Final Project Report

Executive Summary

This report brings together all key aspects of our project, including planning, requirements, architecture, detailed design, and testing. It documents how we developed a real-time location tracking application for service personnel, focusing on reliability, usability, and scalability. Each section highlights the steps taken to ensure the project met its goals, balancing technical, social, and environmental considerations to deliver a robust solution.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 - Purpose and Scope

The purpose of the product is to improve security, trust, and efficiency during service visits to customer homes through location tracking and verification, as outlined by our stakeholder.

1.2 - Product Overview

The product aims to use Real-Time Location Tracking, Identity Verification, Reporting and analysis and device integration in order to solve pre-existing pain points in user scenarios such as verifying the identity of service personnel, reporting service to supervisors as a service worker, or even supervisors tracking service personnel in real time to provide additional customer insight.

1.3 - Structure of the Document

This final report outlines the full scope of our software development life cycle from end to end of our group project over the course. Following this introduction, which outlined the scope, purpose, and structure of the application, we will delve deeper into each stage and deliverable in the software development process. Firstly, we will outline our Project Management Plan, then transition into our Requirements Specifications, afterwards we define our Architecture, display our Design process, emphasize our Test Plan, then close off with our configuration management information and any relevant references.

1.4 - Terms, Acronyms, and Abbreviations

UI/UX: User Interface/User Experience; design and usability of the app.
API: Application Programming Interface; enables communication between software.
GPS: Global Positioning System; used for real-time location tracking.
IDEs: Integrated Development Environments; tools for coding and testing.
SME: Subject Matter Expert; team member with specialized knowledge.
QA: Quality Assurance; ensures the app meets quality standards.
MFA: Multi-Factor Authentication; enhances security with multiple verification steps.
CSV: Comma-Separated Values; a file format for data export.
PDF: Portable Document Format; a standard file format for documents.
SDK: Software Development Kit; tools and libraries for app development.
DAO: Data Access Object; manages database access in software.
DTO: Data Transfer Object; transfers data between software layers.
Waterfall Model: A sequential software development approach.
Geolocator Package: Tool for GPS access in Flutter applications.
C4ISR: Military systems for intelligence and communication; inspiration for app features.
Real-Time Updates: Immediate data updates as events occur.
End-to-End Encryption: Data security ensuring only authorized access.
Event-Driven Architecture: Design where actions respond to specific events.

2. Project Management Plan

Project Organization
Our team consists of 6 people, seated under Fellows Consulting Group. The designated group leader is Abdullah Chaudhry, which was decided upon on a volunteering-basis.
We communicate through an iMessage group chat for updates on deliverables, scheduling and submission, and any information necessary on a group level.

We have arranged to meet on a weekly basis at 4 pm to check in with our stakeholder, in order to review our final submissions. Following our weekly 30 minute check-in, we meet virtually as a team for 40 minutes to an hour in order to formally approve submission on a team level.

Given the project requirements and traditional roles within development, we have listed 6 roles below that our team have discussed and assigned according to our pre-existing skillsets with names listed beside each role, in order to ensure speedy delivery. These roles aren't strict, however they will be used as a guideline for project responsibilities/SMEs within the team.

Core Roles:
1. **Project Manager:** Oversees the entire project, ensures deadlines are met, and manages team dynamics. ***Abdullah***
2. **UI/UX Designer:** Creates the app's interface, user experience, and wireframes, ensuring a seamless and intuitive user journey. ***Mageto***
3. **Front-End Developer:** Builds the app's user interface and handles interactions using technologies like HTML, CSS, and JavaScript. ***Omar***
4. **Back-End Developer:** Develops the server-side logic and API for the app, handling data storage, retrieval, and location verification processes. ***Abdullah, Alvin***

Supporting Roles:
5. **Quality Assurance Engineer (QA):** Tests the app throughout development, identifies bugs, and ensures it meets quality standards. ***Danya***
6. **Mobile Developer Specialist:** Provides expertise in mobile development platforms (e.g., iOS, Android) and assists in platform-specific optimizations. ***Reya***

Life Cycle Model Used
For this project, we decided to use the Waterfall Model, since the project length is on the shorter side and the requirements are generally well-defined and fixed. The Waterfall Model consists of phases that are executed sequentially. Starting with requirements gathering, the lifecycle then proceeds to the architectural design, followed by the implementation phase, testing phase, and finally the deployment phase. This model provides simplicity and ease of management, as everything is in a chronological order and the milestones are clearly planned out.

Risk Analysis
Every project has its risks, and recognizing them early on is key to staying on track, on budget, and meeting the required quality standards. For this mobile location verification app we're developing we've identified some potential risks, looked at how likely they are to happen, and thought of ways to handle them if they do.

Identified Risks:

1. **Technical Challenges:**
   Risk: Integrating real-time location tracking with identity verification could get tricky. We might face issues like inaccurate location data, slow response times, or challenges in working across both iOS and Android platforms.
   Likelihood & Impact: Medium, High
   How to handle it: We'll start with early prototyping and testing of the most important features—like location tracking and identity verification. This will help us spot problems early. Plus, regular code reviews and using strong, reliable frameworks should keep technical risks under control.
   Rationale: Integrating real-time location tracking with identity verification needs to be accurate and responsive. If location data is off or there are delays, it could impact user trust and overall experience. Also, ensuring it works well on both iOS and Android adds another layer of difficulty. Tackling this risk early through prototypes and testing helps catch and resolve issues before they become bigger problems.
2. **Resource Availability:**
   Risk: The project needs certain resources, like smartphones for testing and the right development environments. Any delays in getting these resources could slow things down.
   Likelihood & Impact: Low, Medium
   How to handle it: We'll make sure we know exactly what resources we need from the start and have them ready early. If there's any delay, we'll have backup options in place to keep things moving.
   Rationale: The development process relies on having the right hardware and software, especially smartphones for testing. If there are delays in getting these resources, it could slow down progress, push back deadlines, and affect testing quality. To avoid this, we need to ensure all necessary resources are available from the start, with backup plans in place to keep things moving smoothly.
3. **Team Communication Issues:**
   Risk: If the team isn't communicating well, it could lead to confusion, duplicated efforts, or missed deadlines.
   Likelihood & Impact: Medium, Medium
   How to handle it: We'll set up clear communication protocols, with regular team meetings, detailed documentation, and tools like Microsoft Teams. Open communication is key and it will be encouraged across the team.
   Rationale: Miscommunication can lead to inefficiencies in the project. If team members aren't clear on their responsibilities or deadlines, it can result in duplicated work or missed deadlines. Establishing clear communication channels and scheduling regular check-ins ensures everyone is aligned and potential misunderstandings are addressed promptly.
4. **Timeline Overruns**
   Risks: Unexpected challenges or delays could cause the project to fall behind schedule.
   Likelihood & Impact: Medium, High
   How to handle it: We'll break the project into manageable phases. Regularly monitoring our progress will allow us to reallocate resources as needed to stay on track. We'll also build buffer time for critical tasks, just in case something takes longer than expected.
   Rationale: Unexpected challenges, such as technical issues or resource delays, can cause timelines to slip. By breaking the project into phases, we can track progress more effectively and identify potential delays early. Adding buffer time to critical tasks ensures that delays don't jeopardize the overall schedule.
5. **Scope Change Risk:**
   Risks: There's a chance that stakeholders might ask for extra features as the project progresses, which could increase our workload and cause delays.
   Likelihood & Impact: Medium, High
   How to handle it: We'll clearly define and document the project scope from day one. If any changes come up, we'll follow a formal process to assess their impact and get approval before

making adjustments.
Rationale: Without a clearly defined scope, there's a risk that stakeholders might request additional features or changes, which can increase workload and cause delays. By clearly defining the project scope and implementing a formal approval process for changes, we can manage expectations and prevent it from overwhelming the team.

6. **Quality Control Issues:**
   Risks: If testing or quality assurance falls short, we could end up delivering a product that doesn't meet the required standards or stakeholder expectations.
   Likelihood & Impact: Medium, High
   How to handle it: We'll implement a thorough testing plan, including unit tests, integration tests, and user acceptance tests. Along the way, we'll have clear quality checkpoints to make sure everything is meeting expectations.
   Rationale: A product that doesn't meet quality standards can damage the project's success. Inadequate testing can lead to bugs, poor performance, or failure to meet stakeholder requirements. Implementing a thorough testing strategy from the start ensures the product meets high standards and reduces the risk of issues after launch.

7. **Stakeholder Misalignment:**
   Risks: If the project team and stakeholders aren't on the same page about the objectives or deliverables, it could lead to dissatisfaction or rework.
   Likelihood & Impact: Low, High
   How to handle it: We'll involve stakeholders in the planning process to make sure everyone is aligned. Regular reviews and updates will help keep us all on the same page throughout the project.
   Rationale: If the project team and stakeholders have different understandings of the project's goals or deliverables, it can lead to dissatisfaction, rework, or even failure. Involving stakeholders from the beginning ensures alignment and helps manage expectations throughout the project.

Software and Hardware Resource Requirements
(Do not forget to describe what *new* software or hardware each team member learned during the project)

Hardware Resource Requirements

- Smartphones to download the app and test it
- Laptops with the required IDE's installed
- Internet Connection

Software Resource Requirements

Operating System:
    Windows 11
    Latest Version of MacOS
Development Environment:
    Flutter
    Android Studio
    Xcode
Programming Languages and Frameworks:
    Swift
    JavaScript
    Java/Kotlin
Web and Server Tools

Apache
Version Control:
Github

New Software:
The whole team learned flutter, android and xcode emulators.

Deliverables and Schedule

1. Research

Activity: Find information related to the project. Discover similar open source applications, discover how iCloud/find works, discover how military C4ISR products perform location functions, and present the results of research to one team
Dependencies: Functional team
Estimated Time: 1 Week
Team Allocation: PM, UI/UX Designer, Frontend Developer, Backend Developer, QA Engineer, Mobile Developer
Deliverables: Research presentation
Rationale: Have a deep understanding and clear plan of the project.

2. Project Initialization

Activity: Define project requirements and objective
Dependencies: Stakeholder meetings to gather requirements
Estimated Time: 2 weeks
Team Allocation: PM, UI/UX Designer, Frontend Developer, Backend Developer, QA Engineer, Mobile Developer
Deliverables: Project requirements document, stakeholder meeting notes, and approvals
Rationale: It is critical to understand project scope, objectives, and constraints to predict the project's progress accurately

3. System Design

Activity: Design Architecture and User Interface
Dependencies: Completed analysis of requirements
Estimated time: 3 weeks
Team Allocation: PM, UI/UX Designer, Frontend Developer, Backend Developer, Mobile Developer
Deliverables: System architecture design, UI mockups, Technical specification document
Rationale: This is the technical foundation for the app and is mandatory for a smooth development process. Scalability, reliability, and easy-use design are prioritized during this phase.

4. Environment Setup

Activity: Configure development environments
Dependencies: Finalized system design
Estimated time: 1 week
Team Allocation: Frontend Developer, Backend Developer, Mobile Developer
Deliverables: Fully configured development and testing environments, Setup guide for developers
Rationale: Developers need a fully functional environment to build and test system

5. Backend Development

Activity: Develop API and Backend Services
Dependencies: Completed system design and environment setup
Estimated time: 6 weeks
Team Allocation: Backend Developer, Mobile Developer
Deliverables: Fully functional backend code, unit test reports, database schema
Rationale: The backend must have robust data handling, user authentication, and location validation services

6. Frontend Development

Activity: Implement mobile app user interface
Dependencies: completed system design and environment setup
Estimated time: 4 weeks
Team allocation: UI/UX Designer, Frontend Developer
Deliverables: Mobile app UI codebase, mobile app navigation
Rationale: Meet mobile design standards and accessibility guidelines

7. Integration and Testing

Activity: Integrate Backend with Frontend, Conduct System Testing
Dependencies: Complete backend and frontend
Estimated time: 4 weeks
Team allocation: UI/UX Designer, Frontend Developer, Backend Developer, QA Engineer, Mobile Developer
Deliverables: Integrated mobile application with backend services, integration testing report
Rationale: Testing ensures both systems work together seamlessly

8. User Acceptance Testing

Activity: Conduct User Acceptance Testing with select users
Dependencies: Completed integration and system testing
Estimated time: 2 weeks
Team allocation: QA Engineer
Deliverables: User Acceptance Testing test plan, User Acceptance Testing feedback report
Rationale: User acceptance testing validates that the system meets the stakeholder's requirements and real feedback from the user means the product is reliable.

9. Documentation and Training

Activity: Prepare User Manuals and Conduct Team Training
Dependencies: Completed User Acceptance Testing
Estimated time: 2 weeks
Team allocation: PM
Deliverable: User manuals, training materials, training completion report
Rationale: Ensure users can efficiently use and maintain the system after deployment

10. Deployment

Activity: Deploy the application
Dependencies: Successful User Acceptance Testing and training completion
Estimated time: 1 week
Team allocation: PM, UI/UX Designer, Frontend Developer, Backend Developer, QA Engineer, Mobile Developer
Deliverables: Deployed application, deployment checklist
Rationale: The application is officially released to users

11. Post-Deployment Support

Activity: Provide ongoing support and bug fixes
Dependencies: Deployment
Estimated time: 4 weeks
Team allocation: QA Engineer
Deliverables: Post-deployment support plan, bug fix reports
Rationale: Ensure the system is smooth post-launch and handle all issues in production

Monitoring, Reporting, and Controlling Mechanisms
We will produce three distinct types of reports to ensure proper documentation of our developmental process, those being weekly reports, milestone reports, and lastly our final report.

Weekly Reports
Weekly reports will be utilized as a means of keeping up with individual task distribution, and work from

the perspective of us as individuals. We will document each person’s developments, roadblocks, and estimated timeline on delivery in order to ensure even balancing of work.

Milestone Reports
Milestone Reports will be attached to each deliverable we submit, which will encompass all of the weekly reports.

Final Report
The final report will cover the project's organization, risk analysis, resource requirements, and impact on individuals and organizations, alongside the architecture, design, and test planning. The report also will include evidence of our adherence to engineering standards and configuration management, and concludes with references and acknowledgments, demonstrating our group’s application of software engineering principles and learning outcomes.

PROFESSIONAL STANDARDS

Scholastic Dishonesty

Commitment to Original Work

Expected Behavior: Each member is expected to produce original work and contribute their own ideas to the project. When using external resources such as libraries, frameworks, or online content, members must ensure they properly cite sources.

Rationale: Original contributions not only reflect the individual’s capabilities but also enhance the overall quality of the project. Proper citation practices acknowledge the efforts of others and uphold the integrity of the team’s work. This commitment ensures that the project is a true reflection of the team’s skills and knowledge.

Open and Honest Collaboration

Expected Behavior: Team members should engage in honest collaboration, sharing ideas and feedback while providing credit where it is due. If someone utilizes another team member's code or concepts, they must acknowledge that contribution openly.

Rationale: Collaboration is essential in a team environment, and honesty in these interactions builds trust among members. Acknowledging contributions fosters a supportive atmosphere where individuals feel valued and encourages the free exchange of ideas, leading to more innovative solutions.

Addressing Dishonesty

Expected Behavior: If a team member suspects or witnesses any form of academic dishonesty, they are expected to address the issue promptly. This may involve a direct conversation with the individual involved or reporting the concern to the stakeholder.

Rationale: Allowing dishonesty to go unaddressed can undermine the integrity of the whole project and the team’s reputation. Reporting concerns ensures that issues are handled appropriately and reinforces the importance of accountability. It also demonstrates that the team prioritizes ethical behavior and is committed to maintaining high standards.

Meeting Schedule

Punctuality and Preparedness
Expected Behavior: Team members are expected to arrive on time for meetings and come prepared with relevant materials, updates on their tasks, and any questions or issues they wish to discuss.

Rationale: Punctuality shows respect for each other's time and commitments. Being prepared allows meetings to run efficiently and productively, maximizing the time spent together. It also illustrates professionalism and a commitment to the project's success.

Setting Clear Agendas
Expected Behavior: Each meeting should have a clear agenda for all team members to know about prior to the weekly meetings. The agenda should outline the topics to be discussed, the goals for the meeting, upcoming deliverables, and any specific time allocations for each task that needs to be done.
Rationale: A defined agenda helps keep meetings focused and organized, ensuring that all necessary topics are covered within the allotted time. It allows team members to prepare for discussions and contributes to more effective and meaningful conversations.

Action Items and Accountability
Expected Behavior: At the end of each meeting, the team should summarize action items, assigning responsibilities and deadlines for each task. These action items should be documented and shared with all members after the meeting.

Rationale: Clearly defined action items ensure that everyone knows their responsibilities moving forward. This accountability helps maintain momentum between meetings and ensures that tasks are completed in a timely manner, contributing to the overall project goals.

Follow-Up and Progress Check-Ins
Expected Behavior: Team members should provide updates on their assigned action items at the beginning of each meeting, discussing progress, challenges, and any support they may need from the team.

Rationale: Regular follow-ups allow the team to assess progress and identify potential roadblocks early. This transparency fosters collaboration and encourages team members to seek help when needed, ensuring that the project stays on track.

Quality Expectations for Tasks & Deliverables

Time Management and Prioritization
Expected Behavior: Team members are expected to manage their time effectively and prioritize tasks based on project timelines and deliverables. This includes setting realistic deadlines for their work and adhering to the project schedule.

Rationale: Effective time management is essential for meeting project deadlines and ensuring that all tasks are completed to a high standard. Prioritization helps team members focus on critical tasks first, reducing the risk of last-minute rushes that can compromise quality.

Continuous Learning and Improvement

Expected Behavior: Team members should engage in continuous learning by seeking opportunities to learn new skills and knowledge related to software development practices. This can include learning new languages and studying new technologies within the scope of the project to help accomplish deliverables.

Rationale: The technology landscape is constantly evolving, and continuous learning ensures that team members remain up-to-date with best practices and emerging trends. This commitment to learning can lead to higher quality deliverables and more innovative solutions.

Meeting Project Specifications and Deliverables

Expected Behavior: Team members should regularly review the project specifications to ensure their work remains aligned with the given deliverables and that they meet these deliverables on time.

Rationale: Adhering to project specifications ensures that the final product meets the needs of stakeholders and users. Regularly reviewing requirements ensures that all team members are focused on the same objectives, which is crucial for project success.

Resolving Unacceptable Behavior

Initial Observation and Documentation

Expected Behavior: Team members should remain observant and document any incidents of unacceptable behavior as they occur. This can include noting specific actions, the context in which they occurred, and how they impacted the team or project.

Rationale: Documentation is crucial for understanding the full scope of the issue and provides a factual basis for any discussions or interventions. It helps to avoid misunderstandings and ensures that any actions taken are grounded in specific incidents rather than generalizations.

Private Discussion

Expected Behavior: There should be a private conversation between the involved parties. The team member who observes the behavior should approach the individual in a non-confrontational manner, expressing concerns about the specific behavior rather than making personal judgments.

Rationale: Open dialogue allows for clarification and understanding. The team member may not be aware that their behavior is perceived as unacceptable. A private discussion helps to maintain the dignity of the individual while creating an opportunity for resolution.

Expressing Impact and Concerns

Expected Behavior: During the conversation, the observing team member should clearly articulate how the behavior affects the team dynamic, project progress, and overall morale. It is important to use "I" statements to express personal feelings and observations, such as "I felt concerned when the deadlines were not met because it impacts the entire team."

Rationale: By focusing on the impact of the behavior rather than labeling the person, the conversation stays constructive. This approach encourages reflection and reduces defensiveness, facilitating a more open exchange of ideas.

Collaboratively Identifying Solutions

Expected Behavior: After discussing the issue, both parties should work together to identify practical solutions to prevent future occurrences. This may involve setting clear expectations, adjusting responsibilities, or providing additional support.

Rationale: Collaborative problem-solving empowers team members to take ownership of their behavior and commitments. It fosters a sense of teamwork and demonstrates that the group values each member's contributions.

Involving Team Leadership or Advisors

Expected Behavior: If the behavior persists despite private discussions, or if the situation is severe (e.g., harassment, discrimination, or other serious breaches of conduct), it will be necessary to involve the stakeholder for mediation.

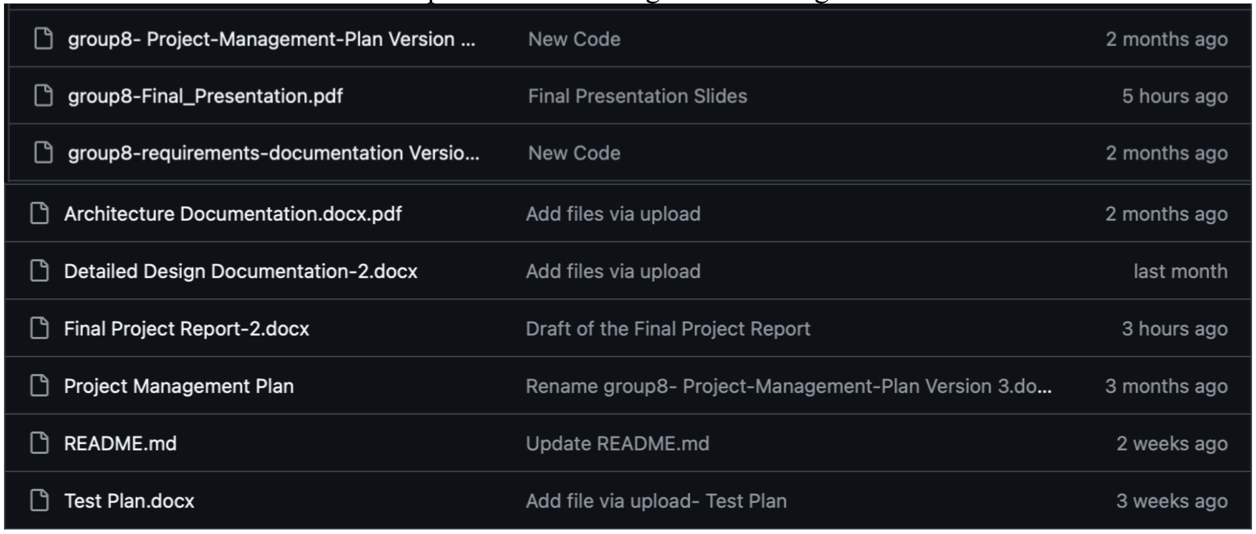
Rationale: Involving a third party can provide an objective perspective and facilitate a resolution. Those who are in leadership roles often have experience in handling interpersonal conflicts and can offer guidance on appropriate next steps.

Monitoring Progress

Expected Behavior: The team should monitor the situation over time to ensure that the behavior has improved and that the solutions are effective. Regular feedback sessions can help maintain open lines of communication.

Rationale: Continuous monitoring demonstrates a commitment to resolving the issue and reinforces the importance of maintaining a positive team environment. It allows for adjustments to be made if problems recur.

Evidence all the artifacts have been placed under configuration management



group8- Project-Management-Plan Version ...	New Code	2 months ago
group8-Final_Presentation.pdf	Final Presentation Slides	5 hours ago
group8-requirements-documentation Versio...	New Code	2 months ago
Architecture Documentation.docx.pdf	Add files via upload	2 months ago
Detailed Design Documentation-2.docx	Add files via upload	last month
Final Project Report-2.docx	Draft of the Final Project Report	3 hours ago
Project Management Plan	Rename group8- Project-Management-Plan Version 3.do...	3 months ago
README.md	Update README.md	2 weeks ago
Test Plan.docx	Add file via upload- Test Plan	3 weeks ago

Figure 1: All Artifacts are in Github

Our project enhances efficiency and trust in service delivery by enabling real-time tracking of personnel, ensuring accurate location monitoring. For individuals, it improves safety and convenience by providing transparency and accountability. On a broader scale, it contributes to public welfare by promoting reliable service standards and fostering trust between service providers and clients.

The development process considered global and cultural diversity by ensuring accessibility and adaptability across regions. Social and economic factors were addressed by prioritizing affordability and user-friendly design. Environmental considerations include efficient resource use and minimizing energy consumption during app operation.

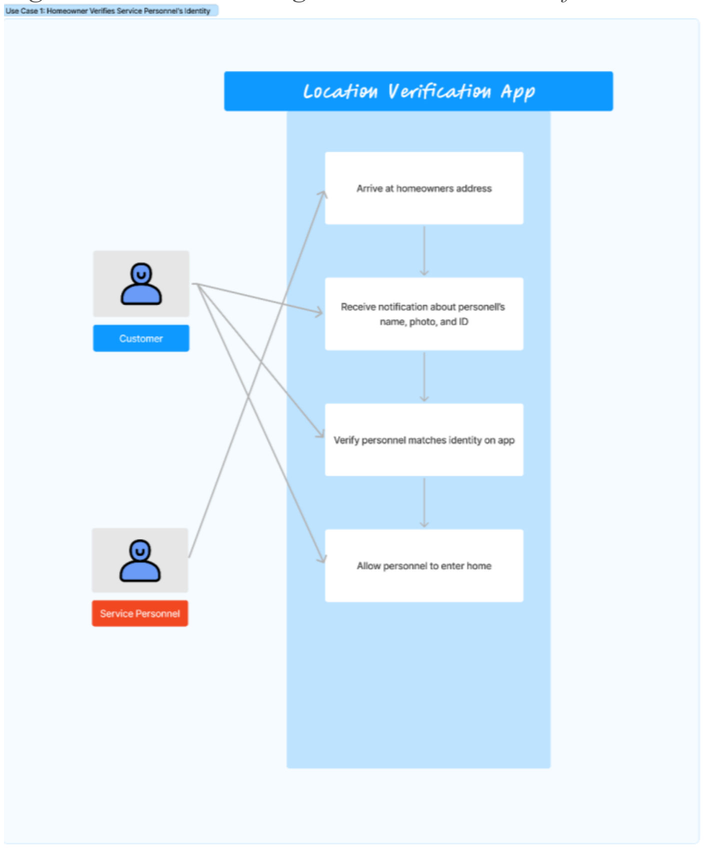
3. Requirement Specifications

Stakeholders for the system

1. Service Personnel
2. Clients
3. Development Team
4. Stakeholder

Use case model for functional requirements

Figure 2 - Use Case Diagram 1: Customer Verifies Service Personnel's Identity

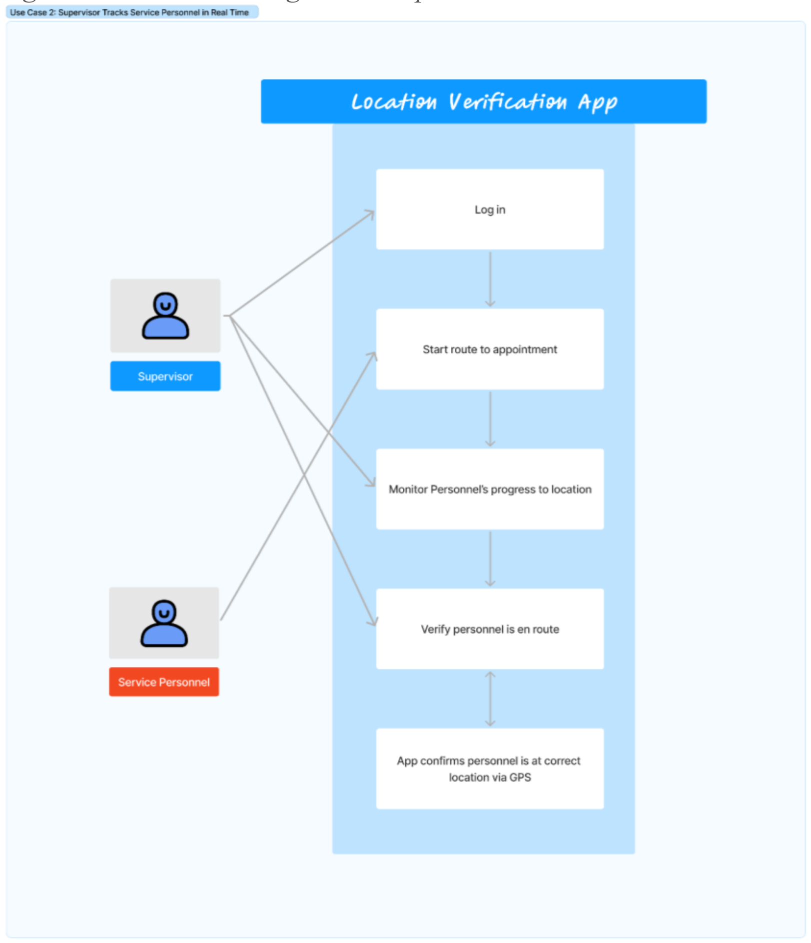

Use Case 1: Homeowner Verifies Service Personnel's Identity
Location Verification App
Arrive at homeowners address
Receive notification about personell's name, photo, and ID
Verify personnel matches identity on app
Allow personnel to enter home
Customer
Service Personnel

- **Use Case Name:** UC1 - Customer Verifies Service Personnel's Identity
- **Actors:** Homeowner, Service Personnel
- **Entry Conditions:**
    - Service personnel installed app

        - Customer has access to app
        - Service appointment is scheduled and recorded into system
- **Normal Flow of Events:**
        - Service personnel arrives to customer's house for appointment
        - Mobile app verifies service personnel's identity using unique ID or biometric Data (fingerprint or face recognition)
        - App sends notification to customer, notifying them about personnel's name, photo, and ID
        - Customer opens app to verify person at door matches verified identity
        - App confirms service personnel is at correct location via GPS
        - Customer allows personnel to enter once personnel is verified

- **Exit Conditions:**
        - Customer successfully verified service personnel's identity and allows access
        - Homeowner denies access if identity verification fails
- **Exceptions (Alternate Flow of Events):**
        - Homeowner can't access mobile app
            - System sends SMS/call notification to customer with personnel's details
            - Customer can manually verify identity via other methods (call verification with service provider)
- **Special Requirements:**
        - Secure communication for identity verification
        - App should support multiple authentication methods
        - App must have fallback mechanisms

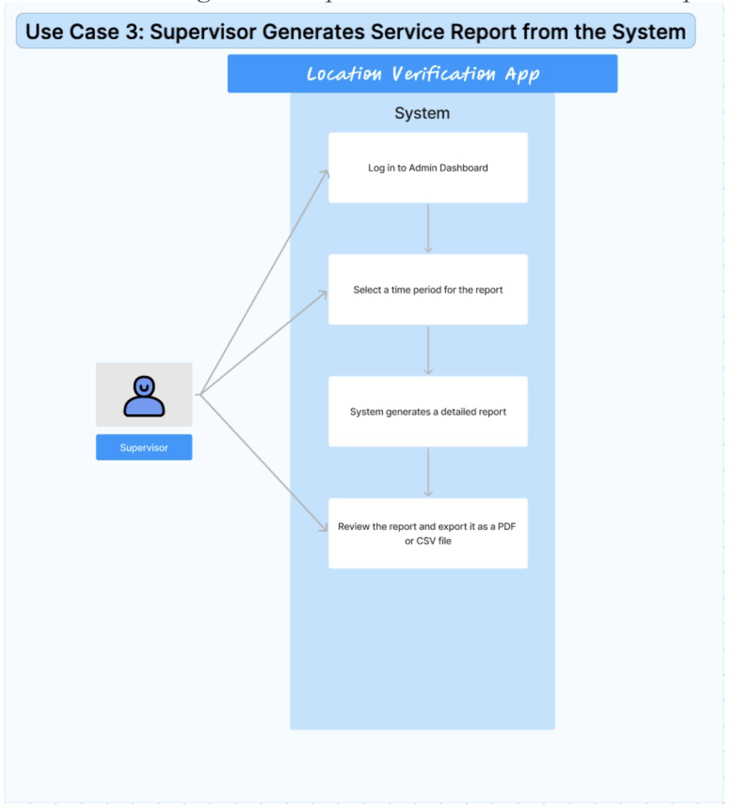
Figure 3 - Use Case Diagram 2: Supervisor Tracks Service Personnel in Real-Time


Use Case 2: Supervisor Tracks Service Personnel in Real Time
Location Verification App
Log in
Start route to appointment
Monitor Personnel's progress to location
Verify personnel is en route
App confirms personnel is at correct location via GPS
Supervisor
Service Personnel

- **Use Case Name:** UC2 - Supervisor Tracks Service Personnel in Real-Time
- **Actors:** Homeowner, Service Personnel
- Entry Conditions:
  - Service personnel is logged into mobile app
  - Service personnel enables GPS
- **Normal Flow of Events:**
  - Service personnel start route to appointment
  - Location verification app tracks personnel in real time
  - Supervisor views real-time location of personnel using admin dashboard
  - Supervisor monitors personnel's progress towards the service location
  - Supervisor is notified if there are deviations from expected route
- **Exit Conditions:**
  - Supervisor can verify personnel are en route and on time to appointment and track personnel until job is complete
- **Exceptions (Alternate Flow of Events):**
  - Personnel's GPS is off or unavailable
    - App sends reminder to enable GPS
    - If GPS unavailable for extended period, an alert is sent to the supervisor to directly contact the personnel
- **Special Requirements:**
  - Continuous GPS tracking

- App must alert supervisor if there are any tracking interruptions
- Ability for service provider to override automatic alerts (if deviations are expected)

Figure 4 - Use Case Diagram 3: Supervisor Generates Service Report from the System


Use Case 3: Supervisor Generates Service Report from the System
Location Verification App
System
Log in to Admin Dashboard
Select a time period for the report
System generates a detailed report
Review the report and export it as a PDF or CSV file
Supervisor

Use Case Name: UC3 - Supervisor Generates Service Report from the System
Actor: Supervisor
Entry Conditions:

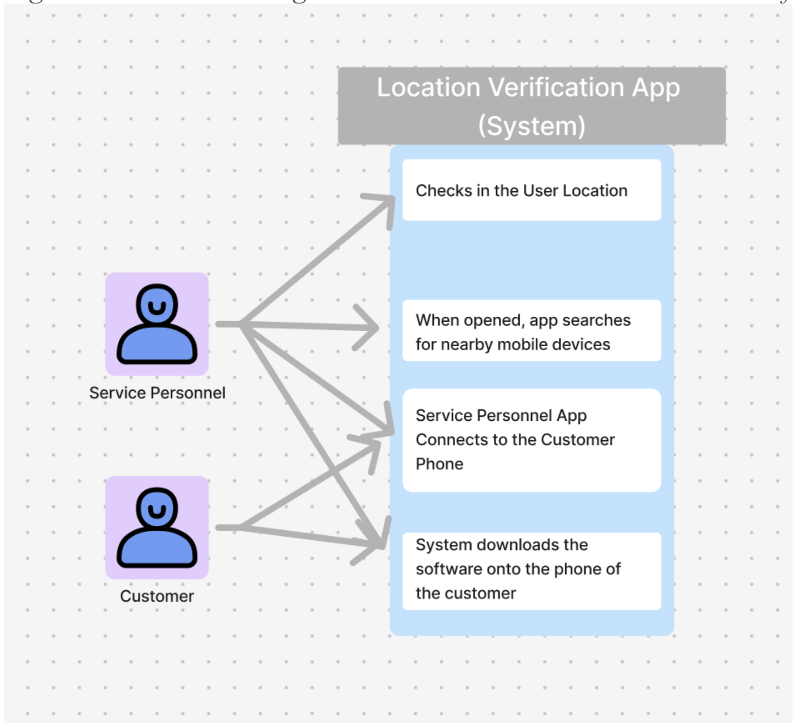
The supervisor is logged into the app's admin dashboard.
The system has recorded service personnel's location history and performance data during a selected time period.
The supervisor has access to reporting features in the app.

- **Normal Flow of Events:**
  - Supervisor logs into the admin dashboard of the app.
  - Supervisor navigates to the reporting section and selects a time period for which the report is required (daily, weekly, or monthly).
  - The System compiles data related to service personnel's location history, service completion times for the selected period.
  - Supervisor reviews the generated report in the app, and exports the report as a PDF or CSV file if needed.
- **Exit Conditions:**
  - The supervisor successfully reviews the report and optionally exports it.
  - The report is generated based on the selected criteria and presented to the supervisor.
- **Exceptions (Alternate Flow of Events):**
  - If the System does not have sufficient data for the selected time period, it notifies the Supervisor that no report can be generated for the chosen period.

- If there is a system error or issue generating the report, the System displays an error message and allows the Supervisor to retry the process or select a different time period.

- **Special Requirements:**
  - The system should have access to accurate, up-to-date service personnel data.
  - The application must be able to export reports in both PDF and CSV formats.
  - The system should handle cases where data is incomplete or missing and notify the supervisor accordingly.

Figure 5 - Use Case Diagram 4: Service Personnel downloads Software onto Customer Phone


Location Verification App (System)
Checks in the User Location
When opened, app searches for nearby mobile devices
Service Personnel App Connects to the Customer Phone
System downloads the software onto the phone of the customer
Service Personnel
Customer

- **Use Case 4:** Service Personnel downloads Software onto Customer Phone
- **Actors**: Service Personnel, Customer
- **Entry Conditions:**

  - The service personnel is within the range of the customer's phone
  - The service personnel must have the app on their phones
  - The service personnel must have internet or data based connection
- **Normal Flow of Events:**
  - The Service Personnel arrives to the location of the user
  - The System notes that the system personnel has arrived
  - The Service Personnel opens up the application
  - The System searches for nearby mobile devices
  - The Service Personnel connects to the customer's phone
  - the System downloads the software onto the phone of the user

Exit Conditions:
  - The system completes its download of the software
  - the location of the user is confirmed to be tracked

Special Requirements:

- Location is being continuously traced
- The customer device must have space to store the software

Rationale for Use Case Model:

The use case models are designed to address important security and efficiency needs in service-based businesses. The identity verification use case helps homeowners feel safe by confirming the identity of the service personnel through biometric methods and notifications. This improves security and builds trust in the process.

The real-time tracking use case helps service providers monitor their workers. Supervisors can see where service personnel are in real-time, ensuring they stay on schedule and helping if there are any issues. This improves transparency and reduces delays.

In addition, generating service reports helps supervisors track performance, increase accountability, and export data in various formats for easier analysis and decision-making. The system's error handling ensures reliability, even when data is incomplete.

Lastly, the service personnel downloads the app on the customer's phone so that their location can be tracked. This allows the service personnels phone to connect to the customer, and this allows for the identity and location to be confirmed.

These use cases enhance security, transparency, and operational efficiency while ensuring reliable performance tracking and communication in service-based businesses.

Non-functional Requirements:

- The app should be able to process location verification requests within 2 seconds to ensure timely updates
- The app should have an intuitive user interface that allows users to easily navigate and access features
- The app must be compatible with a range of devices and operating systems specifically targeting the latest two versions of iOS and Android
- The app should have a 99.9% uptime ensuring that location verification services are available at all times
- The app’s codebase should be modular and well-documented to facilitate easy updates and maintenance
- The app must implement end-to-end encryption for all location data transmission to protect user privacy and prevent unauthorized access
- The app must comply with relevant legal and regulatory standards to ensure that user data is handled in accordance with privacy laws
- The app should adhere to accessibility standards to enable easy navigation for users with disabilities
- The app must provide consistent performance across different screen sizes, resolutions, and form factors.
- The app should have the capability to integrate with third-party service provider company apps
- The app must enforce additional layers of security such as multi-factor authentication for logging into the app or accessing sensitive data.
- The app should implement session timeouts and automatic logouts after periods of inactivity to improve security.
- The app should allow users to customize alert types (e.g., SMS, push notifications) for location

updates or changes in service status.
- The app should ensure users can easily export their location history or account data in a structured format if required.
- The app should support real-time updates for continuous or frequent location tracking without significant delays.
- The app should be optimized to minimize battery consumption during location tracking and data transmission.
- The app should be able to handle an increasing number of users and location verification requests without compromising performance.

4. Architecture

Architectural style(s) used:

Client-Server Architecture: The devices communicate with a central server that manages data and authentication services

- Real-Time Data Exchange: Client-server allows for seamless real-time communication between homeowner and service personnel's mobile devices and the server
- Scalability: Client-server supports adding new features or increasing the number of users without worsening performance

Microservices Architecture: The app uses different functions (location tracking, identity verification, user management) and they are handled by separate, loosely coupled services

- Modular Development: Each feature is built as an independent service so updating each feature is easy without affecting the entire system
- Platform Independence: Microservices can be deployed independently on both iOS and Android

Event-Driven Architecture: The app uses event-driven architecture to respond to triggers (location updates, verification requests) in real-time

Real-Time Tracking: Service personnel's location is constantly updated and streamed to homeowner through event-driven mechanisms

Immediate Notifications: When the app detects significant events (ex. the service personnel arriving at location), notifications are pushed to homeowner and supervisor

Architectural model:

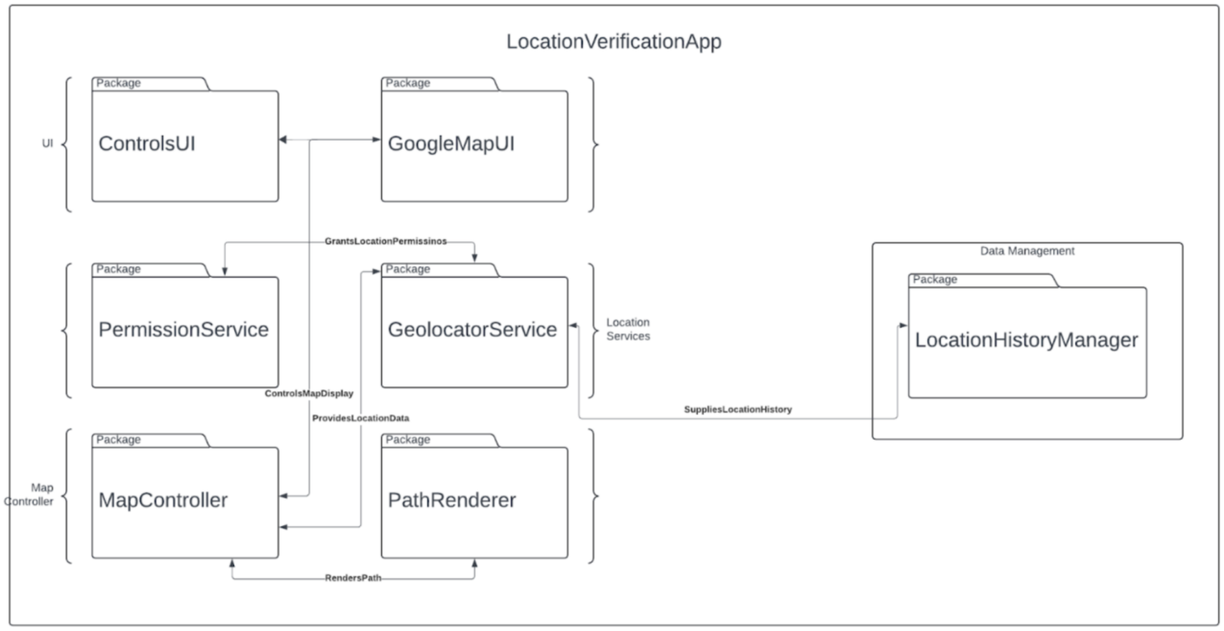


Figure 6 - Package Diagram

Technology, software, and hardware used

Technology Description: This project is implemented using Flutter, a cross-platform UI toolkit that allows for building natively compiled applications for mobile, web, and desktop from a single codebase. The core functionality of this application involves tracking the real-time location of service personnel, achieved using the `geolocator` package for accessing GPS services and obtaining location data. The app uses asynchronous programming (async/await) to handle real-time location updates and user interactions efficiently.

Software Requirements:

1. **Flutter SDK**: For cross-platform development, supporting both iOS and Android.
2. **Dart Programming Language**: Flutter applications are written in Dart.
3. **Geolocator Package**: A Dart package for accessing GPS services, allowing the app to get the current location and listen to location updates.
4. **Firebase (Optional)**: For real-time database functionalities and managing user authentication.
5. **Visual Studio Code or Android Studio**: Integrated Development Environments (IDEs) for writing, testing, and debugging the code.

Hardware Requirements:

1. **Smartphones/Tablets**: Devices running either iOS or Android, with GPS capabilities to support location tracking.
2. **Server**: A cloud server (e.g., AWS, Google Cloud) to host the backend and database for managing user data, location history, and communication between clients.
3. **Computers/Laptops**: For developing and testing the application, equipped with enough memory and processing power to run emulators and compile the application.

Communication Between Application Server and Database Server: The communication between the application server and the database server typically follows a client-server model:

- The **client app** (Flutter app) sends HTTP requests to the **application server** for operations like user authentication, location updates, and fetching location history.
- The **application server** processes the request and interacts with the **database server** to retrieve or store data.
- The **database server** (e.g., a real-time database like Firebase or an SQL/NoSQL database) stores information such as user profiles, authentication tokens, and real-time location data.
- The **application server** sends the requested data or a response back to the **client app**.
- The app uses this data to update the UI, such as displaying the current location of service personnel or generating service reports for supervisors.

This setup ensures that data exchange is secure, synchronized, and consistent across all clients while allowing the app to scale and support real-time updates.

Rationale for Architectural Style and Model:

Architectural Styles

The centralization of the client-server architecture is essential for maintaining data consistency, integrity, and quick access across various user devices. This ensures that all data, including real-time location updates and authentication, is centrally managed and synchronized. The responsiveness of the event-driven architecture is crucial where real-time data is vital for accurate tracking and instant notifications which ensures that the app can handle real-time updates and immediate responses to specific triggers like location changes or verification requests. The modular approach of the microservices architecture ensures that the system remains flexible and can easily adapt to new requirements or scale to meet increasing demand for the system. Combining these architectural styles creates a balanced system that excels in unified data management, real-time responsiveness, event handling, modularity, and scalability. By leveraging the strengths of each architecture, the app can deliver a robust and flexible solution for real-time location verification. Essentially, these architectures work synergistically to create an app that is not only efficient and reliable but also adaptable and resilient. This integrated approach ensures that every aspect of the app functions smoothly and effectively, providing users with a seamless experience.

Architectural Model

The ControlsUI handles user commands and interactions, while GoogleMapUI integrates Google Maps for real-time location visualization. This ensures users can easily view and interact with location data, providing an intuitive user experience. The PermissionService enables the app to have the necessary permissions to access location data, complying with privacy and security regulations. The GeolocatorService provides the actual location data which is fundamental for the app's primary function of tracking and verifying locations in real-time. The MapController handles the map's functionality including receiving and processing location data while the PathRenderer visualizes the service personnel’s path on the map. This ensures that users receive accurate and up-to-date location tracking which enhances the app’s reliability and effectiveness. The LocationHistoryManager maintains a record of past locations, allowing the app to offer insights and verification of historical data which is crucial for both transparency and security. Collectively, these elements of the model help create a cutting-edge app tailored to meet the demands of real-time tracking and verification.

Traceability From Requirements to Architecture:

Functional Requirements Mapping:

1. UC1 - Customer Verifies Service Personnel’s Identity:
   - PermissionService: Handles user permissions for accessing location data, ensuring secure identity verification.
   - GeolocatorService: Provides real-time location to verify the personnel’s location, ensuring identity verification in context.
2. UC2 - Supervisor Tracks Service Personnel in Real-Time:
   - GeolocatorService: Provides live location data for continuous tracking.
   - PathRenderer: Visualizes the path of the personnel in real-time on the GoogleMapUI.
3. UC3 - Supervisor Generates Service Report:
   - LocationHistoryManager: Collects and supplies location history for report generation.
4. UC4 - Service Personnel Downloads Software:
   - Not directly represented in the architecture diagram but can be handled by external integration components that could work with the UI (ControlsUI) and PermissionService.

Non-Functional Requirements Mapping:

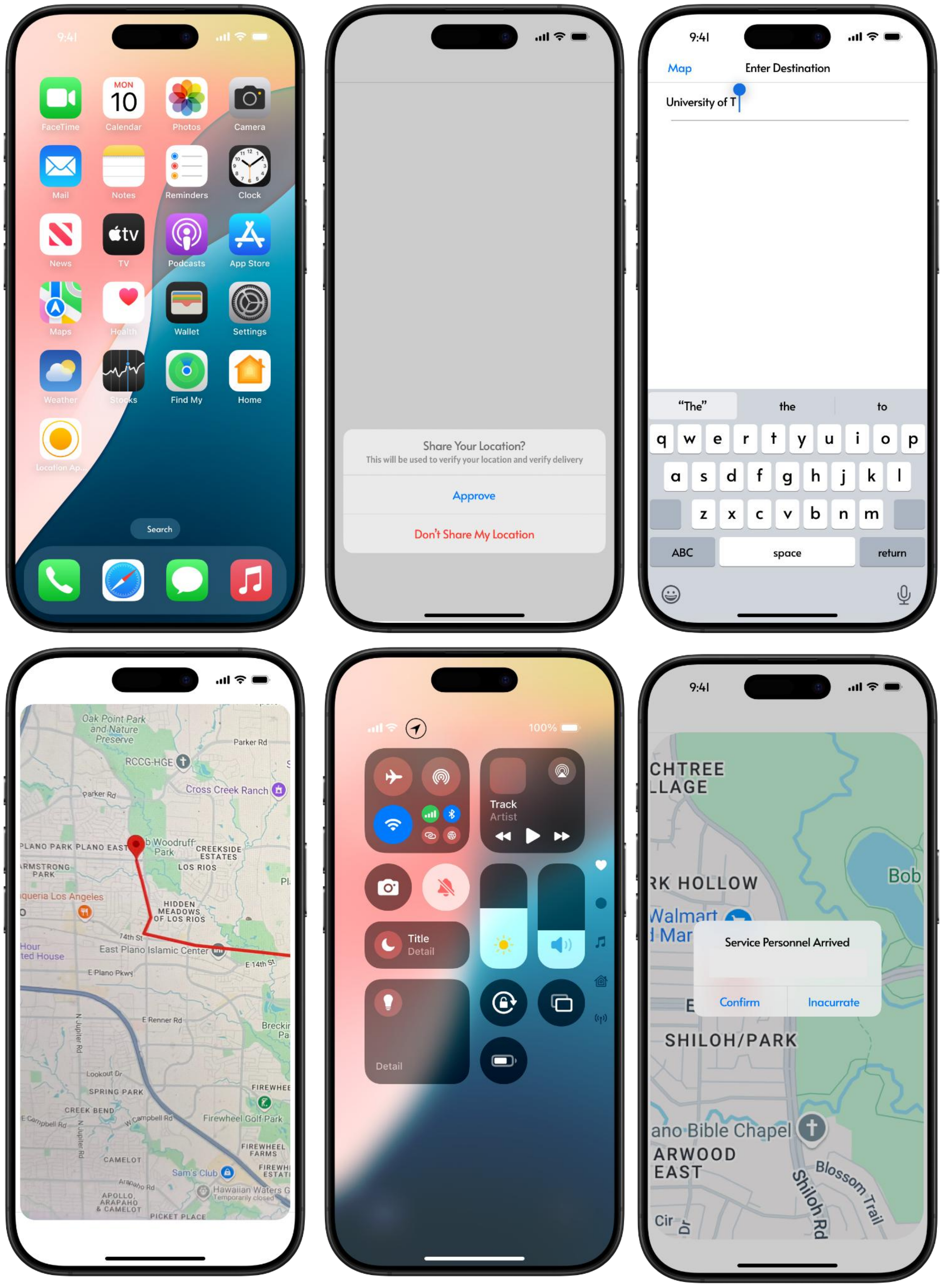
1. Processing Location Verification within 2 Seconds:
   - GeolocatorService and LocationHistoryManager ensure real-time data is delivered within a tight time frame.
2. Intuitive User Interface:
   - ControlsUI and GoogleMapUI provide a user-friendly interface for location management.
3. Platform Compatibility:
   - Modular Services (UI, PermissionService, Location Management) ensure the system can be deployed across different platforms, including iOS and Android
4. 99.9% Uptime:
   - Microservice architecture (depicted by independent packages like GeolocatorService and LocationHistoryManager) ensures high availability by isolating components for better fault tolerance.
5. Modular Codebase:
   - Each service (PermissionService, GeolocatorService, LocationHistoryManager) operates independently, facilitating modular updates and ease of maintenance.
6. End-to-End Encryption:

- Secure transmission for location data between GeolocatorService and LocationHistoryManager (encryption implied but not shown in the diagram).

7. Regulatory Compliance:
    - LocationHistoryManager handles location data storage and could be designed to ensure it complies with legal regulations regarding data privacy.
8. Accessibility Standards:
    - ControlsUI and GoogleMapUI can incorporate accessibility features for improved usability.
9. Consistent Performance Across Devices:
    - The UI components (ControlsUI, GoogleMapUI) ensure consistent rendering across different devices and resolutions.
10. Multi-Factor Authentication (MFA):
    - While MFA is not directly represented, it would integrate with PermissionService for secure access.
11. Session Timeouts and Auto Logouts:
    - This could be managed via PermissionService, ensuring sessions are handled securely.
12. Customizable Alerts:
    - PathRenderer could trigger customizable event-driven notifications for significant location changes or updates.
13. Data Export (Location History):
    - LocationHistoryManager manages historical location data, enabling easy export for reporting purposes.
14. Real-Time Location Updates:
    - GeolocatorService provides continuous real-time updates for service personnel's location.
15. Optimized Battery Consumption:
    - GeolocatorService can be optimized for efficient data transmission to reduce battery usage.
16. Scalability:
    - The modular design of the system (independent services like GeolocatorService and LocationHistoryManager) supports scalability for increasing users and location verification requests.

5. Design

GUI (Graphical User Interface) design


Share Your Location?

This will be used to verify your location and verify delivery

Service Personnel Arrived

Figure 7: User Interface of the Overall Application Experience

Static model – class diagrams

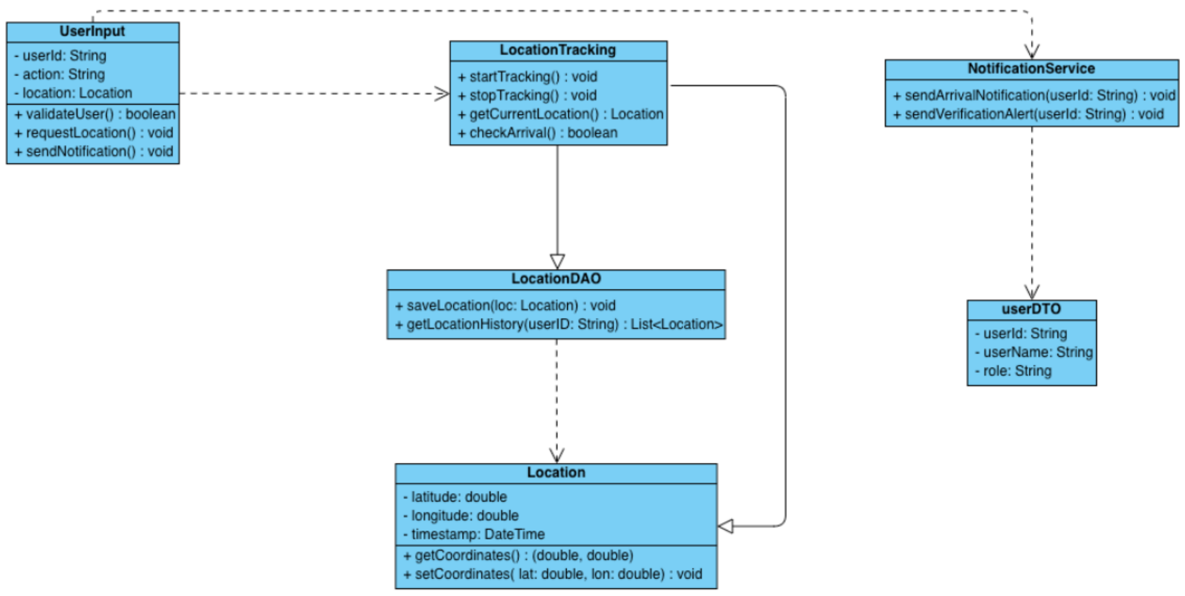

UserInput
- userId: String
- action: String
- location: Location
+ validateUser() : boolean
+ requestLocation() : void
+ sendNotification() : void
LocationTracking
+ startTracking() : void
+ stopTracking() : void
+ getCurrentLocation() : Location
+ checkArrival() : boolean
NotificationService
+ sendArrivalNotification(userId: 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

Figure 8: Class Diagram of the Location Tracking App

Dynamic model – sequence diagrams

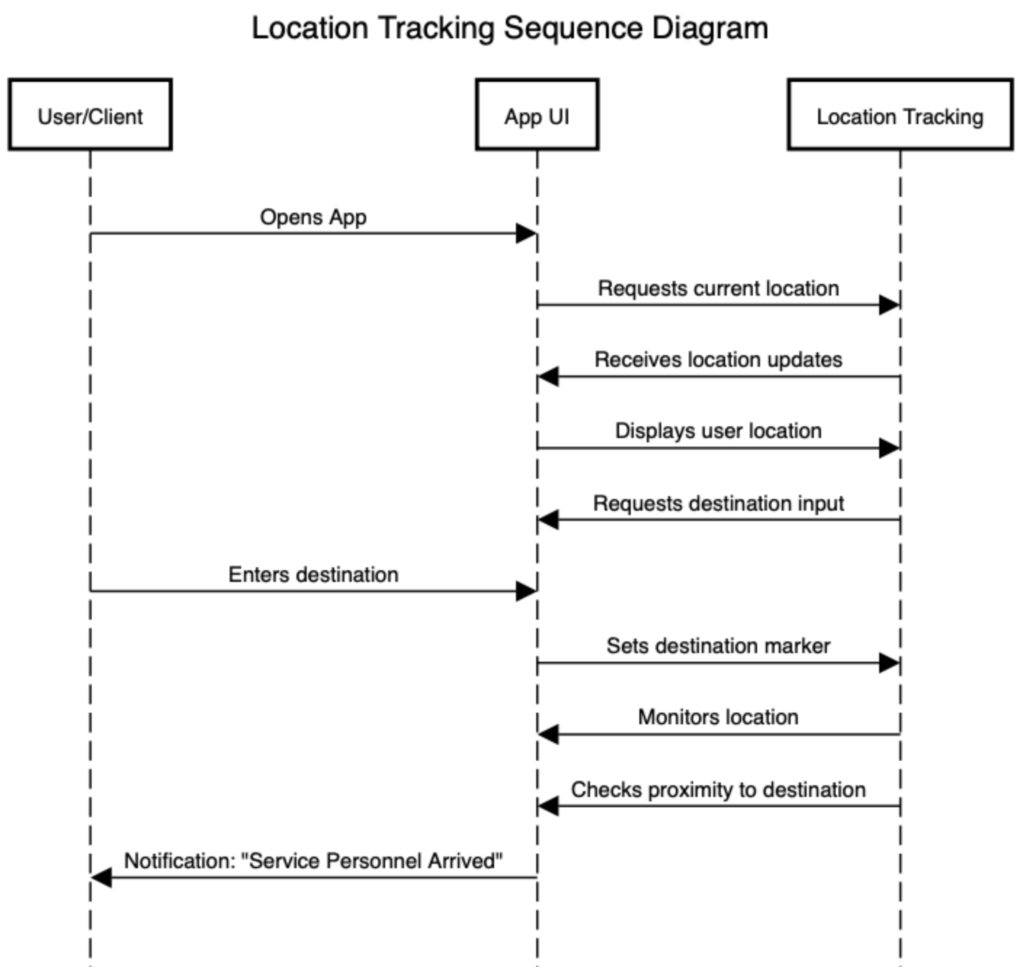

Location Tracking Sequence Diagram
User/Client
App UI
Location Tracking
Opens App
Requests current location
Receives location updates
Displays user location
Requests destination input
Enters destination
Sets destination marker
Monitors location
Checks proximity to destination
Notification: "Service Personnel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

Figure 9: Sequence Diagram of the Location Tracking App

Rationale for 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 personnels 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.

Non-Functional Requirement Mappings

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.

6. Test Plan

Requirements/specifications-based system level test cases

Test Case ID	**Test Description**	**Test Data**	**Expected Result**	**Pass/Fail**
TC01	Verify the app requests location permissions on startup	Launch the app	App prompts the user to allow location permissions.	Pass
TC02	Verify the map centers on the user's current location	Grant location permission	Map should display the user’s current location as the center of the screen.	Pass
TC03	Verify destination search displays marker on map	Enter a valid address	Marker appears on the map at the destination address.	Pass
TC04	Verify arrival notification at the destination	Set destination nearby and simulate driving toward it	App shows a notification indicating arrival at the destination.	Pass

TC05	Verify app's behavior on invalid address entry	Enter invalid address	App shows an error message or prompt requesting a valid address.	Pass
TC06	Verify map's responsiveness and accuracy	Use simulated driving route	Map follows the route accurately and updates the user's position in real-time without delays	Pass
TC07	Verify that the supervisor can export reports in both PDF and CSV format	Supervisor logged into the admin dashboard, navigates to reporting section, selects a time period (e.g., weekly)	The report is generated and successfully exported as a PDF and CSV file	Not Implemented
TC08	Verify that the app detects nearby mobile devices	Service personnel within range of a customer's phone, Bluetooth or Wi-Fi enabled on both devices	The system detects and lists the customer's phone as a nearby device	Not Implemented
TC09	Verify the software download process to customer's phone	Wi-Fi connected on customer device, and sufficient storage space available	Software is downloaded successfully onto the customer's phone	Pass
TC10	Verify route display from user to delivery personnel	Enter user and delivery personnel locations	App displays the route from the user's location to the delivery personnel's location	Pass
TC11	Verify user receives estimated time of arrival (ETA) updates	Track delivery personnel and simulate changes in speed or route	App updates the ETA to the user based on the current location and route	Not Implemented

TC12	Verify app's behavior when switching between different network connections	Switch from Wi-Fi to mobile data during tracking	App maintains real-time updates and notifies the user of the network change without interrupting the tracking process	Not Implemented

Table 1: Test Cases

Techniques used for test generation
In designing test cases for our location-based application, we applied the following techniques for test generation:
Requirements/Specifications-Based Testing:
We designed test cases directly from the app's documented requirements to ensure its core functionality. Examples include:
- TC01: Verifies that the app requests location permissions when it starts.
- TC02: Ensures that the map centers on the user's location once permission is granted.
- TC04: Checks that an arrival notification appears when the user reaches their destination.

Boundary Value Analysis:
In this technique, we focused on testing edge cases to see how the app behaves at its limits. Examples include:
- TC04: Tests if the app correctly triggers notifications as the user gets close to their destination.
- TC05: Tests entering an invalid address to make sure the app prompts the user to provide a valid one.

Error Guessing:
We used error guessing to think of common mistakes users might make and created test cases for these possible failure points. Here are some examples:
- TC05: Tests how the app handles invalid addresses. We expect the app to show an error or prompt the user to enter a valid address.
- TC12: Switches network connections while tracking the user's location. This tests whether the app stays responsive and continues to update the location data correctly even when the network changes.

Code Coverage Analysis and Path Testing:
We used code coverage analysis to make sure we tested all important parts of the app's code. Path testing helped us confirm that the app's logic flows correctly.
- TC01: Tests if the app asks for location permissions when it starts. This checks if the permission request logic works the way it should.
- TC02: Verifies that the map centers on the user's location after permission is granted. This checks that the app correctly retrieves and displays the user's location.
- TC04: Makes sure the app sends an arrival notification when the user reaches the destination. This tests if the notification logic is triggered properly.

Scenario-Based Testing:
We tested the app based on how a real user might interact with it in different situations. For example:
Scenario 1: We tested what happens when a user starts navigation, pauses it, and then resumes to make

sure the app keeps tracking the location correctly and doesn't mess up.
Scenario 2: We also tested how the app responds when the user quickly switches between locations, making sure the map updates smoothly and shows the correct location each time.

Assessment of the goodness of your test suite

We focused on these key criteria to measure the quality of our test cases:

1. Accuracy
   Each test case is designed with clear and specific expected results, avoiding any ambiguity.
   - Example: TC02 specifies that the map should display the user's current location, centered on the screen, after location permission is granted.
2. Cross-Platform Compatibility
   Each test case is aligned with specific app requirements, ensuring a clear link between the app's specifications and the corresponding tests.
   - Example: TC07 directly maps to the requirement for supervisors to export reports in PDF and CSV formats.
3. Reliability
   Tests are designed to consistently produce the same results under the same conditions, ensuring stable app performance.
   - Example: TC06 verifies that the map updates in real-time along the user's route, with updates occurring without delay, even across different network conditions and tests.

Traceability of Test Cases to Use Cases:

Use Case 1 (UC1): Customer Verifies Service Personnel's Identity

Design Mapping:

1. **UserInput Class**: validateUser() handles verification requests using ID
2. **LocationTracking Class**: getCurrentLocation() fetches real-time GPS data to validate personnel's location.
3. **NotificationService Class**: sendVerificationAlert(userId) sends notifications to the customer

4. **Test Case 1**: Verify that validateUser() in userInput successfully handles ID verification
   - **Requirements Addressed**:
     - Verified Personnel Identity Using ID
     - Security (Multi-factor authentication for verification)
   - **Non-Functional Requirements**: Security (end-to-end encryption), User Interface (intuitive for easy biometric/ID verification)
5. **Test Case 2**: Verify that sendVerificationAlert(userId) in NotificationService sends a notification to the customer confirming the personnel's identity.
   - **Requirements Addressed**:
     - Sends Notification to the Customer Confirming Personnel's Identity
   - **Non-Functional Requirements**: Compatibility (cross-platform notifications), Performance (notification delivery within 2 seconds)
6. **Test Case 3**: Confirm that getCurrentLocation() in LocationTracking accurately fetches real-time GPS data to verify personnel's location.

- **Requirements Addressed**:
  - Uses GPS to Validate Personnel's Location
- **Non-Functional Requirements**: Performance (location fetched within 2 seconds), Reliability (accurate GPS updates)

7. **Test Case 4**: Validate that data used in identity verification and notifications is secure.
   - **Requirements Addressed**:
     - Security (Implements end-to-end encryption)
   - **Non-Functional Requirements**: Security (data encryption for all personal data)

Use Case 2 (UC2): Supervisor Tracks Service Personnel in Real-Time

Design Mapping:

- **LocationTracking Class**: startTracking() initiates location tracking; getCurrentLocation() fetches updated locations.
- **LocationDAO Class**: saveLocation(Location) stores location data.
- **NotificationService Class**: sendArrivalNotificaiton(userId) sends alerts for deviations in route or arrivals.

1. **Test Case 5**: Verify that startTracking() in LocationTracking initiates GPS tracking successfully.
   - **Requirements Addressed**:
     - Tracks Personnel's Location via GPS
   - **Non-Functional Requirements**: Battery Optimization (efficient GPS use), Reliability (continuous tracking)
2. **Test Case 6**: Confirm that getCurrentLocation() in LocationTracking provides real-time location updates.
   - **Requirements Addressed**:
     - Tracks Personnel's Location in Real-Time
   - **Non-Functional Requirements**: Performance (location updates every 2 seconds), Scalability (handle multiple users)
3. **Test Case 7**: Verify that saveLocation(Location) in LocationDAO stores location data for future tracking and reporting.
   - **Requirements Addressed**:
     - Real-Time Location Tracking and Storage
   - **Non-Functional Requirements**: Data Handling (compliance with data retention laws), Reliability (consistent data storage)
4. **Test Case 8**: Confirm that sendArrivalNotification(userId) in NotificationService sends alerts for deviations or arrival events.
   - **Requirements Addressed**:
     - Sends Alerts for Deviations
   - **Non-Functional Requirements**: Compatibility (works on iOS and Android), Performance (alert within 2 seconds)

Use Case: UC3 - Supervisor Generates Service Report from the System

Requirements in UC3 and Corresponding Test Cases:

1. **Requirement:** Supervisor must be able to navigate to the reporting section.

    - **Mapped Test Case:** *TC01* - Ensures that location permissions are requested on startup, which is a prerequisite for tracking data that will be used in reporting.
2. **Requirement:** System compiles data related to location history and service completion times for a specific period.
    - **Mapped Test Case:** *TC02* - Confirms that the app centers the map on the current location, which ensures location data collection is accurate and can be used for reporting.
    - **Mapped Test Case:** *TC06* - Checks map responsiveness and accuracy to ensure data continuity for location history, critical for compiling reliable reports.
3. **Requirement:** Supervisor reviews and optionally exports the report as a PDF or CSV file.
    - **Mapped Test Case:** *TC07* - Verifies that the supervisor can export reports in both PDF and CSV formats, meeting the requirement for multiple export options.
4. **Requirement:** System should handle cases of insufficient data and notify the supervisor.
    - **Mapped Test Case:** *TC05* - Tests invalid address entry handling, which simulates scenarios of invalid or insufficient data, ensuring that the app provides an appropriate notification in these cases.
5. **Requirement:** System displays error messages on report generation failures and provides options to retry or select another time period.
    - **Mapped Test Case:** *TC05* - Also relevant here, as it verifies that error messages are correctly handled, simulating scenarios in which the system might need to notify the supervisor about unavailable reports.

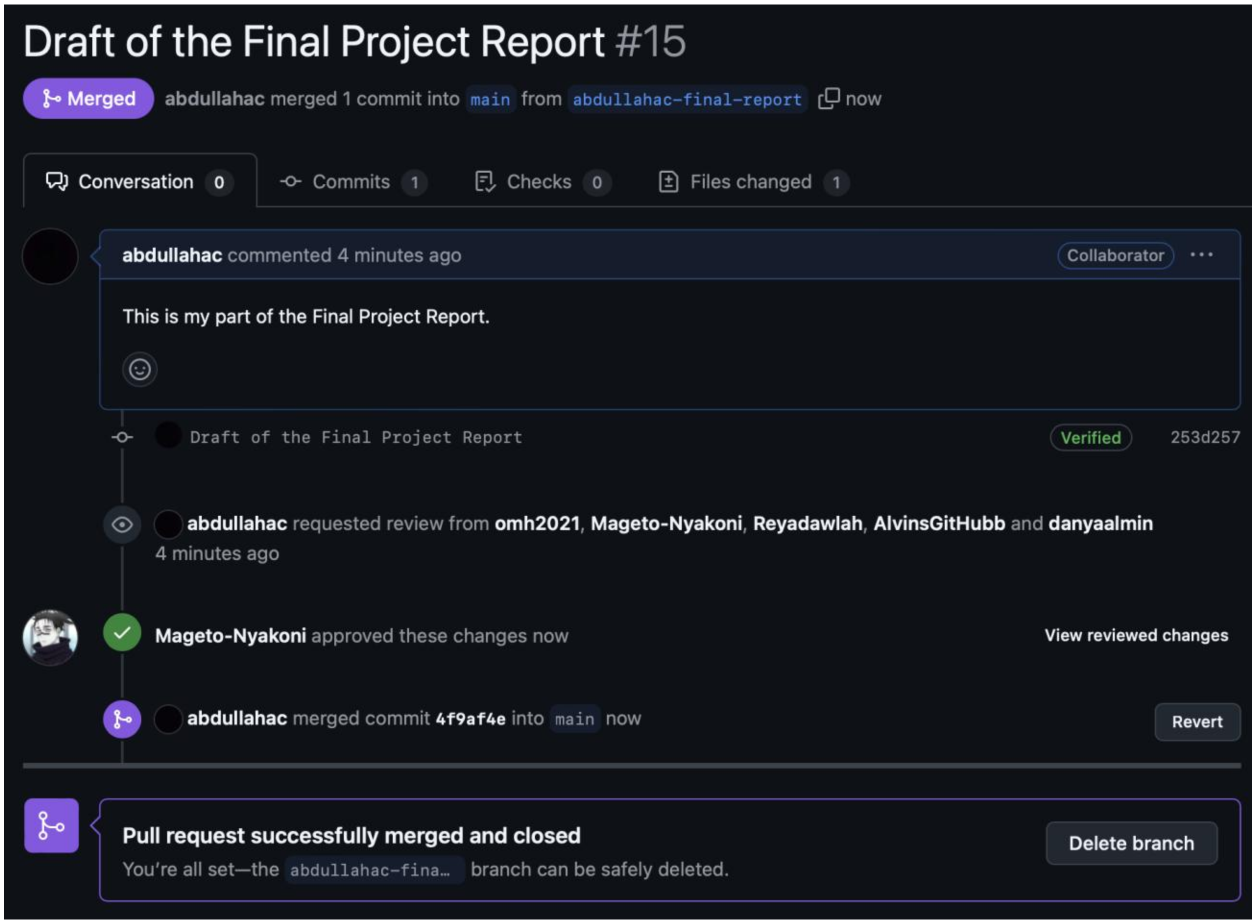
Use Case: UC4 - Service Personnel Downloads Software onto Customer Phone

Requirements in UC4 and Corresponding Test Cases:

1. **Requirement:** Service personnel's arrival is tracked, and the system confirms the arrival location.
    - **Mapped Test Case:** *TC04* - Specifically verifies arrival notifications, ensuring the app notifies when the service personnel reach the designated location. This aligns with the requirement that personnel's arrival location is recorded.
2. **Requirement:** System should detect nearby mobile devices.
    - **Mapped Test Case:** *TC08* - Ensures the system detects and lists nearby devices, enabling connection to the customer’s phone, which is necessary for the download process to begin.
3. **Requirement:** System downloads software onto the customer's phone.
    - **Mapped Test Case:** *TC09* - Verifies the complete software download process, ensuring that the app successfully transfers the software to the customer’s device, assuming necessary storage space and connectivity are available.
4. **Requirement:** Location is continuously tracked during the software download.
    - **Mapped Test Case:** *TC02* and *TC06* - Both of these test cases support continuous tracking. *TC02* confirms the initial location tracking, while *TC06* ensures real-time map updates, which are essential for continuous tracking during software download.
5. **Requirement:** Customers' devices must have enough storage space for the software.
    - **Mapped Test Case:** *TC09* - This test case verifies the software download process, specifically checking that the download only proceeds if there is sufficient storage on the customer’s device, fulfilling this requirement.

7. Evidence the Document Has Been Placed under Configuration Management

- This document along with the full source code and other documentation can be found here: https://github.com/AlvinsGitHubb/Location-Verification-App



Draft of the Final Project Report #15

Merged abdullahac merged 1 commit into `main` from `abdullahac-final-report` now

Conversation 0 · Commits 1 · Checks 0 · Files changed 1

abdullahac commented 4 minutes ago — Collaborator

This is my part of the Final Project Report.

`Draft of the Final Project Report` Verified 253d257

abdullahac requested review from omh2021, Mageto-Nyakoni, Reyadawlah, AlvinsGitHubb and danyaalmin 4 minutes ago

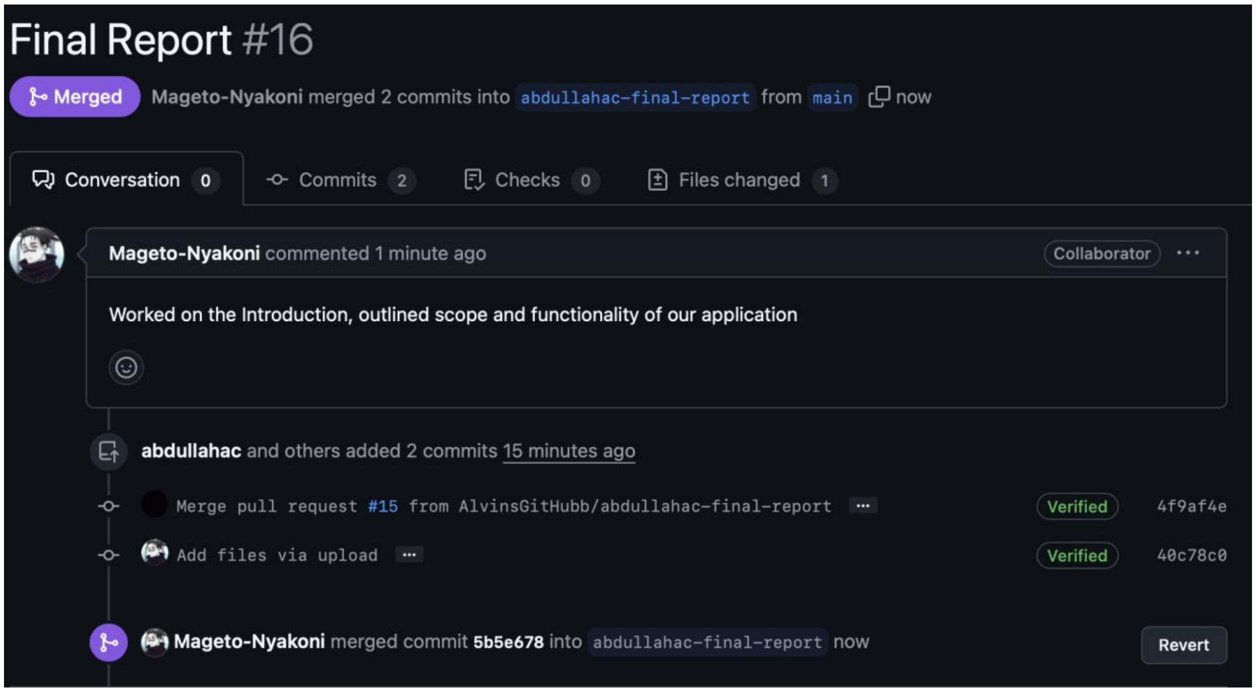
Mageto-Nyakoni approved these changes now — View reviewed changes

abdullahac merged commit `4f9af4e` into `main` now — Revert

Pull request successfully merged and closed

You're all set—the `abdullahac-fina…` branch can be safely deleted. — Delete branch

Figure 10: Abdullah Created the Pull Request for the Final Report initial draft, Approved by Mageto


Final Report #16

Merged Mageto-Nyakoni merged 2 commits into `abdullahac-final-report` from `main` now

Mageto-Nyakoni commented 1 minute ago — Collaborator

Worked on the Introduction, outlined scope and functionality of our application

abdullahac and others added 2 commits 15 minutes ago

```
Merge pull request #15 from AlvinsGitHubb/abdullahac-final-report   Verified   4f9af4e
Add files via upload                                                  Verified   40c78c0
```

Mageto-Nyakoni merged commit 5b5e678 into `abdullahac-final-report` now

Figure 11: Mageto created a pull request to merge his changes.

8. Engineering Standards and Multiple Constraints

- IEEE Std 1058-1998 for Software Project Management Plans
- IEEE Std 830-1998 for Software Requirements
- IEEE Std 1471-2000 for Software Architecture
- IEEE Std 1016-1998 for Software Design
- IEEE Std 829-1983 for Software Testing
- PMBOK® Guide: Project Management Body of Knowledge
- IEEE Std 12207: Software Life Cycle Processes
- IEEE Std 15939: Measurement Process
- ISO/IEC/IEEE Std 29148-2018: Systems and Software Engineering
    - Life Cycle Processes
    - Requirements Engineering
- IEEE Std 830-1998: Software Requirements
- IEEE Std 29148: Requirements Engineering
- ISO/IEC/IEEE Std 29148-2018: Systems and Software Engineering
    - Life Cycle Processes
    - Requirements Engineering
- IEEE Std 1471-2000: Software Architecture
- ISO/IEC/IEEE Std 42030:2019: Software, Systems and Enterprise
    - Architecture Evaluation Framework
- IEEE Std 1016-1998-(Revision-2009): Software Design
- IEEE Std 829-1983: Software Testing
- ISO/IEC/IEEE Std 29119-1-(Revision-2022): Part 1 - Software Testing General Concepts
- ISO/IEC/IEEE Std 29119-2-(Revision-2021): Part 2 - Test Process
- ISO/IEC/IEEE Std 29119-3-(Revision-2021): Part 3 - Test Documentation

- ISO/IEC/IEEE Std 29119-4-(Revision-2021): Part 4 - Test Techniques

9. Additional References

- Larson, E. and Gray, C., 2014. Project Management: The Managerial Process. McGraw Hill
- Humphrey, W.S. and Thomas, W.R., 2010. Reflections on Management: How to Manage Your Software Projects, Your Teams, Your Boss, and Yourself. Pearson Education
- Lamsweerde, A.V., 2009. Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley
- Lattanze, A.J., 2008. CRC Press
- IEEE CS Guide To The Software Engineering Body Of Knowledge v4.0
- Bass, L., Clements, P. and Kazman, R., 2003. Software Architecture in Practice. Addison-Wesley
- 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
- Jorgensen, P.C., 2013. Software Testing: A Craftsman's Approach. Auerbach Publications
- Mathur, A.P., 2013. Foundations of Software Testing, 2/e. Pearson Education

Acknowledgment

We extend our sincere gratitude to our stakeholder for his guidance, support, and valuable feedback throughout the development of this project.