#### **a. Identifying Subsystems**

A UML package diagram helps to identify and organize the system into subsystems that highlight relationships and dependencies between different parts of the system. For our **Credit Fraud Detection System**, the primary subsystems can be categorized as follows:

* **User Interface Subsystem**: Responsible for the interaction between the user and the system, this package manages user authentication, CSV data uploads, alert notifications, and report generation.
* **Transaction Processing Subsystem**: This handles all data input and validation, including transaction data parsing and formatting checks. It interacts directly with the HMM and RNN models for fraud detection.
* **Fraud Detection Subsystem**: The core processing component, which leverages Hidden Markov Models (HMM) and Recurrent Neural Networks (RNN) for analyzing transaction sequences and identifying potential fraud.
* **Alerting and Reporting Subsystem**: This manages communication with the user, sending real-time alerts and generating reports on suspicious activity or system performance.
* **Database Subsystem**: This package deals with the storage and retrieval of user data, transaction history, and fraud detection metrics.

The diagram below illustrates the interaction between the major subsystems:

**[Insert UML Package Diagram]**(Visualize different subsystems and how they interact with each other, showing dependencies and flows between components like UI, Fraud Detection, and Database subsystems.)

#### **b. Architecture Styles**

The architecture of our **Credit Fraud Detection System** is designed using the following styles:

* **Layered Architecture**: Our system follows a layered design pattern where each layer has a specific role. For example, the presentation layer (UI) is separated from the business logic layer (fraud detection algorithms) and the data access layer (database).
* **Client-Server Architecture**: The system operates on a client-server model where the front-end (client) handles user input, while the back-end (server) processes transactions using fraud detection models and stores results in the database.
* **Microservices Architecture**: The fraud detection, alerting, and reporting functionalities are isolated into smaller independent services that can be developed, deployed, and scaled independently.

This choice was made to ensure scalability, flexibility, and ease of maintenance, allowing us to adapt or extend individual components without affecting the whole system.

#### **c. Mapping Subsystems to Hardware**

Our system is designed to run on multiple machines. The primary mapping of subsystems to hardware is as follows:

* **Client Side (User Machine)**: The **User Interface Subsystem** operates on the client side, which could be a web browser or mobile app. The client submits transaction data and receives notifications.
* **Server Side (Cloud/Server)**: The **Transaction Processing Subsystem**, **Fraud Detection Subsystem**, **Alerting and Reporting Subsystem**, and **Database Subsystem** reside on the server. The server handles all backend processes, including transaction analysis and data storage.

The separation allows for scalability, where multiple clients can interact with the server concurrently, and the fraud detection algorithms can run efficiently on powerful backend servers.

#### **d. Connectors and Network Protocols**

Since the system runs on both client and server, we rely on several communication protocols:

* **HTTP/HTTPS**: Used for secure communication between the user interface (browser or mobile) and the server. HTTPS is crucial to ensure data privacy and security.
* **RESTful APIs**: The subsystems communicate via RESTful APIs, especially for client-server interactions, such as uploading transaction data and fetching reports.
* **JDBC (Java Database Connectivity)**: Used for accessing and manipulating data within the relational database on the backend.

We chose these protocols for their widespread use, robustness, and ease of integration with cloud services and modern web technologies.

#### **e. Global Control Flow**

The system uses an **event-driven control flow**. Instead of following a strict, linear sequence of actions, the system waits for specific user actions (events), such as CSV uploads or fraud alerts, before triggering specific functions.

* **Execution Order**: It is not procedure-driven, but rather event-driven, allowing users to upload files, check reports, and respond to alerts in any order, depending on the situation.
* **Time Dependency**: There are no strict real-time constraints, but the system is designed to provide near real-time fraud detection and alerting. It operates on asynchronous events and processes data in the background.

The system is **not real-time** in the sense of having hard time constraints, but it is expected to deliver fraud alerts promptly after transaction analysis.

#### **f. Hardware Requirements**

The system has the following hardware and resource requirements:

* **User Machine (Client Side)**:
  + Browser capable of running modern JavaScript frameworks (e.g., Chrome, Firefox).
  + Minimum screen resolution of 1024 × 768 pixels for proper display of UI components.
  + Stable internet connection with at least 1 Mbps bandwidth for uploading transaction data.
* **Server Machine (Back-End)**:
  + Minimum 4 CPU cores and 8 GB RAM to support transaction analysis and real-time fraud detection.
  + Minimum of 10 GB storage for transaction data and model training data.
  + Network bandwidth of at least 10 Mbps for handling multiple concurrent client requests.
  + Access to a secure, scalable cloud environment like AWS, GCP, or Azure for deployment.

### **Part 6: Project Size Estimation Based on Use Case Points**

Use case points (UCP) help in estimating the project size by assigning points based on the complexity of use cases.

* **Actors Complexity**:
  + Simple actors (e.g., UI components): 1 point each.
  + Medium actors (e.g., APIs): 2 points each.
  + Complex actors (e.g., Database, Fraud Detection Subsystems): 3 points each.
* **Use Case Complexity**:
  + Simple use cases (e.g., User Authentication): 5 points each.
  + Medium use cases (e.g., CSV Upload, Alert Notifications): 10 points each.
  + Complex use cases (e.g., Fraud Detection, Manual Checking, Report Generation): 15 points each.

#### **Example Estimation:**

* Total number of actors: 5.
* Total number of use cases: 8.

Based on this evaluation:

* **Total UCP for Actors** = (2 × 1) + (2 × 2) + (1 × 3) = 10 points.
* **Total UCP for Use Cases** = (3 × 5) + (3 × 10) + (2 × 15) = 75 points.

Thus, the **total use case points** estimation for the project is approximately **85 UCP**. This gives us an estimation of the project size, allowing for better planning of resources and time.