Khaula Molapo

20240001

Web and Mobile Application Development

Week 4

## 1.

A startup wants to build a social media mobile app where users can post pictures, like content, and send messages. The app needs to integrate with several external systems to function properly. First, it connects to a cloud database where all user data, posts, and comments are stored. Second, it integrates with a payment gateway to allow users to buy premium features. Third, the app connects to a notification service so users receive real-time alerts when someone likes or comments on their posts. The development team starts by defining how each system communicates using APIs. Authentication is handled through a secure login system that connects with Google or Apple accounts. Data from the mobile app is sent to the backend server, which processes requests and updates the database. Testing is done to ensure that data flows correctly between systems without delays or errors. Monitoring tools are added to track performance and failures. This integration approach ensures that all services work together smoothly, creating a stable and responsive user experience while keeping data secure and organized.

## 2.

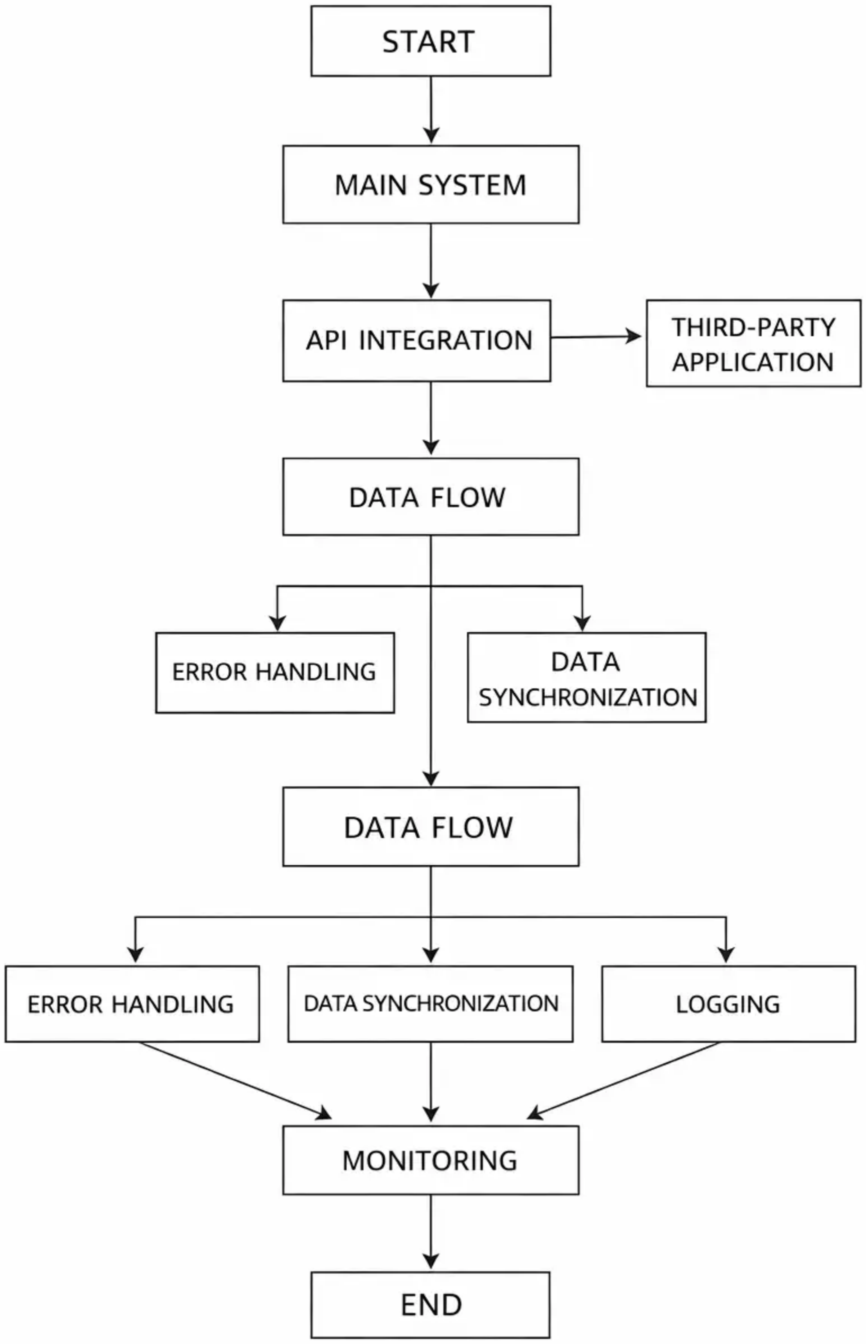
A connection pool is a method used in software systems to manage database connections efficiently. Instead of creating a new database connection every time a request is made, the system creates a group of reusable connections at the start. When the application needs to access the database, it takes an available connection from the pool and returns it after finishing the task. The main role of connection pools is to improve performance and reduce system load. Creating database connections takes time and system resources, so reusing existing ones makes the application faster and more efficient. Connection pools also help control how many users can access the database at the same time, preventing overload or crashes. They improve scalability because the system can handle more requests smoothly. Overall, connection pools make backend applications more stable, responsive, and capable of handling high traffic without slowing down or failing.

## 3.

Setting up a Node.js API connected to MongoDB or PostgreSQL showed how backend systems communicate with databases. The process started by installing Node.js, creating a basic server, and connecting it to the database using a driver or ORM tool. API routes were created for tasks such as creating users, reading data, updating records, and deleting entries. Postman was used to send test requests and check if the API returned correct responses. One important lesson learned was the importance of proper error handling and validation, because small mistakes caused failed requests or incorrect data. Testing with Postman helped quickly find bugs and confirm that endpoints worked correctly. Another benefit was understanding how APIs structure data using JSON format. Overall, the experience showed how backend integration works in real systems and improved understanding of server logic, database communication, and API testing processes.

## 4.

The integration workflow diagram created in Canva shows how different parts of the system communicate with each other. The diagram starts with the user interacting with the mobile app or website. From there, requests are sent to the backend API server, which processes the information. The server then communicates with external services such as the database, payment gateway, and notification system. Arrows are used to show the direction of data flow between each component. The purpose of this diagram is to provide a clear visual overview of how the system works. Instead of reading long technical descriptions, developers and stakeholders can quickly understand how data moves through the system. It also helps identify potential problems or bottlenecks during integration. By using a visual workflow, teams improve communication, plan integrations more effectively, and ensure that all systems connect properly to deliver a reliable and efficient application.


START
MAIN SYSTEM
API INTEGRATION
THIRD-PARTY APPLICATION
DATA FLOW
ERROR HANDLING
DATA SYNCHRONIZATION
DATA FLOW
ERROR HANDLING
DATA SYNCHRONIZATION
LOGGING
MONITORING
END