Project Design Phase-II
Technology Stack (Architecture & Stack)

Date	26TH OCT 2032
Team ID	Team-590954
Project Name	Money matters- A personal finance management app
Maximum Marks	4 Marks

Technical Architecture:

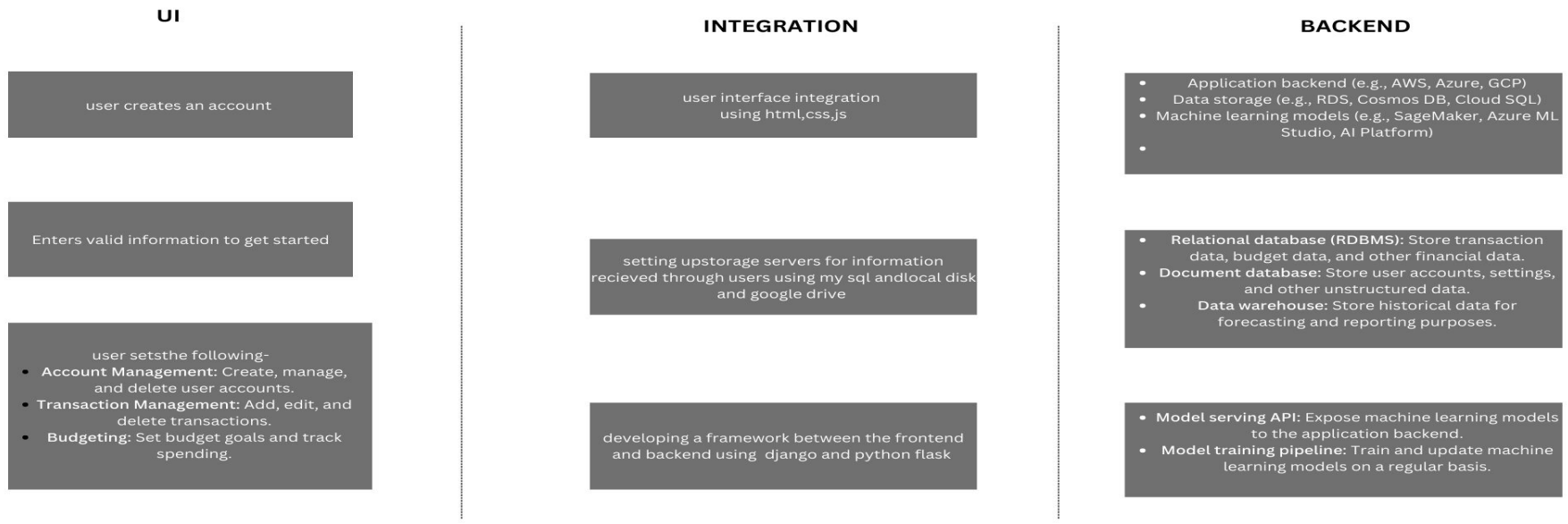

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Dashboard, account summary, transaction tracking budgeting tools, bill reminders, saving goals, security features, offline mode, help and support.	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	User authentication and authorization, user profle, data storage, ransaction management, account balancing, reports and analytics, security, data backup and recovery etc	Java / Python
3.	Database	**The database logic for a finance management app is crucial for storing and managing financial data securely and efficiently like, data schema, data relationships, normalization, indexes, data encryption, data validation and constraints, backup and recovery etc.**	File Manager, MySQL, NoSQL, etc.
4.	File Storage/ Data	**File storage and data management for a finance management app typically involve handling various types of data, including documents, receipts, reports, and user-generated content** **File metadata, document storage, user permissions, data backup, scalability, user-friendly interface, metadata search and security etc.**	Local System, Google Drive Etc

5.	Frame Work	**When developing a finance management app, choosing the right framework is crucial for efficiency, maintainability, and scalability. The choice of framework can depend on your development team's expertise, the specific requirements of your app, and the platform you intend to target (e.g., web, mobile, desktop). Here are some popular frameworks for building finance management apps across different platforms:** Django, ruby on rails, angular, react, react native, flutter, swift and kotlin, electron, apache cordove etc.	Python Flask, Django etc
6.	Deep Learning Model	In the context of a financial management app, the model plays a critical role in handling and managing financial data and related business logic. Here are the specific purposes of a model in a financial management app: 1. **Data Representation**: 2. **Data Validation**: 3. **Data Access and Manipulation**: 4. **Financial Calculations**: 5. **Currency Conversion and Exchange Rates**: 6. **Budget Management**: 7. **Data Synchronization**: 8. **Data Encryption and Security**: 9. **Business Rules and Logic**: 10. **Data Auditing and Logging**: 11. **Notification and Alerts**: 12. **Report Data**: 13. **Transaction History**: 14. **Compliance with Financial Regulations**: 15. **Data Integrity and Consistency**: 16. **User Profile and Account Management**:	CNN, Transfer Learning etc.

7.	Infrastructure (Server / Cloud)	Deploying a financial management app on a local system or a cloud server involves several steps, including setting up the infrastructure, configuring the server environment, deploying the application, and ensuring security and scalability. Here are the key steps for deploying a financial management app on a cloud server: **1. Infrastructure Selection: ** **2. Server Setup: ** **3. Operating System Installation:** **4. Database Setup:** **5. Web Server Configuration:** **6. Application Deployment:** **7. Application Environment:** **8. Security Configuration:** **9. Backup and Data Management:** **10. Load Balancing (Optional):** **11. Monitoring and Alerts:** **13. DNS and Domain Configuration:** **14. Testing:** **15. Compliance and Regulations:** **16. Documentation:** **17. Continuous Deployment (Optional):** **18. User Training (if applicable):** The deployment process may vary based on the technology stack and cloud service provider you choose. It's essential to maintain good practices for security, scalability, and regular maintenance to ensure the optimal performance and security of your financial management app in a cloud server environment.	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology

1.	Open-Source Frameworks	1. **Django**: 2. **Flask**: 3. **Ruby on Rails**: 4. **Spring Boot**: 5. **React**: 6. **Angular**: 7. **Vue.js**: 8. **Node.js**: 9. **Electron**: 10. **React Native**: 11. **Apache Cordova**: 12. **Quasar Framework**: 13. **Chart.js**: 14. **NumPy** and **Pandas**: 15. **Chartist.js**: 16. **Money.js**: 17. **Quandl**: 18. **Leaflet**:	Python’s Flask

2.	Security Implementations	Developing a secure financial management app is of utmost importance, as it deals with sensitive financial data. Below is a list of common security and access controls, along with the use of firewalls and other measures to protect a financial management app: **1. Authentication and Authorization:** **2. Data Encryption:** **3. Firewall and Network Security:** **4. Secure APIs:** **5. Data Access Controls:** **6. Password Policies:** **7. Security Patch Management:** **8. Security Logging and Monitoring:** **9. Data Backup and Recovery:** **10. User Session Management:** **11. Third-Party Integration Security:** **12. Compliance and Regulation:** **13. Cross-Site Request Forgery (CSRF) Protection:** **14. Cross-Origin Resource Sharing (CORS) Control:** **15. Mobile App Security (if applicable):** **16. Regular Security Audits and Penetration Testing:** **17. Privacy Controls:** **18. Disaster Recovery Plan:** **19. Legal and Ethical Considerations:**	e.g. SHA-256, Encryptions, IAM Controls, OWASP etc.

3.	Scalable Architecture	**Three-Tier Architecture:** 1. **Scalability of Three-Tier Architecture**: - **Balanced Resource Allocation**: **Improved Performance**: **Microservices Architecture:** 1. **Scalability of Microservices Architecture**: - **Granular Scalability**: - **Flexibility**: - **Fault Isolation**: - **Elasticity**: **Choosing the Right Architecture for a Financial Management App**: - **Three-Tier Architecture** is well-suited if - **Microservices Architecture** is a good choice if your app is expected to grow significantly, has complex functionalities, or if you want to maintain a high level of flexibility and fault tolerance. It allows for more efficient allocation of resources and makes it easier to develop and deploy new features independently.	Technology used

4.	Availability	Ensuring high availability is crucial for a financial management app because users rely on these applications for critical financial transactions and data access. Any downtime or unavailability can have significant consequences. To justify the availability of a financial management app, consider the following factors: 1. **Continuous Access to Financial Data**: 2. **Transaction Reliability**: 3. **Business Continuity**: 4. **Load Balancers**: 5. **Redundancy and Failover**: 6. **Distributed Servers**: 7. **Geographical Redundancy**: 8. **Scalability and Elasticity**: 9. **Disaster Recovery**: 10. **Monitoring and Alerts**: 11. **Regular Maintenance and Updates**: 12. **Data Replication**: 13. **Highly Available Database Systems**: 14. **CDN for Content Delivery**: 15. **Service Level Agreements (SLAs)**: In conclusion, for a financial management app, ensuring high availability is paramount. Load balancers, distributed servers, redundancy, failover mechanisms, and disaster recovery plans are essential components of an infrastructure that can justify and deliver the level of availability required to meet user expectations and business needs.	Technology used

S.No	Characteristics	Description	Technology
5.	Performance	Performance is a critical aspect of a financial management app to ensure a smooth and responsive user experience, especially during peak usage times. Here are several design considerations to optimize the performance of your financial management app: 1. **Scalability**: 2. **Caching**: 3. **CDN (Content Delivery Network)**: 4. **Asynchronous Processing**: 5. **Database Optimization**: 6. **API Optimization**: 7. **Lazy Loading**: 8. **Content Minification**: 9. **Content Delivery Optimization**: 10. **Use of Content Delivery Networks (CDNs)**: 11. **Client-Side Caching**: 12. **Compression**: 13. **Response Time Monitoring**: 14. **Database Connection Pooling**: 15. **Resource Cleanup**: 16. **Content Security Policy (CSP)**: 17. **Load Testing**: 18. **High Availability**: 19. **Database Sharding**: 20. **Resource Cleanup**: 21. **Caching Strategies**: 22. **User-Friendly Feedback**: Optimizing the performance of a financial management app is an ongoing process. Regularly monitor and fine-tune the application to address any new performance challenges that may arise as usage grows or changes.	Technology used

References:

Khatabook app