# **Technical Documentation for Full-Stack Application Using 4-Tier Architecture**

Version 1.0

Index

1. Overview
   * System Technologies
   * Architecture and Deployment
2. 4-Tier Architecture Layers
   * Presentation Layer (ReactJS)
   * API Layer (Node.js)
   * Application Layer (Python Flask)
   * Data Layer (MySQL)
3. Presentation Layer: ReactJS
   * Overview and Role
   * Key Dependencies in package.json
   * Important Packages
   * Scripts for Development and Build
4. API Layer: Node.js
   * Overview and Role
   * Key Dependencies
   * Important Packages
5. Application Layer: Python Flask
   * Overview and Role
   * Key Responsibilities
   * Common Flask Routes
6. Data Layer: MySQL
   * Overview and Role
   * Key Responsibilities
   * Example MySQL Interaction with Flask
7. SaaS Model Customization
   * Frontend Customizations
   * Backend Customizations (Flask)
   * API Layer Customizations (Node.js)
   * Database Customizations
8. System Architecture Diagram
   * Overview of the Architecture
9. Conclusion
   * Summary of Architecture and Flexibility

## **Overview**

This technical documentation outlines the architecture and package dependencies for a full-stack application using a **4-tier architecture**. The system is built with the following technologies:

* **Frontend**: ReactJS
* **Backend**: Python Flask
* **Database**: MySQL
* **API Layer**: Node.js (for API requests, authentication, and communication between frontend and backend)

The application can be deployed as a **Software as a Service (SaaS)** solution or implemented for individual clients. The architecture is modular and customizable to meet specific client requirements.

### **4-Tier Architecture Layers**

The system is structured into four layers:

1. **Presentation Layer**: Handles the User Interface and User Experience with ReactJS.
2. **API Layer**: Manages API requests and authentication with Node.js.
3. **Application Layer**: Implements business logic and computations with Python Flask.
4. **Data Layer**: Provides persistent data storage with MySQL.

## **Presentation Layer: ReactJS**

**Technology**: ReactJS

The **Presentation Layer** is responsible for the user interface (UI) and user experience (UX). The application is built using ReactJS to provide a dynamic and responsive interface that communicates with the backend APIs for fetching and displaying data.

### **Key Dependencies in package.json (Frontend)**

json

Copy

"dependencies": {  
 "@babel/plugin-proposal-private-property-in-object": "x.x.x",  
 "@testing-library/jest-dom": "^5.16.5",  
 "@testing-library/react": "^13.4.0",  
 "@testing-library/user-event": "^13.5.0",  
 "axios": "^1.4.0",  
 "bootstrap": "^5.3.2",  
 "dotenv": "^16.4.7",  
 "jsonwebtoken": "^9.0.1",  
 "loglevel": "^1.8.1",  
 "path-browserify": "^1.0.1",  
 "query-string": "^8.1.0",  
 "react": "^18.2.0",  
 "react-auth-kit": "^2.12.4",  
 "react-bootstrap": "^2.9.2",  
 "react-dom": "^18.2.0",  
 "react-modal": "^3.16.1",  
 "react-router-dom": "^6.0.0",  
 "react-scripts": "^5.0.1",  
 "react-treebeard": "^3.2.4",  
 "stack-trace": "^1.0.0-pre2",  
 "web-vitals": "^2.1.4"  
}

### **Important Packages:**

* **React**: A JavaScript library for building user interfaces. The core of the presentation layer.
* **Axios**: A promise-based HTTP client for making API requests to the backend (Node.js).
* **Bootstrap**: A popular CSS framework for designing responsive and modern UIs.
* **React Router DOM**: A library for handling client-side routing and navigation.
* **React-Bootstrap**: Bootstrap components rebuilt for React.
* **JWT (jsonwebtoken)**: Used for handling JSON Web Tokens for user authentication.

### **Scripts:**

json

Copy

"scripts": {  
 "start": "cross-env HTTPS=true SSL\_CRT\_FILE=./certs/server.crt SSL\_KEY\_FILE=./certs/server.key react-scripts start",  
 "build": "react-scripts build",  
 "test": "react-scripts test",  
 "eject": "react-scripts eject"  
}

These scripts are used for running, building, and testing the ReactJS application. They also handle environment-specific configurations, like enabling HTTPS in development.

## **API Layer: Node.js**

**Technology**: Node.js

The **API Layer** in Node.js is responsible for handling API requests, user authentication, and interacting with the application layer (Python Flask) to perform business logic. It exposes REST APIs to the frontend and ensures secure data transfer and authentication.

### **Key Dependencies (Backend)**

bash

Copy

bcrypt==4.2.1  
blinker==1.9.0  
certifi==2024.8.30  
charset-normalizer==3.4.0  
click==8.1.7  
colorama==0.4.6  
Flask==3.1.0  
Flask-Cors==5.0.0  
Flask-JWT-Extended==4.7.1  
idna==3.10  
itsdangerous==2.2.0  
Jinja2==3.1.4  
MarkupSafe==3.0.2  
mysql-connector-python==9.1.0  
numpy==2.1.3  
PyJWT==2.10.1  
python-dotenv==1.0.1  
requests==2.32.3  
urllib3==2.2.3  
Werkzeug==3.1.3

### **Important Packages:**

* **Flask**: A micro web framework for Python, used for building the REST API in the application layer.
* **Flask-CORS**: Provides Cross-Origin Resource Sharing (CORS) support for the Flask app, allowing communication between the frontend (ReactJS) and backend (Node.js).
* **Flask-JWT-Extended**: A Flask extension for handling JWT-based authentication.
* **PyJWT**: A Python library for encoding and decoding JSON Web Tokens.
* **mysql-connector-python**: The MySQL connector for Python, allowing the backend to communicate with the MySQL database.
* **python-dotenv**: Used for managing environment variables, like database credentials and API keys, securely.

## **Application Layer: Python Flask**

**Technology**: Python Flask

The **Application Layer** is built using Python and Flask, which handles the core business logic and computations of the application. It processes data requests sent from the API Layer (Node.js), performs necessary computations, and returns the results.

### **Key Responsibilities:**

* Handle requests from the Node.js API Layer.
* Perform business logic, such as complex calculations or data transformations.
* Interface with the MySQL database for data storage and retrieval.
* Return the processed data to the API Layer for further handling.

### **Common Flask Routes:**

Example of a basic Flask route in the application layer:

python

Copy

from flask import Flask, jsonify, request  
import jwt  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/api/data', methods=['GET'])  
def get\_data():  
 # Business logic for fetching and processing data  
 data = fetch\_data\_from\_db()  
 return jsonify(data)

## **Data Layer: MySQL**

**Technology**: MySQL

The **Data Layer** provides persistent data storage for the application. It is responsible for managing and querying the application's data.

### **Key Responsibilities:**

* Store and retrieve application data (e.g., user information, transaction history).
* Handle complex queries and ensure fast data access and retrieval.
* Interact with the **Application Layer** (Python Flask) to provide or update data as needed.

### **Example of MySQL Interaction in Python Flask:**

python

Copy

import mysql.connector  
  
def fetch\_data\_from\_db():  
 connection = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="password",  
 database="app\_db"  
 )  
 cursor = connection.cursor()  
 cursor.execute("SELECT \* FROM users")  
 data = cursor.fetchall()  
 connection.close()  
 return data

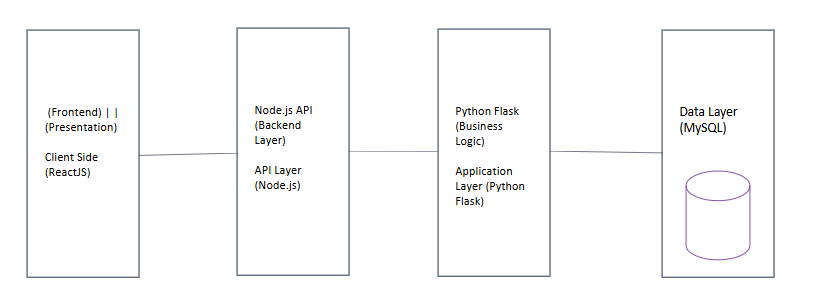
## **SaaS Model Customization**

This application can be deployed as a **Software as a Service (SaaS)** solution. Clients can customize the application as follows:

1. **Frontend**:
   1. Modify the user interface and design.
   2. Add custom branding and themes.
2. **Backend (Flask)**:
   1. Add or modify business logic to cater to specific client needs.
   2. Implement additional API endpoints for client-specific functionality.
3. **API Layer (Node.js)**:
   1. Configure new API routes for custom data handling and authentication methods.
4. **Database**:
   1. Tailor database schemas to match client-specific requirements.
   2. Allow clients to integrate with their own data sources or services.

## **System Architecture Diagram**

Below is a simplified architecture diagram that represents the 4-layer structure of the application:



**Conclusion**

This document outlines the technical architecture, key dependencies, and components used in the 4-tier application built with **ReactJS**, **Node.js**, **Python Flask**, and **MySQL**. The application is designed to be scalable and customizable, offering flexibility for both client-specific deployments and SaaS-based offerings.

The **ReactJS** frontend interacts with the **Node.js API Layer**, which in turn communicates with the **Python Flask Application Layer** for business logic and with the **MySQL Data Layer** for persistent storage. This architecture ensures separation of concerns, scalability, and easy maintainability.