S26CS6.401 - Software Engineering

Take Home Activity - 1

Smart vehicle booking system

Due: 23 Jan 2026, 11:59 PM

==========

For this task, each team will be tasked with crafting a high-level design for a software system. The design will be visualized through a single UML diagram, showcasing the comprehensive class structure of the system. This diagram should encompass class names, primary responsibilities, and the connections between classes, including aspects like inheritance, association, aggregation, and potentially details like cardinality and role names. A dedicated section of the document is required to provide concise descriptions outlining the responsibilities assigned to each class. The diligence and clarity exercised in creating this high-level design will significantly influence the success of the task.

==========

Problem Description:

We anticipate the development of a software system for reserving smart bikes intended for on-campus transportation at IIIT-H. The design should support users in booking smart bikes via a mobile app and conducting payment transactions. Users should be able to complete the onboarding process, scan QR codes on bikes located in the parking lot to initiate and conclude trips and facilitate payments manually or through an auto-deduct feature. While creating this prototype, it is essential to consider various user profiles, ensuring their unique characteristics are clearly reflected in the design, associated documentation, and during the presentation. For this prototype, you need to incorporate the following:

Smart vehicle:

- Smart vehicles (bike, bicycle or moped) can be used by users (staff, student, teacher) by registering on the application and making payment on the same.
- The vehicle can be docked in docking stations provided on campus and users will be charged according to a defined scheme. The vehicle can be used both inside and outside campus. The payment can be made using the software app.

User Account Management:

- The software product must allow the user to get onboarded. Once the user opens the app, they should be able to:
    - Create an account - You can decide on the login mechanism to be implemented

- Upload id - You can decide on what ids should be uploaded for various kinds of users.
- Add money to Wallet - Decide on how the wallet functionality will work with respect to the payment mechanisms, boundations like minimum balance etc.
- Users History - Users should be able to look at their trip history

Bike reservation rates/charges:

- The payment is such that for the first x kilometers, a base rate of y rupees is taken. After that, it is at z rupees per 100 meters.
- The user books a vehicle using the application. Keep track of the current vehicle, money due, user details, etc. If a bike is not returned to the docking station within 8 hours and the bike is not renewed, a fine of 50 rupees is deducted every day.

Payment Management:

- The user should be able to make payment for the rides they've taken. Payment can be made via in-app wallet. If there is sufficient money in the wallet, the amount can either be auto deducted (if so enabled by the user before) or else, can be done manually via the app.
- If the wallet doesn't have sufficient money, money has to be added to the wallet via other
- payment options before proceeding to make the payment. Money can be added by existing UPI apps. (Other options are open to interpretation)
- Other options can include deducting from the salary of the user (in case of staff or professors) or adding to the fees (of students). You may add details for adding money to the wallet as per your understanding of payment systems. Your design and/or presentation must make the added details obvious.

Support, Feedback and Ratings:

- Ratings should be provided indicating the satisfaction of the availed service.
- Additionally, user must be able to provide feedback which can improve the app or can help in sustaining the current software
- Support information in the form of documentation should be provided for users' ease of travel and usability.

Trip Management:

- To move the smart bike from the parking lot, the trip must be started. Also, trips can only be started and ended at the designated parking lots of the campus. Trips can be started and ended by using the bike's QR code.
- For starting the trip
  - Scan the QR code -> bike's details get listed in the app -> Start the Trip
- For ending the trip:
  - Park the bike in the parking lot -> End trip on app screen -> Scan QR code

You may add more details/functionalities pertaining to this. List all the assumptions you make.

Parking Lot Management:

- Track the availability of smart bikes in the parking lot, manage bike locations, and update bike statuses based on user actions.

- It should also monitor overall capacity of the parking lot, should include maintenance status (repairs if any, condition of bikes, etc.) and security features (open-ended). It should also have a data logging system facilitating system analysis and future improvements.

Tasks

Phase I: Manual Object-Oriented Design

1. As a team, create a complete UML Class Diagram with all attributes, methods, and cardinalities for the entire system.
2. Create 3 Sequence Diagrams for 3 different scenarios.

Note:

- Proper explanation for all the considered classes (attributes and functions), inheritance(s), relationship(s), cardinalities. The explanation must be concise and clear. Irrelevant explanations will get a penalty.
- Use PlantUML or StarUML to model the relationships (inheritance, association, composition, etc.). Hand-drawn or general drawing tools (draw.io, etc.) will not be considered for final submission.

Phase II: LLM-Assisted Generation

Using the same problem description, you must use an LLM to generate class diagram. Ask the LLM to design the system and provide the output in Mermaid syntax. You must also save a screenshot/image of the diagram the LLM generates from that syntax.

Note: Do not edit the AI's output. We need to see exactly what it produced.

Phase III: Comparative Analysis (Human vs. LLM Review)

Provide a detailed comparison between your Manual Design (Phase I) and the LLM's Design (Phase II) in your **`documentation.md`** file:

1. Compare your manual UML with the LLM's Mermaid structure. Check and analyse whether the LLM correctly interpreted IIIT-H specific constraints.
2. List the specific corrections needed to make the LLM output meet the actual requirements.

Submission Instructions:

Please follow below instructions carefully. **Severe Penalties** will be given if the submission format doesn't adhere to this.

- Submit a single zip file named **`team<team no>.zip`**. For example, if you team number is 10, then your submission file should be **`team10.zip`**.
- **Inside the Zip:** The file must contain exactly one folder and one file:
  - **`Diagrams/` Folder:** This folder **must** have two sub-folders:
    - **`Manual/` Folder:** Manually created Class Diagram and 3 Sequence Diagrams (.png, .jpeg or .pdf).
    - **`LLM/` Folder**: The LLM-generated Class Diagram (screenshot of the Mermaid output).

- `documentation.md` **File:** This is a markdown file where you must include:
  - **Class Explanations (Phase I):** Provide a clear one-line explanation for all considered classes (attributes and functions), inheritances, relationships, and cardinalities. Irrelevant explanations will get a penalty.
  - **Assumptions:** All assumptions made for the design.
  - **LLM Prompts:** The exact prompts used and the LLM generated raw Mermaid code to create class diagram.
  - **Additional Features**: Any additional functionalities you implemented.
  - **Comparative Analysis (Phase III):** A side-by-side comparison of your manual Class Diagram/Code vs. the LLM's Mermaid/Code.
  - **Contributions:** Clearly mention the contribution of each member in this activity.

Deadlines & Other Instructions:

- **Deadline:** January 23rd - 11:59 pm
- **Only one member from the team should submit.** Multiple submissions from the same team are not required.
- **No deadline extensions will be entertained for this activity.**