**FoodPickup Application**

The "Food in IIIT Hyderabad" application aims to revolutionize the dining experience within the IIIT Hyderabad campus by introducing an efficient, user-friendly mobile application that facilitates digital menu access, pre-ordering, real-time order tracking, secure payment, and feedback submission for the campus canteens and mess facilities.

**Requirements** –

• Digital Menus and Pre-ordering: Enables users to browse and order from up-to-date menus.

• Real-Time Order Tracking: Allows users to view the status of their orders.

• Payment Integration: Supports multiple secure payment methods.

• Feedback Mechanism: Facilitates submission of feedback on meals and services.

**Task 1: Requirements and Subsystems**

**Functional and Non-functional Requirements: Akanksha**

**Functional Requirements:**

**Digital Menus and Pre-ordering**

Description: Allows users to browse digital menus and place orders ahead of time.

Architectural Significance: This feature demands a dynamic, updatable content management system for menus, requiring robust database design and API integration to ensure real-time accuracy and efficiency in order processing.

**Real-Time Order Tracking**

Description: Enables users to view the status of their orders in real-time.

Architectural Significance: Necessitates the implementation of WebSocket or similar technology for maintaining a live connection between the server and client app, ensuring immediate updates to users.

**Payment Integration**

Description: Supports secure processing of payments through multiple methods.

Architectural Significance: Requires secure, reliable third-party payment gateway integration, emphasizing the need for secure transmission of sensitive information and compliance with financial data protection standards.

**Feedback Mechanism**

Description: Allows users to submit feedback on meals and services.

Architectural Significance: Calls for a responsive system that can collect, store, and analyse feedback, potentially requiring natural language processing for insights and prioritization of feedback for operational improvements.

**Non-Functional Requirements:**

**Performance**

The application must handle multiple simultaneous requests efficiently, ensuring fast loading times and swift order processing to improve user experience and manage peak load times effectively.

**Usability**

An intuitive, user-friendly interface is crucial for encouraging adoption among campus residents. This involves employing best practices in UI/UX design, accessible to all users, including those with disabilities.

**Reliability**

High reliability and uptime are essential, particularly during peak dining hours, requiring robust error handling, failover mechanisms, and possibly the use of cloud services for scalability and reliability.

**Security**

Given the handling of personal and payment information, adhering to security best practices and compliance with data protection regulations is paramount. This includes secure data storage, encrypted communications, and regular security audits.

**Scalability**

The system should be designed to easily accommodate growth, whether in terms of user numbers, menu items, or functionality, without significant rework. This could involve microservices architecture or similar strategies to ensure components can be scaled independently.

**Subsystem Overview:Akanksha**

**User Interaction Subsystem:**

This subsystem facilitates interactions between the users (students, faculty, and staff of IIIT Hyderabad) and the application through a user-friendly mobile interface. It allows users to browse digital menus, place pre-orders, track orders in real-time, and submit feedback on meals and services. This subsystem aims to enhance the dining experience by making it more convenient and efficient for the campus community.

**Order Management Subsystem:**

Central to the application, this subsystem manages the processing of orders from initiation to completion. It includes functionalities for digital menu display, pre-ordering capabilities, real-time order tracking, and integration with payment systems. It serves both the users by providing a seamless ordering experience and the canteen and mess operators by optimizing order handling and preparation processes.

**Payment Integration Subsystem:**

This subsystem is responsible for handling all aspects of payment processing within the application. It supports multiple secure payment methods, ensuring that transactions are executed safely and efficiently. This subsystem is crucial for facilitating seamless financial transactions between the users and the service providers, thus enhancing the overall usability of the application.

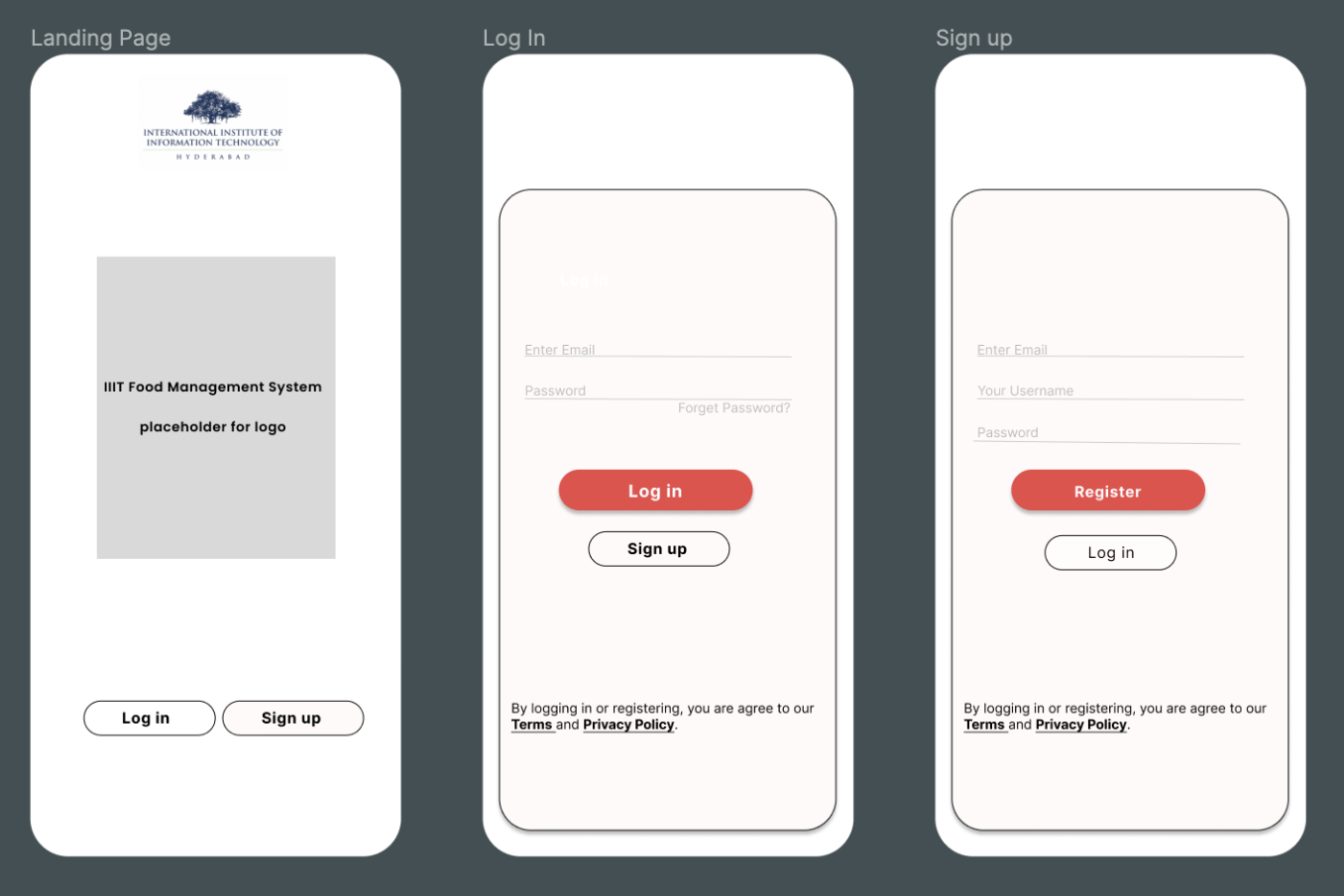
**Feedback Mechanism Subsystem:**

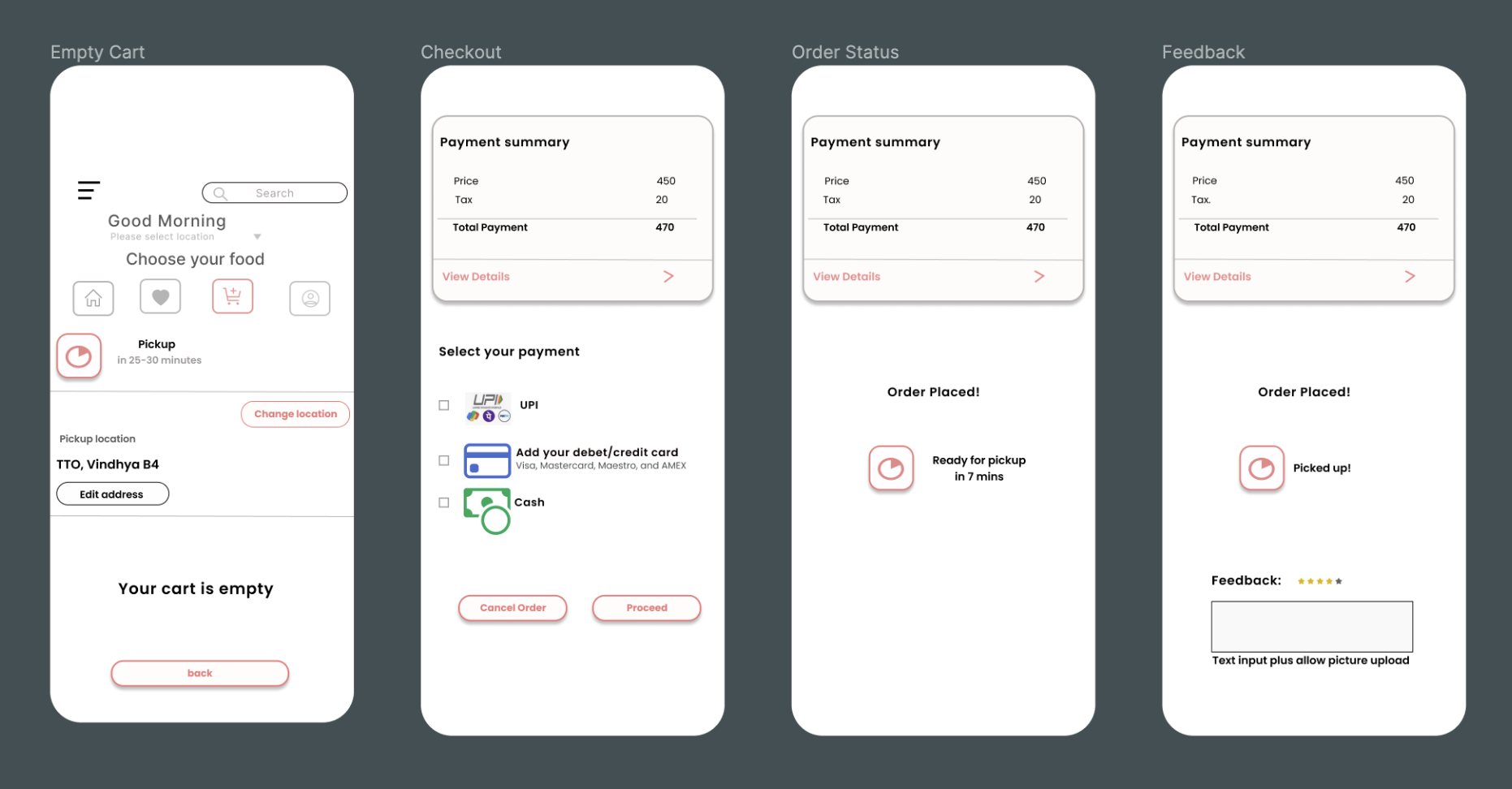
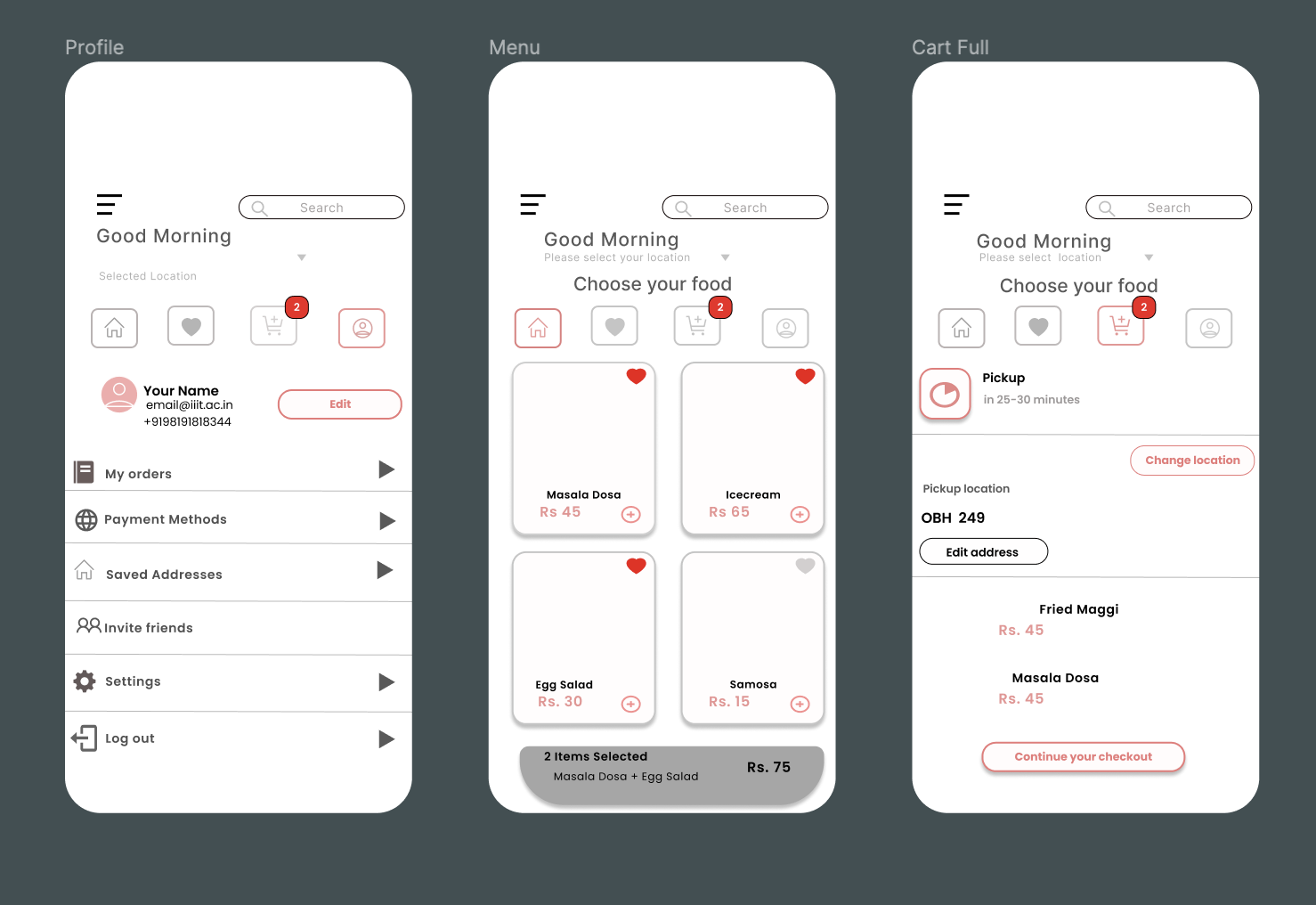
Designed to gather and manage user feedback, this subsystem allows users to submit feedback on their dining experiences directly through the application. This feedback is then available to the canteen and mess operators for service improvement purposes. By fostering engagement between users and service providers, this subsystem plays a vital role in the continuous enhancement of dining services on campus.

**Operational Insights Subsystem:**

A subsystem dedicated to providing actionable insights to canteen and mess operators. This could involve data analysis tools for understanding user preferences, order patterns, and feedback trends to optimise operations and improve services.

**UX Design – Akanksha**

****

****

**Stakeholder Identification:Akanksha**

1. Users (Students, Faculty, Staff)

Concerns:

* Ease of Use: Users demand an intuitive and easy-to-navigate application.
* Service Availability: Concerned with the app's reliability and uptime.
* Menu Variety and Nutritional Information: Looking for diverse food options and nutritional details.
* Payment Security: Ensuring that their financial transactions are safe.
* Order Accuracy and Feedback Responsiveness: Expecting orders to be correctly processed and feedback to be acted upon

Viewpoints and Views:

* User Interface Design View: Focuses on creating an intuitive layout that is accessible and easy to navigate.
* Security View: Emphasizes protecting user data and securing payment transactions.
* Feedback System View: Outlines how users can provide feedback and how it is addressed by the service providers.

2. Canteen and Mess Operators

Concerns:

* Order Management Efficiency: Streamlining the process from order receipt to fulfilment.
* Payment Processing and Inventory Management: Efficient handling of transactions and stock.
* Feedback Reception and Operational Insights: Using feedback for service improvement and gaining insights from data analysis.

Viewpoints and Views:

* Operational Management View: A comprehensive dashboard for tracking orders, managing inventory, and monitoring feedback.
* Payment Integration View: Ensures smooth processing of payments.
* Data Analytics View: Provides insights into customer behavior, menu preferences, and feedback trends.

3. Administrative Staff of IIIT Hyderabad

Concerns:

* System Integrity and Data Privacy: Ensuring the application is secure and user data is protected.
* User Satisfaction and Financial Transparency: Maintaining high service quality and clear financial transactions.
* Regulatory Compliance: Adhering to legal and policy requirements.

Viewpoints and Views:

* Compliance and Privacy View: Focuses on meeting legal standards and protecting user data.
* Financial Management View: Overseeing and reporting on financial transactions.
* System Performance View: Monitoring application reliability and user satisfaction.

4. Developers and Maintainers

Concerns:

* System Scalability: Ability to accommodate growth in user numbers and data volume.
* Maintainability and Extensibility: Ease of updating and adding new features.
* Interoperability: Integration with existing and future systems or services.

Viewpoints and Views:

* Development and Maintenance View: Ensures the technical architecture facilitates easy maintenance and future development.
* Scalability View: Focuses on the system's ability to scale resources as needed.
* Integration View: Addresses the need for seamless integration with other services.

5. External Partners (Payment Gateway Providers, Nutritional Information Sources)

Concerns:

* Integration Ease: Simplifying the integration process with the main system.
* Data Exchange Security: Ensuring secure transmission of data.
* Service Reliability: Dependability of external services for continuous operation.

Viewpoints and Views:

* External Integration View: Outlines the process for integrating external services smoothly.
* Security View: Ensures secure data exchange between systems.
* Reliability View: Focuses on the dependability of external services to maintain uninterrupted operation.

**Major Design Decisions: Aditya**

ADR 1: Adoption of the MVC Architecture

Context: The "Food in IIIT Hyderabad" application requires a robust structure that facilitates clear separation of concerns, simplifies development, and enhances maintainability.

Decision: Implement the application using the Model-View-Controller (MVC) architecture.

Rationale: MVC architecture allows for the separation of data access (Model), user interface (View), and business logic (Controller), promoting organized and modular development. This separation enhances developer productivity, facilitates easier testing, and supports scalable application growth.

Consequences: While MVC provides a structured approach, it requires diligent adherence to the separation of concerns to prevent any one component from becoming overly complex. The team must ensure clear communication and documentation to maintain the architecture's integrity as the application evolves.

**Architecture** – **Aditya**

Users will interact with the system through a mobile application developed using Flutter, known for its excellent UI capabilities and cross-platform efficiency.

**Design Patterns and Architecture**

• MVC Pattern: For separating the application logic, UI, and data.

• Repository Pattern: To abstract the data layer, making the system more modular and testable. • Singleton Pattern: For database connections and API clients to ensure efficient resource use.

• Observer Pattern: For real-time updates on order status.

**Technical Stack** –

• Frontend: Developed in Flutter for compatibility with both Android and iOS devices.

• Backend: Python with frameworks such as Flask to manage APIs, database interactions, and business logic.

• Database: Use of PostgreSQL or MongoDB, depending on schema requirements.

• Payment Integration: Implementation of secure APIs for payment processing.

**Task 3: Architectural Tactics and Patterns - Nileema**

1. Performance –
   1. Resource Management – maintain multiple copies – Using Microservices **Database Sharding**: Sharding involves horizontally partitioning data across multiple databases or database instances. By distributing data storage and processing across multiple shards, it prevents any single database from becoming a bottleneck and improves scalability and resource utilization.
2. Security
   1. Resisting Attacks – Authenticate Users, Authorize Users – using the authentication and authorization module for security and role based access.
3. Modifiability
   1. Localize changes – Generalize module – Having microservices architecture and modularizing the code for general modules like User management.
   2. Prevent Ripple Effect – Maintain existing interfaces, Restrict communication paths – Implemented Microservices for the limited and generalized access.
   3. Usability - Design time – Separate UI from application. Having separate UI project from API in Microservices implementation.

**Architecture Diagram**

A diagram of a software company

Description automatically generated with medium confidence

**DB model** –

A computer screen shot of a computer

Description automatically generated

**Task 4: Prototype Implementation and Analysis**

**Prototype Development:**

**Work Allocation for Prototype -**

1. **API** 
   1. User Registration – Akanksha
   2. FoodMenu Cart, Order, Menu– Chandana
   3. Feedback, Notification, Address, Payment, Canteen - Nileema
2. **DB Creation – Nileema**
3. **UI – Aditya and Bharti**
4. **Layered Architectural Pattern for notification – Aditya**

**Architecture Analysis: Nileema**

Compare your implemented architecture against another pattern of your choice. Provide quantification for at least 2 non-functional requirements (e.g., response time, throughput). Discuss the trade-offs involved.

**Layered Architectural Pattern for Notification Vs Microservices for Notification**

**Notification Usecase –** Notify the user in case the food is ready for the pick up.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Microservices Architecture** | **Layered Architectural** |
| Testability | API, DB and UI are separate components. API and UI are easily testable | The Layered architecture is tightly coupled and hence not easy to test the dependent components |
| Modifiability | Any modification in Business logic example sending notification to all users can be easily implemented and testable. | Any modification to Business logic, is difficult to maintain and modify. |
| Performance | As the DB is separate, the performance is better than Layered architecture. | When there are many simultaneous users, as we are using the same DB, the performance slows down |

**DB model script** –

@startuml

' hide the spot

' hide circle

' avoid problems with angled crows feet

skinparam linetype ortho

!define primary\_key(x) <b><color:#b8861b><&key></color> x</b>

!define foreign\_key(x) <color:#aaaaaa><&key></color> x

!define column(x) <color:#efefef><&media-record></color> x

!define table(x) entity x << (T, white) >>

table(address) {

primary\_key(addressid) : UUID

column(typeofaddress): character varying(120)

column(address\_desc): character varying(120)

column(createddate): datetime

column(updateddate): datetime

foreign\_key(userid): integer <<FK>>

}

table(canteen) {

primary\_key(canteenid): UUID

column(location): character varying(120)

column(canteenname): character varying(120)

column(canteenowner): character varying(120)

column(canteenstatus): character varying(120)

column(createddate) : datetime

column(updateddate) :datetime

}

table(cart){

primary\_key(cartid): UUID

column(orderid): integer

column(cartstatus): character varying(50)

column(cartprice): integer

column(cartusername): character varying(120)

column(createddate): datetime

column(updateddate): datetime

column(cartuserid): integer

}

table(cartmenuitem) {

primary\_key(cartmenuitemid) : UUID

foreign\_key(cartid) : integer <<FK>>

foreign\_key(menuitemid) : integer <<FK>>

foreign\_key(canteenid): integer <<FK>>

column(menuitemquantity) : integer

column(permenuitemprice) : integer

column(totalmenuitemprice): integer

column(createddate): datetime

column(updateddate): datetime

}

table(feedback) {

primary\_key(feedbackid): UUID

foreign\_key(feedbackuserid) : integer <<FK>>

column(feedbackdesc) : character varying(120)

foreign\_key(orderid) : integer <<FK>>

foreign\_key(menuitemid) : integer <<FK>>

column(feedbackdate) : datetime

column(feedbackstatus) : character varying(50)

column(feedbackaction) : character varying(120)

column(feedbackclosureremarks) :character varying(120)

column(feedbackactionuserid) : character varying(120)

column(createddate) : datetime

column(updateddate) : datetime

column(feedbackusername) : character varying(120)

}

table(menuitem) {

primary\_key(menuitemid) : UUID

foreign\_key(canteenid) : integer <<FK>>

column(menuitemdesc) : character varying(120)

column(Permenuitemprice) : integer

column(menuitemstatus) : character varying(50)

column(menuitemtype) : character varying(50)

column(createddate) : datetime

column(updateddate) : datetime

}

table(notification) {

primary\_key(notificationid): UUID

foreign\_key(userid) : integer <<FK>>

column(notificationdesc) : character varying(120)

column(notificationcreateddate) : datetime

column(notificationtype) : character varying(120)

column(notificationstatus) : character varying(50)

column(createddate) : datetime

column(updateddate) : datetime

}

table(order){

primary\_key(orderid): UUID

foreign\_key(userid) : integer <<FK>>

column(orderdate) : datetime

column(cartid) : integer

column(orderstatus) : character varying(120)

column(paymentid) : integer

column(orderprice) : integer

coumn(createdon) : datetime

coumn(updatedon) : datetime

}

table(users) {

primary\_key(userid) : UUID

column(username) : character varying(80)

column(firstname) : character varying(120)

column(lastname) : character varying(120)

column(mobileno) : bigint

column(email) : character varying(120)

column(aadharid) : character varying(120)

column(userrole) : character varying(120)

column(userstatus) : character varying(120)

column(password) : character varying(100)

column(registered\_on) : timestamp

column(updateddate) : timestamp

column(remarks) : character varying(120)

}

table(payment) {

primary\_key(paymentid) : UUID

foreign\_key(userid) : integer <<FK>>

column(mobileno) : bigint

column(amount) : integer

column(bankaccountnumber) : character varying(120)

column(bankname) : character varying(120)

column(FSCIcode) : character varying(120)

column(pincode) : character varying(100)

column(paymentdate) : timestamp

column(updateddate) : timestamp

column(paymentstatus) : character varying(120)

}

order||..|| users

order ||..|| cart

order ||..|| payment

notification }|..|| users

menuitem||..|| canteen

feedback||..|| menuitem

feedback||..|| order

feedback||..|| users

cart||..||order

cart}|..||users

cartmenuitem }|..||cart

cartmenuitem ||..||menuitem

cartmenuitem }|..||canteen

canteen }|..||users

address}|..||users

@enduml