FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGIES

FALL 2023

CA EXAMINATION

Assignment - CA - groups of 5 students

NDE HURICH DILAN nde.dilan@ictuniversity.edu.cm
ICTU20223351

Madick Ange Cesar

ICTU20223024

madick.angecesar@ictuniversity.edu.cm

Chouaten foyet yann rael

ICTU20222956

chouaten.rael@ictuniversity.edu.cm

Takang Joshua Akombark

ICTU20223088

takang.akombark@ictuniversity.edu.cm

NGANA NOA JUNIOR FREDERIC ABEL

ngana.fredericabel@ictuniversity.edu.cm

ICTU20223538

1-Designing the ICT University Bus Service Software Using the ADD Method

The ADD method (Attribute-Driven Design) is an iterative approach to software architecture that uses quality attribute requirements as the key driver for design decisions. Applying the seven steps of ADD to model, design, and develop the ICT University Bus Service software:

1. Review Inputs:

- Stakeholders: ICT University management, students, finance department, bus driver, conductor.
- Quality Attributes: Security, performance, scalability, maintainability, usability, availability, reliability.
- Functional Requirements: Service subscription, bus location tracking, student attendance, arrival/departure times, bus stop management, payment processing, reporting.
- Constraints: Budget, timeline, existing infrastructure, data privacy regulations.

2. Establish Interaction Goal by Selecting Drivers:

- Main Goal: Efficient and secure transportation for ICT University students, balancing their needs with system stability and resource limitations.
- Primary Drivers:
    - Security: Protect student data, secure payments, and prevent unauthorized access.
    - Performance: Real-time tracking, fast response times, and minimal waiting periods.
    - Maintainability: Easily update features, adapt to changing requirements, and ensure efficient bug fixes.
    - Usability: User-friendly interfaces for students, finance department, and bus crew.

3. Choose Elements for Refinement:

- Bus Service System Core: Focus on the central functionality of managing subscriptions, tracking buses, and facilitating student transportation.

4. Choose Design Concepts that Satisfy Drivers:

- Layered Architecture: Separate presentation (mobile app, web interface), business logic (service layer), and data access layers (database) for maintainability and scalability.
- Microservices Architecture: Decouple specific functionalities like payment processing or attendance management into independent services for flexibility and deployment agility.
- Real-time Tracking System: Implement GPS with a dedicated service to transmit location data and update student interfaces in real-time.

5. Instantiate Architectural Elements and Define Interfaces:

- Identify specific components within each layer:
  - Presentation Layer: Mobile app for subscriptions, tracking, and notifications. Web interface for management, finance, and operations.
  - Business Logic Layer: Services for user management, subscription handling, location tracking, payment processing, and reporting.
  - Data Access Layer: Database storing student data, subscriptions, bus information, and location history.
- Define communication protocols between layers and services using APIs (e.g., RESTful).
- Implement role-based access control for different user groups.

6. Sketch Preview and Record Design Discipline:

- Create a UML diagram illustrating the interaction between components, data flows, and communication channels.
- Highlight security measures and access control points within the diagram.
- Document the design decisions and rationale behind each element and interface.

7. Analyze Current Design and Review Iteration Goals:

- Assess how well the chosen design concept satisfies the chosen drivers and stakeholders' needs.

- Identify potential performance bottlenecks and optimize resource allocation.
- Evaluate the feasibility of implementing the design within budget and time constraints.
- Plan further iterations based on analysis results and desired feature expansions.

Addressing Additional Points:

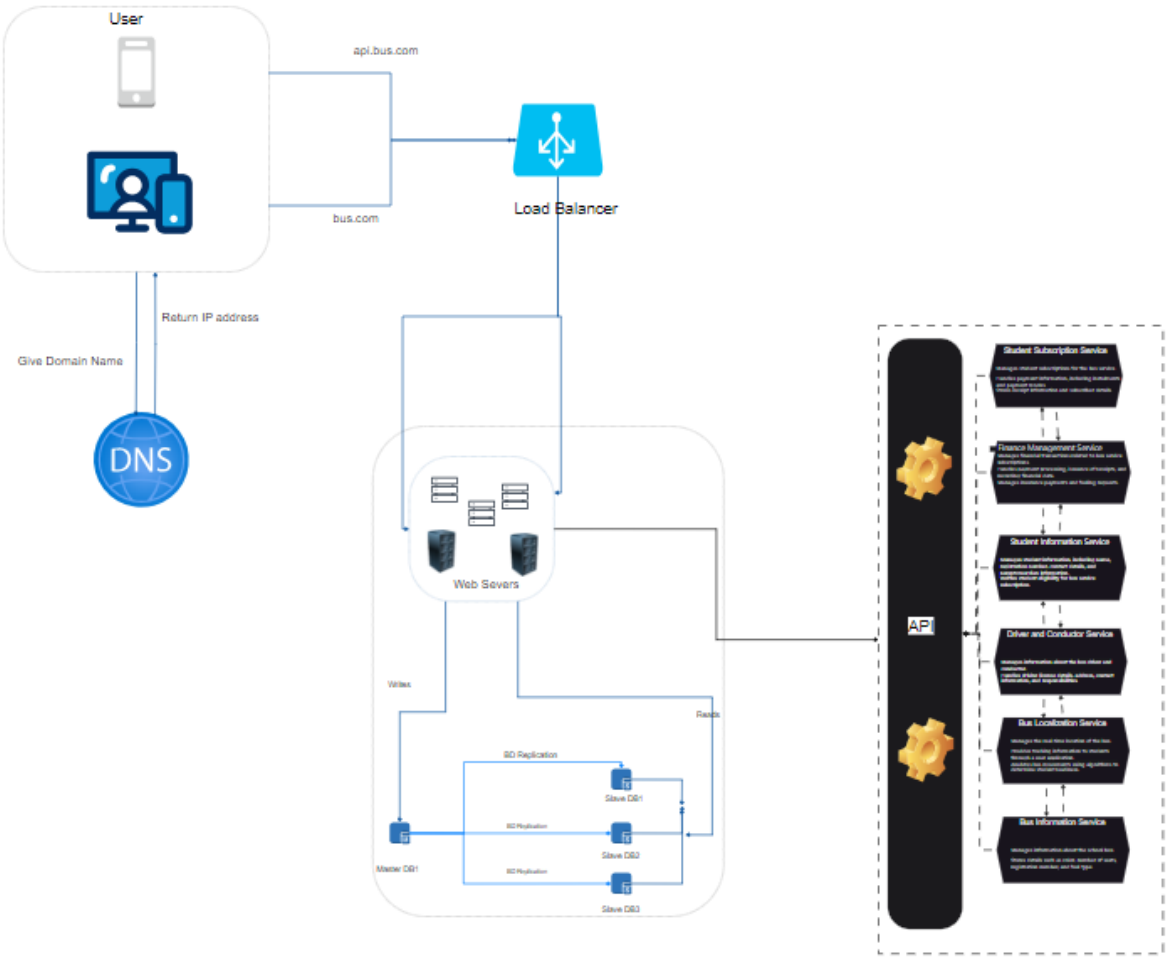
- Why Capture Design Purpose? - To clearly define the overall intent and value of the system, guiding architecture decisions and ensuring alignment with stakeholder needs.
- Why Capture Primary Functional Requirements? - To understand the essential tasks and workflows the system must perform, shaping the architecture to efficiently support them.
- Why Primary QA Scenario? - To identify specific situations where quality attributes are critical, informing architectural choices to address potential vulnerabilities.
- Constraints: - Same as before, including budget, timeline, existing infrastructure, and data privacy regulations.
- Concerns: - Security breaches, system downtime, inaccurate location tracking, poor user experience, and exceeding budget or maintenance resources.

Implementing Similar ADDs:

This process can be applied to refine other aspects of the bus service software, such as:

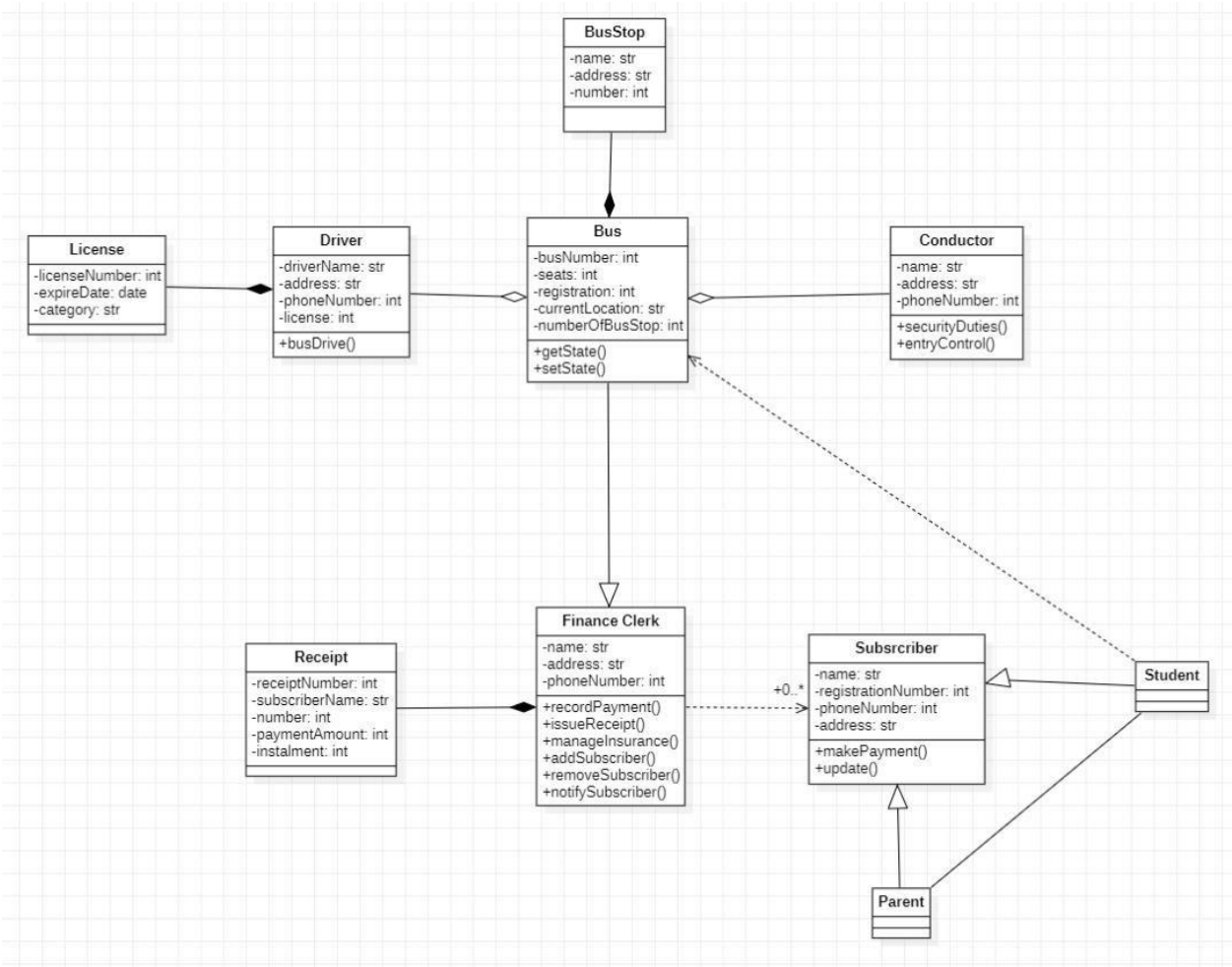
- Payment Gateway Integration: Focus on security, reliability, and integration with existing university payment systems.
- Messaging System Design: Prioritize real-time communication, scalability, and fault tolerance.
- Mobile App Optimization: Emphasize usability, data efficiency, and offline functionality.

Software Architecture



2-Design a class diagram for the system (Apply design principles and patterns to your design)

The Observer pattern to handle the information flow challenges.We implemented an observable object for the bus, and had student subscribers who want to track the bus location register as observers. When the bus location changes,we notify all the subscribed students about the updated location.


BusStop
-name: str
-address: str
-number: int
License
-licenseNumber: int
-expireDate: date
-category: str
Driver
-driverName: str
-address: str
-phoneNumber: int
-license: int
+busDrive()
Bus
-busNumber: int
-seats: int
-registration: int
-currentLocation: str
-numberOfBusStop: int
+getState()
+setState()
Conductor
-name: str
-address: str
-phoneNumber: int
+securityDuties()
+entryControl()
Receipt
-receiptNumber: int
-subscriberName: str
-number: int
-paymentAmount: int
-instalment: int
Finance Clerk
-name: str
-address: str
-phoneNumber: int
+recordPayment()
+issueReceipt()
+manageInsurance()
+addSubscriber()
+removeSubscriber()
+notifySubscriber()
+0..*
Subsrcriber
-name: str
-registrationNumber: int
-phoneNumber: int
-address: str
+makePayment()
+update()
Student
Parent