SmartMove

Lab Assignment 1

The SmartMove Core Engine is a single backend system that runs thousands of shared micro-mobility vehicles, such as bicycles, electric scooters, and mopeds, in several European cities. This system has to handle intricate business rules, real-time hardware integration, following various rules in different places, and full auditability, all while working within stringent technical restrictions like file-based persistence and manual concurrency control.

A. Role assignments

Danijel Miokovic: Full System Implementation Developer

- Implementing the vehicle state machine and transition validation logic.
- Developing SmartMoveCentralController and rental workflow.
- Implementing multi-tenant regulatory rules (London, Milan, Rome).
- Processing asynchronous telemetry updates.
- Implementing manual concurrency control using primitive synchronization techniques.
- Implementing the high-integrity audit log with sequential ID and checksum chaining.
- Ensuring system consistency and manual rollback behavior on write failure.

Balsa Rakocevic: Quality Assurance and Static Analysis Engineer

- Setting up and running SonarQube static analysis
- Generating and documenting the SonarQube report.
- Ensuring a minimum 40% code coverage.
- Running and evaluating unit tests.
- Identifying refactoring opportunities based on analysis results.
- Verifying alignment between design and implementation quality.

Tariq Bhatti: Requirements Analyst and Documentation

- Identifying and documenting all functional requirements
- Identifying and documenting non-functional requirements
- Structuring requirements clearly in the project documentation.
- Ensuring traceability between requirements, UML design, and implementation.

Duy Dao: UML System Architecture Designer

- Translating functional and non-functional requirements into a formal object-oriented design.
- Designing the SmartMove domain mode
- Ensuring modeling correction
- Delivering the .asta class diagram file

B. Descriptive analysis of the functional

Functional requirements define the specific behaviors, features, and capabilities the SmartMove Core Engine must deliver.

1. Vehicle State Management

A clear state machine must be in charge of the whole life cycle of each vehicle in the system. SmartMove needs six separate states, and there are tight rules about how to move between them. This is different from ordinary rental systems, which only switch between "available" and "rented."

Vehicle States:

The automobile is ready to rent, parked legally, and works flawlessly.

Reserved: The user has booked the vehicle but hasn't started the trip yet (limited time).

InUse: A rental is now going on.

Maintenance: The vehicle has to be serviced because the battery is low, there is a mechanical problem, or it is damaged.

EmergencyLock: The vehicle is forcibly deactivated because it was stolen or because it was unsafe.

Moving—moving the vehicle to rebalance it by the operations crew.

The SmartMoveCentralController has to check every transition that is tried against the current state and the conditions that are happening right now. For instance:

Rules for Checking State-to-State

Available to reserve:- To use the vehicle, the user must be logged in, and the vehicle must be in a geofenced area.

Reserved to Inuse:-Milan needs to pass a helmet inspection, the battery needs to be at least 15%, and there can't be any problems.

InUse to emergency lock: If the temperature gets beyond 60°C or there is a theft, InUse EmergencyLock will turn on by itself.

Any state of emergency:- Any maintenance of the state. Only if the telemetry displays a problem or the operator starts it

2. Domain for regulatory compliance with several tenants

The engine needs to be able to run activities in several European locations at the same time, each with its own set of legal and financial rules. This isn't simply a set of configuration flags. It's a complicated rules engine that applies logic based on where each transaction takes place.

Requirements for cities:

London operations

- Cars that enter charging zones must pay a "congestion charge" on top of what they have paid.

Milan Operations

- Sensors check to see if you're wearing a helmet before you may rent a moped.
- The vehicles will be locked up if they go where they aren't supposed to.

Rome Operations

- Different rules for helmets (just for scooters, not bikes).

3. Processing Real-Time Telemetry

The system isn't passive; it has to actively process a constant flood of telemetry data from thousands of cars. Every car transmits updates every now and then that include:

- Coordinates in GPS (latitude, longitude, and accuracy)
- Percentage of battery life (0–100%)
- Temperature of internal parts (Celsius)
- Status of the motor (active, idle, or faulty)
- State of the lock (engaged or disengaged)
- Data from the accelerometer (for crash detection)
- Speed (now and then)
- Codes for hardware problems

Requirements for processing:

The system must be able to handle:

- Peak throughput: about 2,000 messages per second
- Average throughput: about 500 messages per second
- Message size: about 200 bytes in binary or about 500 bytes in JSON

Keeping an eye on safety:

The SafetyMonitor needs to check each telemetry update against important thresholds:

Overheating: If the temperature is over 60°C, it will slow down and enter emergency lock if it keeps happening

Battery: If the battery is less than 5% during the journey, as a result, throttle speed, let the user know, and get ready to end

Theft detection: If movement is found while available, then emergencyLock must apply right away and alert operations

Detecting Crashes: If there is an accelerometer problem then lock the phone, call emergency services, and tell the user.

Zone Violation: if going into a restricted location, it must stop slowly, and if it keeps happening, apply a fine.

All safety measures must be recorded with full audit trails and must not get in the way of other rental operations going on at the same time.

4. Concurrency Management

The system has to handle concurrent access without corruption with 10,000 vehicles and several ways for the state to change (telemetry updates, user requests, operator actions). Because of the limit on high-level frameworks, manual synchronization must be used.

Telemetry Update + Rental Command: A user is trying to initiate a rental on a vehicle at the same time that the vehicle's telemetry comes in.

Multiple Telemetry Updates: Two telemetry packets for the same vehicle arrive at the same time

Safety Intervention + User Command: The emergency lock goes off when the user is trying to end the rental.

Operator Action + Telemetry: While telemetry is being updated, field staff mark the vehicle for maintenance.

Plan for Locking:

There are 256 locks for 10,000 automobiles in the system, which uses a striped locking method:

This gives:

- Fine-grained concurrency (only cars that share a stripe block each other)
- Less competition (only about 39 cars per lock on average)
- No deadlocks (one lock per operation, no problems with lock ordering)

Lock Duration: Locks are only kept during state changes and important reads, never during I/O operations. This keeps things consistent while reducing disagreement.

5. Persistence and Audit Trail

- Before any changes to the state are saved to the main file. This makes sure that you can get back up after a crash:
- Write down the change you want to make to a temporary file.
- Change the main state file (atomic operation)
- If a crash happens during step 2, the recovery process uses a temporary file to finish or go back.

The Audit Trail's Integrity:

The audit log uses cryptographic checksum chaining to prevent anybody from messing with it:

- There is a number for each entry that goes in order (1, 2, 3, etc.).
- The entry has a SHA-256 hash that is made up of the hash from the last entry and the data from the current item.
- You can look at the chain at any time.

6. Recovery and Rollback

If file system writes fail, the system has to keep the in-memory state and the persistent data in sync by allowing users undo changes themselves.

If something goes wrong, do this:

- Write Failure: The disk is full, or there is a permissions problem while trying to save the state.
- Crash During Write: The power goes off after writing to the temporary file but before the main file is updated.
- Finding corruption: When the system starts up, checksum checking fails.
- Partial Write: The system stopped working while it was writing.

Ways to Get Back:

- Checkpoint System: It takes photographs of the state in memory and saves them to files that can be used to recover the system.
- Temporary file Replay: When you start up, the temporary file will replay any operations that are still open.
- Manual Rollback is an admin tool that lets you go back to the last known good state using the audit trail.
- Integrity Check: A scan of all files at startup that checks their checksums.

C. Descriptive analysis of the non-functional

Non-functional requirements set the system's quality standards and limits, such as its performance criteria, reliability expectations, and operational circumstances.

1. Performance

In terms of performance, the telemetry latency is less than 100 milliseconds, the state transition is less than 50 milliseconds, and there are 10 thousand vehicles operating simultaneously.

2. Reliability

99.9% uptime, zero data loss, and 100% audit integrity are all examples of reliability.

3. Scalability

In less time, add a new city, and sustain10 thousand vehicles with sharded telemetry.

4. Constraints

Constraints include the absence of high-level concurrency frameworks (only synchronized, Lock, and wait/notify), as well as the absence of external databases (only CSV and JSON).

5. Security

Data separation that is consistent with the General Data Protection Regulation (GDPR) and access for regulators that is dependent on roles.

6. Testability

Testability is defined as a unit test coverage of more than 80 percent for both the state machine and the rule engine.