method in the UserInput class could handle verification requests by checking ID or biometric data, with input provided by the user.
     - *LocationTracking Class*: getCurrentLocation() fetches real-time GPS data to validate the service personnel's location.
     - *NotificationService Class*: sendVerificationAlert(userId: String) sends notifications to the customer, confirming personnel identity and location.

2. Supervisor Tracks Service Personnel in Real-Time
    - Requirement: Tracks personnel's location via GPS, allowing the supervisor to monitor the personnel's progress and receive alerts for deviations.
    - Design Mapping:
        - *LocationTracking Class*: startTracking() initiates real-time location tracking, and getCurrentLocation() continuously fetches updated locations.
        - *LocationDAO Class*: saveLocation(loc: Location) stores location data, which can later be used for monitoring and tracking.
        - *NotificationService Class*: sendArrivalNotification(userId: String) sends alerts to the supervisor if any deviation or arrival occurs.
3. Supervisor Generates Service Report
    - Requirement: Compiles location history and performance data, with options to export reports as PDF or CSV.
    - Design Mapping:
        - *LocationDAO Class*: getLocationHistory(userID: String) retrieves the location history for generating reports.
        - *Location Class*: getCoordinates() provides access to stored location data points, which can be used to compile the service report.
4. Service Personnel Downloads Software onto Customer Phone
    - Requirement: Enables tracking and identity verification through a mobile device.
    - Design Mapping:
        - *UserInput Class*: requestLocation() can be triggered once the app is downloaded, initiating tracking on the customer's phone.
        - *LocationTracking Class*: startTracking() and stopTracking() manage tracking sessions on the user's device.

<u>Non-Functional Requirement Mappings</u>

1. Performance: Location verification should be processed within 2 seconds.
    - Design Mapping:
        - The LocationTracking and LocationDAO classes are optimized to fetch and store data quickly, ensuring real-time tracking.
        - The lightweight Location class allows quick access to essential data points.
2. User Interface: Must be intuitive and accessible.
    - Design Mapping:
        - The App UI represented in the sequence diagram ensures that user flows like opening the app, requesting location, and receiving notifications are seamless and user-friendly.
3. Compatibility: Supports latest iOS and Android versions.
    - Design Mapping:
        - The NotificationService and UserInput classes are designed to integrate with mobile OS APIs, ensuring compatibility across platforms.
4. Reliability: Achieves 99.9% uptime.
    - Design Mapping:
        - *LocationDAO*: Persistent storage ensures data is consistently saved, and LocationTracking uses continuous updates to guarantee high reliability.
5. Security: Implements end-to-end encryption, multi-factor authentication, and session timeouts.
    - Design Mapping:
        - *UserDTO Class*: Secures user information by encapsulating user-specific data.
        - *UserInput Class*: Handles actions like verification, which can be linked to multi-factor authentication mechanisms.
6. Data Handling: Complies with privacy laws and enables data export.

- Design Mapping:
    - *LocationDAO*: getLocationHistory() retrieves data for export while ensuring secure data storage.
    - *NotificationService*: Ensures data privacy by targeting only verified users for notifications.

7. Scalability: Handles increasing user load without performance issues.
    - Design Mapping:
        - The modular design of classes like LocationTracking, NotificationService, and LocationDAO supports scalability by allowing independent scaling of each service.
8. Battery Optimization: Minimizes battery consumption during tracking.
    - Design Mapping:
        - *LocationTracking Class*: Allows methods like stopTracking() to disable GPS tracking when not needed, conserving battery life.

EVIDENCE THE DESIGN MODEL HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT

ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS

- IEEE Std 1016-1998-(Revision-2009): Software Design [pdf]

ADDITIONAL REFERENCES

- Larman, C., 2012. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development.* Pearson Education
- Hyman, B., 1998. *Fundamentals of Engineering Design.* New Jersey: Prentice Hall
- Simon, H.A., 2014. *A Student's Introduction to Engineering Design: Pergamon Unified Engineering Series* (Vol. 21). Elsevier