COMP 3059 – Capstone Project I

Software Requirements Analysis and Design Assignment

Workspace: https://wrenchit.atlassian.net/

1.0 Introduction

The Software Requirements Analysis and Design document outlines the functional, non-functional, and technical foundations for WrenchIT; including the design outline of the final application and the issues it looks to solve.

1.1 Purpose

The purpose of this document is to provide a clear technical description of the WrenchIT platform in relation to the real world issues it looks to solve. This document uses the identified primary user groups and stakeholders, and defines the functional and non-functional requirements of our system in relation to proposed requirements.

Additionally, this document provides diagrams for high-level data models, specific class structure, and system interactions. This document ensures clarity of the technical objectives and requirements of our project, and what is expected from the development team, stakeholders, and users in our overall design.

1.2 Scope

WrenchIT is a web application that helps users find and evaluate auto repair services. It allows customers to search for nearby repair shops, compare prices and services, read and submit mechanic-verified feedback, write reviews, upload receipts, and book services. A future version may also include roadside assistance and other features.

The platform gives users clearer information when choosing repair services. It also provides shops and mechanics with simple tools to manage bookings, publish service details, and interact with customers. By bringing together shop data and user contributions in one place, WrenchIT supports better decision-making and encourages fair, competitive service among repair providers.

2.0 System Overview

This section describes the high-level architecture, and technical considerations to provide a cohesive understanding of how the core WrenchIT system operates and interacts with all necessary components.

2.1 Project Perspective

WrenchIT is designed to operate as independently as possible but still requires several external services to ensure accurate functionality. The primary dependencies include mapping and geolocation services such as the Google Maps API. The system will also use limited third-party business information including basic shop details, while still relying on user-submitted reviews, receipts, and service records.

In terms of infrastructure, our long-term goal is to keep WrenchIT cloud neutral and deployable across a range of hosting providers using open-source and cost-free technologies. However, certain external integrations (e.g S3 bucket connectors, AWS RDS connectors, identity provider connectors, and basic email or notification services) will remain necessary especially for initial builds.

One important consideration in building our platform is its contrast to similar services such as Yelp or Google Reviews. While those platforms focus on broad community feedback and general interaction, WrenchIT is built specifically for automotive services and repairs. It will showcase a structured comparison of repair experiences, a transparent review process, and simple guidance toward reliable local service providers. This ensures that the information presented to users is accurate, consistent, trustworthy, and thus useful.

2.2 System Context

The auto repair industry lacks transparency which leaves customers without clear information about pricing, service quality, and the reliability of individual mechanics and shops. Prices vary between locations, and the lack of one source of reliable information for services makes it difficult for customers to compare options or make informed decisions.

WrenchIT offers a centralized platform where users can search for shops, compare services, submit and read reviews, and book appointments. Shop owners will manage their listings and pricing while users upload service reviews and data. The platform will then bring together this accurate shop data and user feedback, to give both customers and shop owners a clearer view of the local repair market.

2.3 General Constraints

WrenchIT must operate within several technological, operational, and data-related constraints. As a web-based system, it requires a framework such as Spring Boot and a relational database like PostgreSQL and a comparable cloud-managed option.

The platform relies on external APIs such as Google Maps for essential features like geolocation and shop discovery. Operational needs include version control and CI/CD pipelines through GitHub and Jenkins, and the cloud based services of AWS EC2 and RDS.

From a data standpoint the constraints will be from the limited accuracy of information provided by repair shops, the reliability of third-party API data, and the need for data collection via users submitting valid receipts and review content.

2.4 Assumptions and Dependencies

WrenchIT is based on several key assumptions and dependencies that shape the system's design and functionality. The platform assumes that users will submit accurate receipts and reviews, that repair shops will keep their service information current, and that third-party APIs will remain available throughout the system's operation.

It also expects users to have stable internet access and a compatible browser. The system depends on external components such as the Google Maps API for location and shop identification, cloud services for hosting, storage, and backups, and secure authentication/authorization frameworks like Spring Security. Front-end development

relies on technologies such as React, and TailwindCSS. Thus feature timelines may be influenced by the availability of data from local shops and the time required for development.

3.0 Functional Requirements

This section covers the core functional requirements of the WrenchIT platform.

3.0.1 Shop Search and Discovery

Description:
Users must be able to search for repair shops based on name, location, rating, available services, and distance.
Inputs:
Search term, user location, service filters, distance filter.
Processing:
The system queries its database for matching shops, applies filters, and retrieves distance data from Google Maps. Redis may respond if the data is cached.
Outputs:
A ranked list of relevant shops that includes service offerings, prices, ratings, and distance.

3.0.2 Service and Price Comparison

Description:
Users can compare service offerings across multiple shops to support informed decision-making.
Inputs:
Selected shop IDs or selected service categories.
Processing:
The system retrieves SHOP_SERVICE data, calculates price differences, and aggregates reviews tied to each shop and service.
Outputs:
A comparison table showing prices, estimated service duration, ratings, and review counts.

3.0.3 Ratings and Review Submission

Description:
Clients can submit reviews for completed services, including rating, written feedback, and optional receipt upload.
Inputs:
Rating value, text comments, visit date, images or receipt file.
Processing:
The system stores the review, links it to the shop and user, and assigns a review status of Unverified or Pending Verification. Receipt files are uploaded to storage and linked to the review.
Outputs:
A posted review with a status label displayed on the shop's profile.

3.0.4 Mechanic Verification Workflow

Description:
Mechanics can confirm the authenticity of submitted receipts and the accuracy of reported service details.
Inputs:
Mechanic credentials, selected review ID, verification decision, optional comments.
Processing:
The system records the mechanic's decision in a VERIFICATION_RECORD entry and updates the review status to Verified or Rejected.
Outputs:
A verified or rejected review displayed on the shop profile, along with verification metadata.

3.0.5 Shop Management Portal

Description:
Shop owners can maintain their shop profiles by updating services, pricing, hours, and general information.
Inputs:
Updated listing details, new service items, price changes.
Processing:
The system validates inputs, saves changes in SHOP or SHOP_SERVICE tables, and clears related cached entries.
Outputs:
Updated shop pages and search results reflecting the changes.

3.0.6 Booking System

Description:
Users can book appointments for listed services directly through the platform.
Inputs:
Service selection, date and time, user ID.
Processing:
The system stores the booking, checks availability, and marks the booking as Pending, Confirmed, or Completed.
Outputs:
Confirmation screen, booking ID, and status updates in the user's dashboard.

3.0.7 Role-Based Access Control

Description:
The system supports four user roles: Client, Mechanic, Shop Owner, and Admin. Each role exposes specific features.
Inputs:
User credentials at login, registration details.
Processing:
The system authenticates users through Spring Security and grants access based on USER_ROLE mappings.
Outputs:
Dashboard and navigation tailored to the user's assigned role.

3.1 Use Cases

Use cases reflect how actors interact with the system. Each use case aligns with one or more functional requirements.

3.1.1 Search for Nearby Shops

User: Client
Goal: Find shops that match user needs.
Trigger: User enters a keyword, filter, or location.
Main Steps:

1. The user enters search criteria.
2. The system queries its data and calls Google Maps for distance.
3. Results return with distance, pricing, and ratings.

Outcome:
A sorted list of shops displayed on the search page.

3.1.2 Submit a Review

User: Client
Goal: Provide feedback after a completed service.
Trigger: User selects “Write Review” on a shop profile.
Main Steps:

1. User enters rating, comments, and uploads a receipt if available.
2. The system validates and stores the review.
3. If a receipt is uploaded, review status becomes Pending Verification.

Outcome:
Review is posted with a clear verification status.

3.1.3 Validate a Receipt

User: Mechanic
Goal: Verify if a submitted receipt and service details are legitimate.
Trigger: Mechanic opens the “Pending Verifications” page.
Main Steps:

1. Mechanic selects a review.
2. The mechanic approves or rejects the receipt.
3. The system records the verification and updates the review status.

Outcome:
Review becomes Verified or Rejected.

3.1.4 Manage Shop Listing

User: Shop Owner
Goal: Update and maintain shop information.
Trigger: Shop owner opens the “Manage Shop” dashboard.
Main Steps:

1. Shop owner updates prices, available services, or hours.
2. The system validates and saves the changes.
3. Search results and shop pages are updated.

Outcome:
Accurate, current shop information is visible to users.

3.1.5 Book a Service Appointment

User: Client
Goal: Schedule a service with a chosen shop.
Trigger: User selects a service from a shop's profile.
Main Steps:

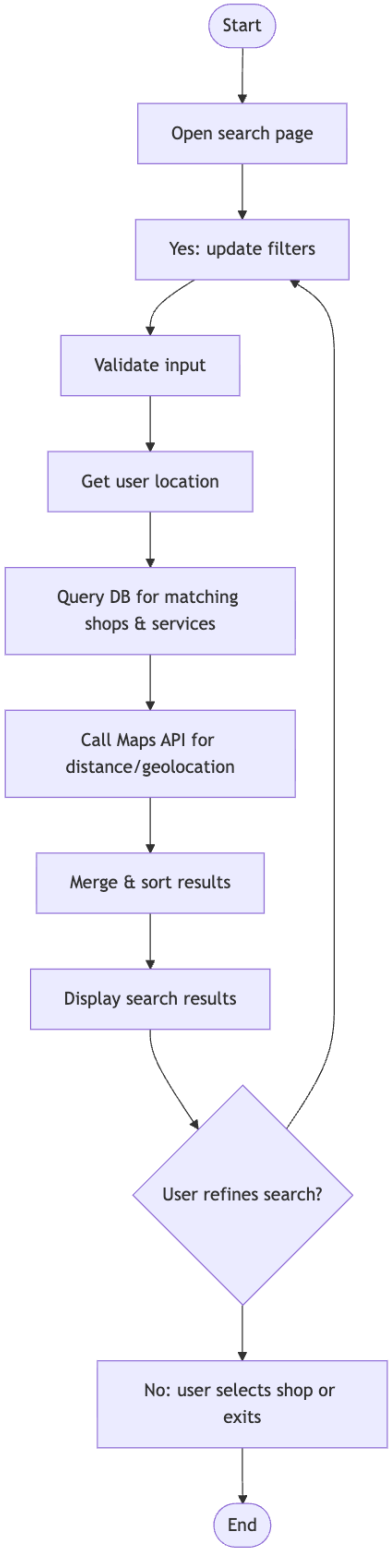
1. The user chooses a date and time.
2. The system stores the booking and sets its status.
3. User receives confirmation.

Outcome:
A confirmed booking appears in the user's account.

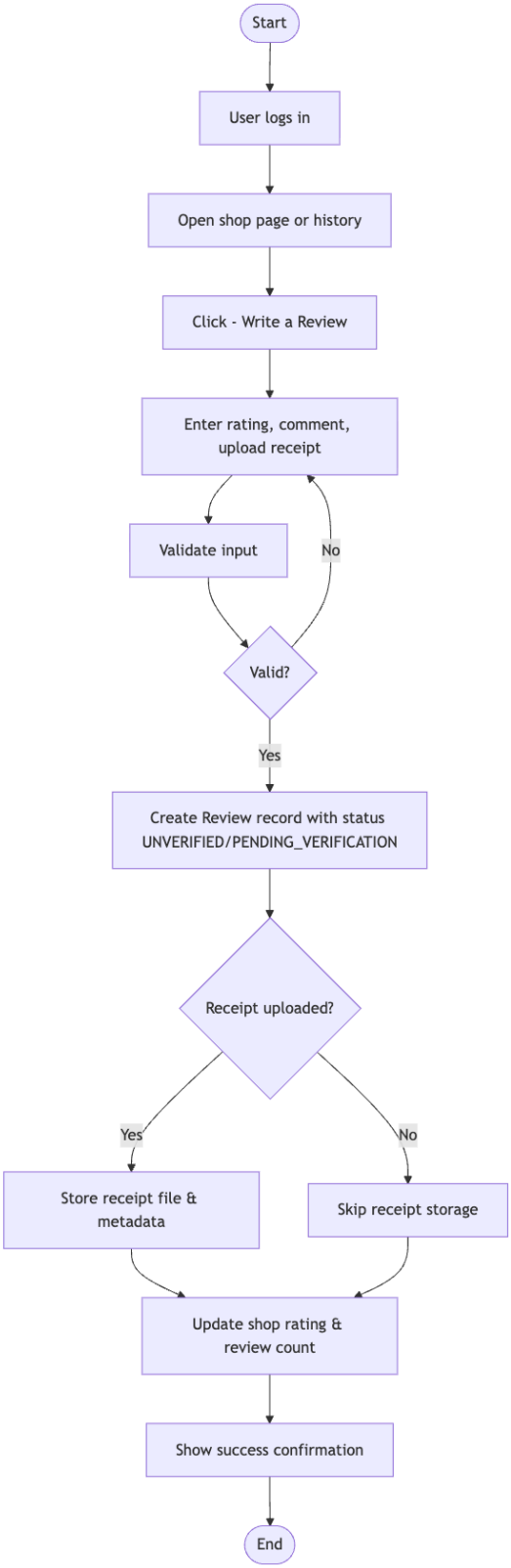
3.3 Data Modelling and Analysis

Ss1 - Normalized Data Model Diagram

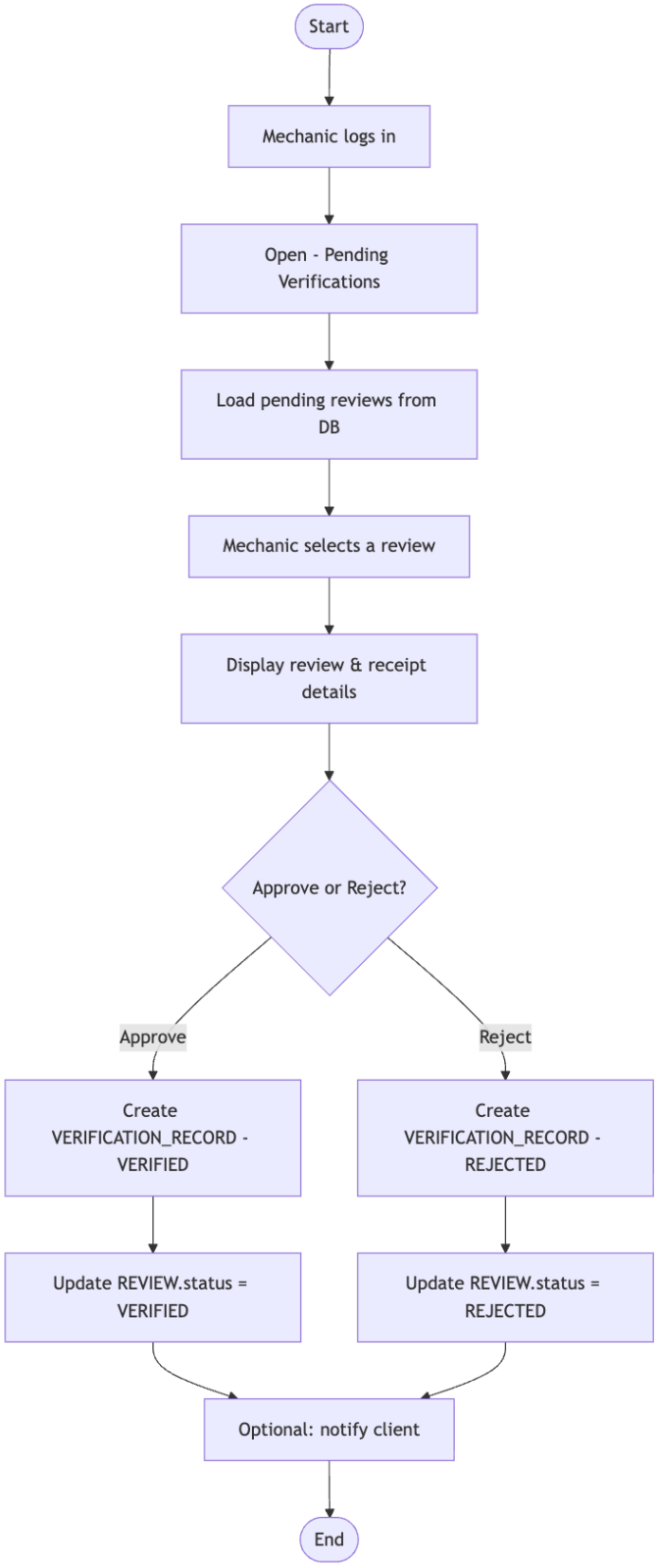
Activity Diagrams Ss1



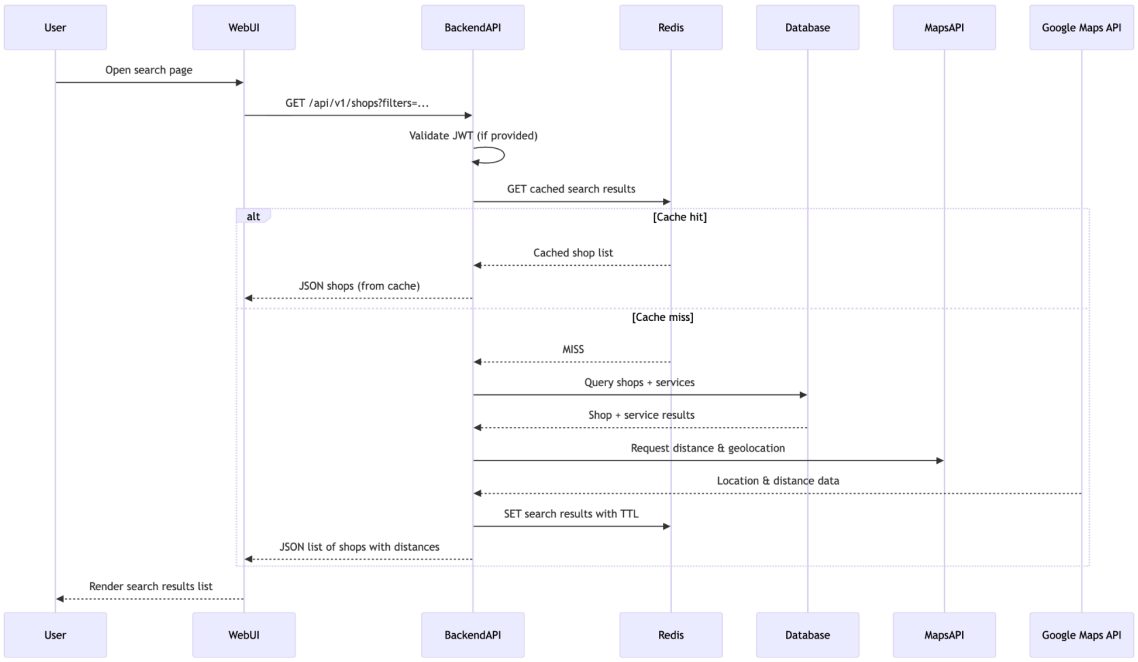
Ss2


Start
User logs in
Open shop page or history
Click - Write a Review
Enter rating, comment, upload receipt
Validate input
No
Valid?
Yes
Create Review record with status UNVERIFIED/PENDING_VERIFICATION
Receipt uploaded?
Yes
No
Store receipt file & metadata
Skip receipt storage
Update shop rating & review count
Show success confirmation
End

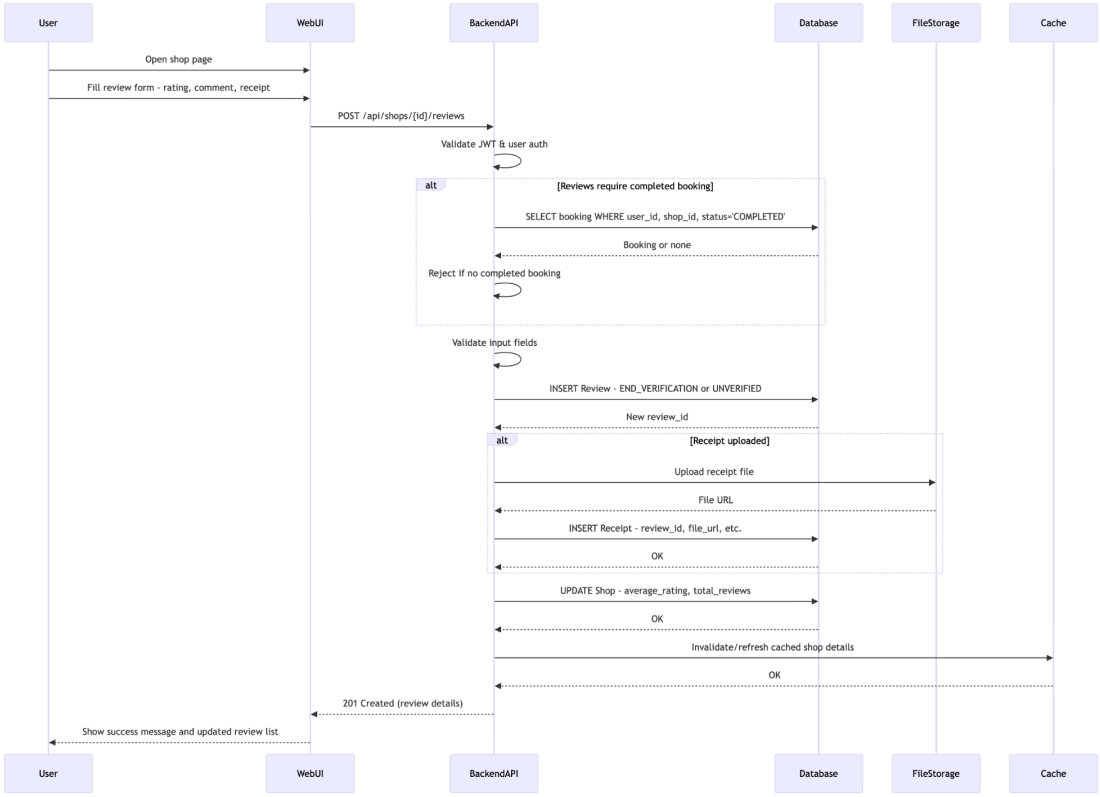
Ss3


Start
Mechanic logs in
Open - Pending Verifications
Load pending reviews from DB
Mechanic selects a review
Display review & receipt details
Approve or Reject?
Approve
Reject
Create VERIFICATION_RECORD - VERIFIED
Create VERIFICATION_RECORD - REJECTED
Update REVIEW.status = VERIFIED
Update REVIEW.status = REJECTED
Optional: notify client
End

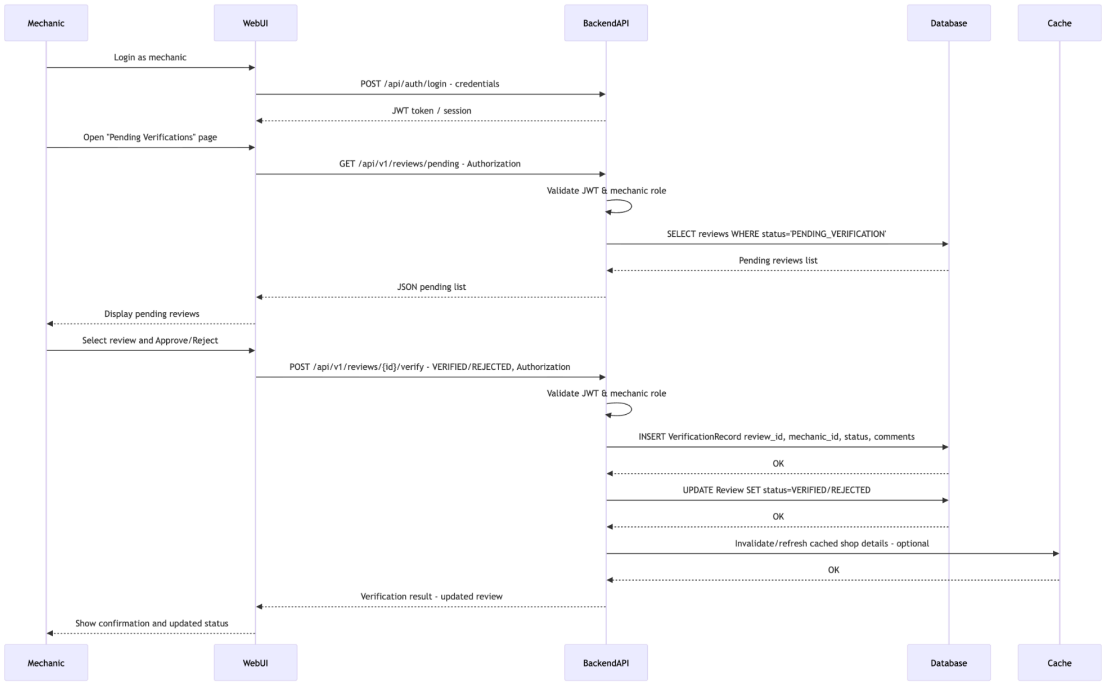
Sequence Diagrams
S1



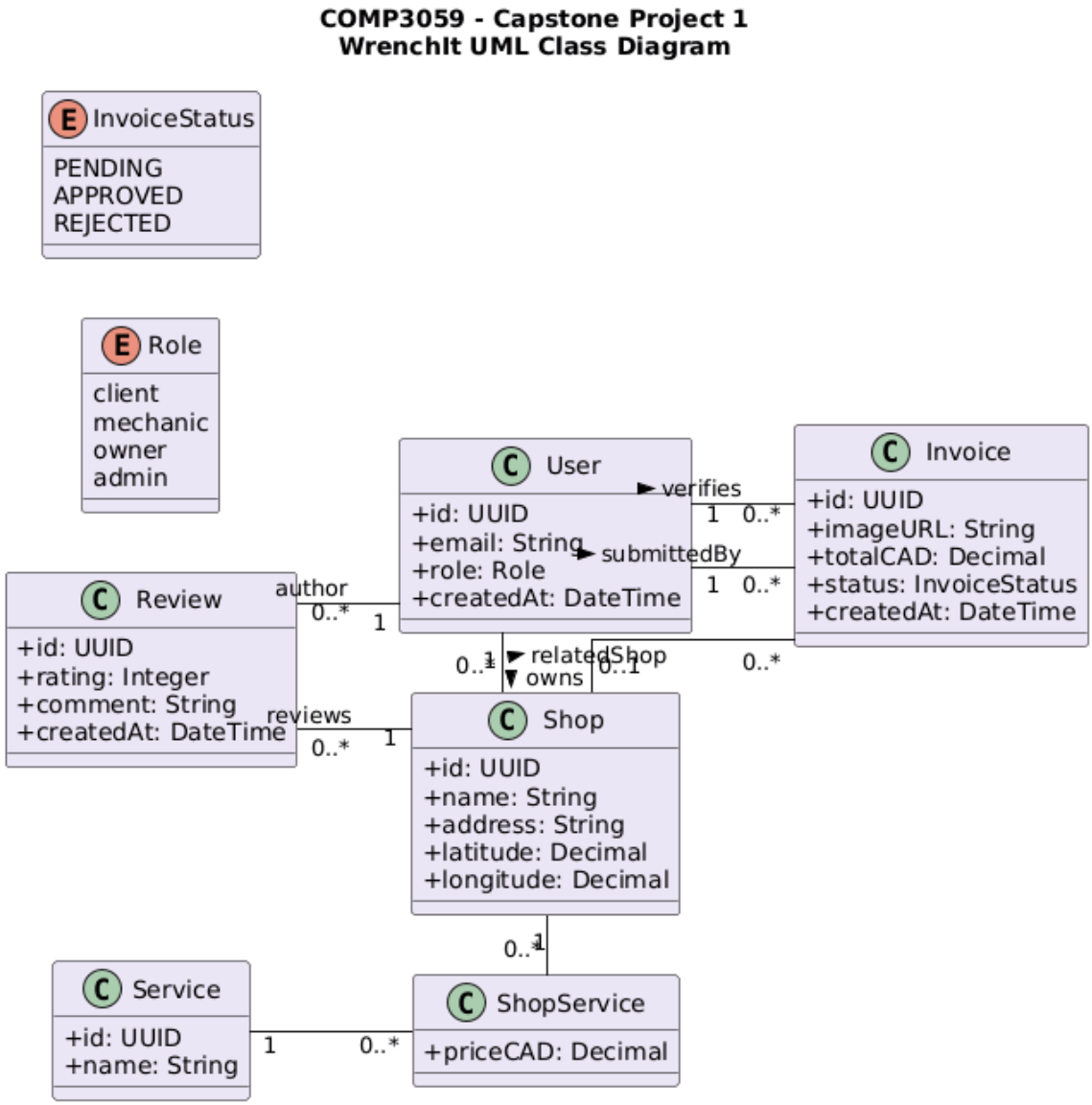
S2


User
WebUI
BackendAPI
Database
FileStorage
Cache
Open shop page
Fill review form - rating, comment, receipt
POST /api/shops/{id}/reviews
Validate JWT & user auth
alt
[Reviews require completed booking]
SELECT booking WHERE user_id, shop_id, status='COMPLETED'
Booking or none
Reject if no completed booking
Validate input fields
INSERT Review - END_VERIFICATION or UNVERIFIED
New review_id
alt
[Receipt uploaded]
Upload receipt file
File URL
INSERT Receipt - review_id, file_url, etc.
OK
UPDATE Shop - average_rating, total_reviews
OK
Invalidate/refresh cached shop details
OK
201 Created (review details)
Show success message and updated review list
User
WebUI
BackendAPI
Database
FileStorage
Cache

S3

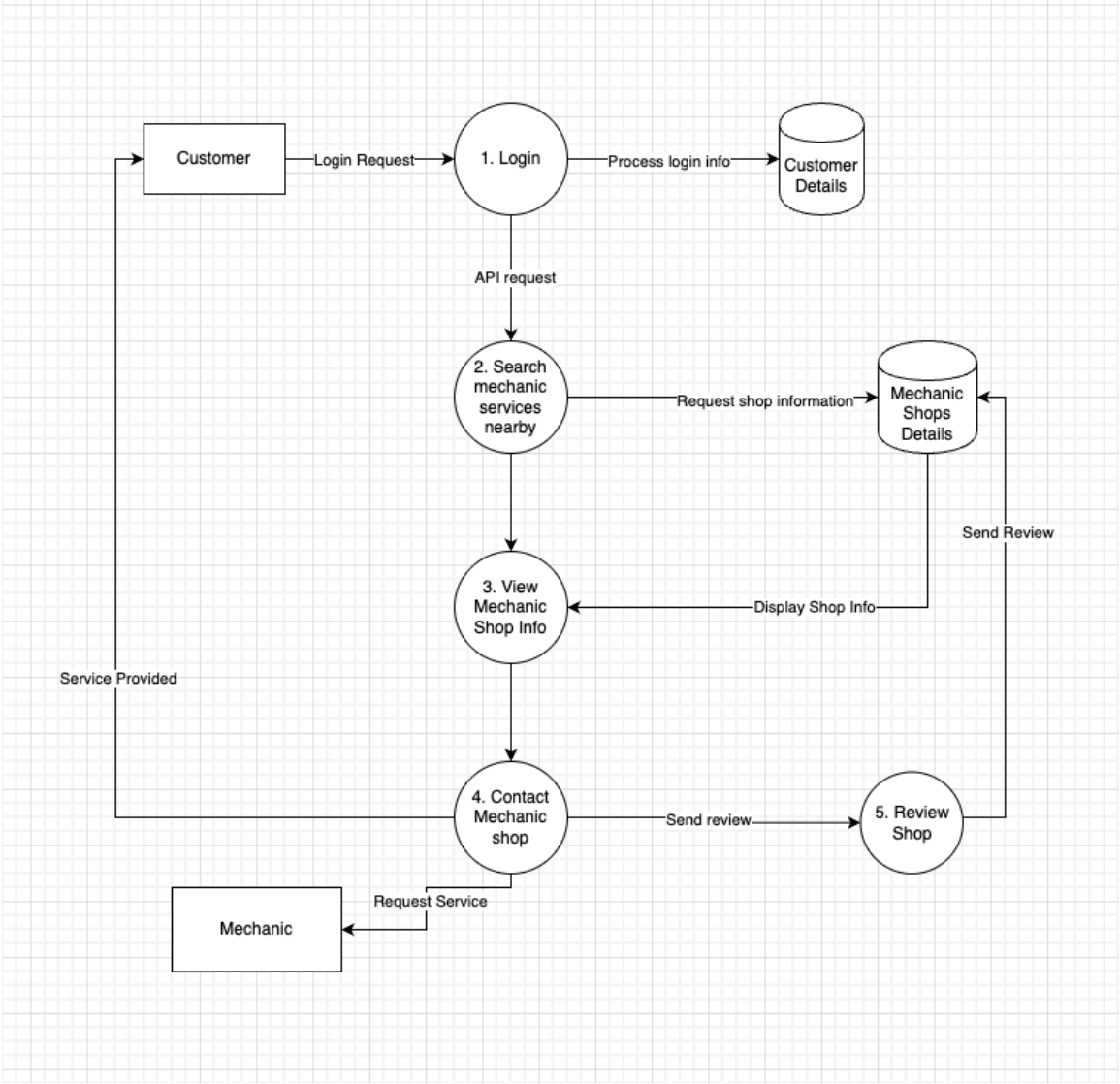

Mechanic
WebUI
BackendAPI
Database
Cache
Login as mechanic
POST /api/auth/login - credentials
JWT token / session
Open "Pending Verifications" page
GET /api/v1/reviews/pending - Authorization
Validate JWT & mechanic role
SELECT reviews WHERE status='PENDING_VERIFICATION'
Pending reviews list
JSON pending list
Display pending reviews
Select review and Approve/Reject
POST /api/v1/reviews/{id}/verify - VERIFIED/REJECTED, Authorization
Validate JWT & mechanic role
INSERT VerificationRecord review_id, mechanic_id, status, comments
OK
UPDATE Review SET status=VERIFIED/REJECTED
OK
Invalidate/refresh cached shop details - optional
OK
Verification result - updated review
Show confirmation and updated status
Mechanic
WebUI
BackendAPI
Database
Cache

UML Class Diagram



3.4 Process Modelling

Data Flow Diagram



4.0 Non-Functional Requirements

Non-functional requirements may exist for any of the following attributes – Performance, Reliability, Availability, Security, Maintainability, Portability.

Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transactions shall be processed in less than a second, system downtime may not exceed 1 minute per day, etc).

The non-functional requirements describe how the system should operate and support the overall user experience. These requirements cover performance, reliability, availability, security, maintainability, and portability. Each item is stated in clear and measurable terms so it can be tested during development.

Performance

1. Search results should load within 2 seconds under normal use.
2. API requests should respond in about 500 ms on average.
3. The system should handle at least 500 users during busy periods.

Reliability

1. Core features should maintain 99.5% uptime each month.
2. When possible, failed operations—such as API calls—should retry automatically.

Availability

1. The system should remain available during regular business hours.
2. Backups must run daily and store data for 30 days.

Security

1. Authentication and authorization must follow role-based access control using Spring Security.
2. Sensitive data (passwords, receipts, uploaded files) must be encrypted in transit and at rest.
3. JWT tokens must be checked on every request.
4. Only mechanics and admins with proper roles can verify receipts or edit shop records.

Portability

1. The system should be deployable on cloud platforms such as AWS or Supabase.
2. The front end and back end should run independently so they can be hosted in different environments if needed.

5.0 Logical Database Requirements

The system uses a PostgreSQL relational database hosted on AWS RDS. The database must follow the normalized structure defined in the Data Model to ensure consistency and clear relationships between users, shops, services, bookings, reviews, receipts, and verification records.

Data Structure

1. All tables must follow the normalized schema shown in the data model.
2. Timestamps should use a standard format (ISO/UTC) for consistency.
3. Receipt files must be stored as URL paths pointing to cloud storage for security.

Storage and Scalability

1. The database should support growing data from reviews, receipts, and shop listings.
2. Indexes should be created on fields that are searched often, such as shop_id, user_id, and service_category_id.

Data Integrity.

1. Primary keys must be unique and auto-generated.
2. Foreign key constraints must enforce valid relationships between tables.
3. Review verification status must follow the defined ENUM values to keep the data consistent.

Data Retention

1. User and shop records should be kept indefinitely unless removed for compliance reasons.
2. Older receipt data may be archived after one year to save space, as long as verification logs remain available.

6.0 Other Requirements

The system should include an admin interface for managing user roles, shop listings, flagged reviews, and overall system health.

Logging must record system errors, failed login attempts, and unexpected API behaviour to support debugging and security checks.

The front-end should follow basic accessibility guidelines (WCAG 2.1) so the system is usable by a wider range of users.

Future features should be designed in a modular way to support scalability and allow the project to expand into microservices if needed.

7.0 Approval

The signatures below indicate their approval of the contents of this document.

Project Role	Name	Signature	Date
Lead Developer	Henrique Custodio	*Henrique Custodio*	Nov 16 2025
Developer	Hamza Hafez	*Hamza Hafez*	Nov 16 2025
Developer	Tyson Ward-Dicks	*Tyson Ward-Dicks*	Nov 16 2025
UI Developer	Hossein Khanzadeh	*Hossein Khanzadeh*	Nov 16 2025
Database Manager	Frederich Tan	*Frederich Tan*	Nov 16 2025