FlickFix

Team 14:

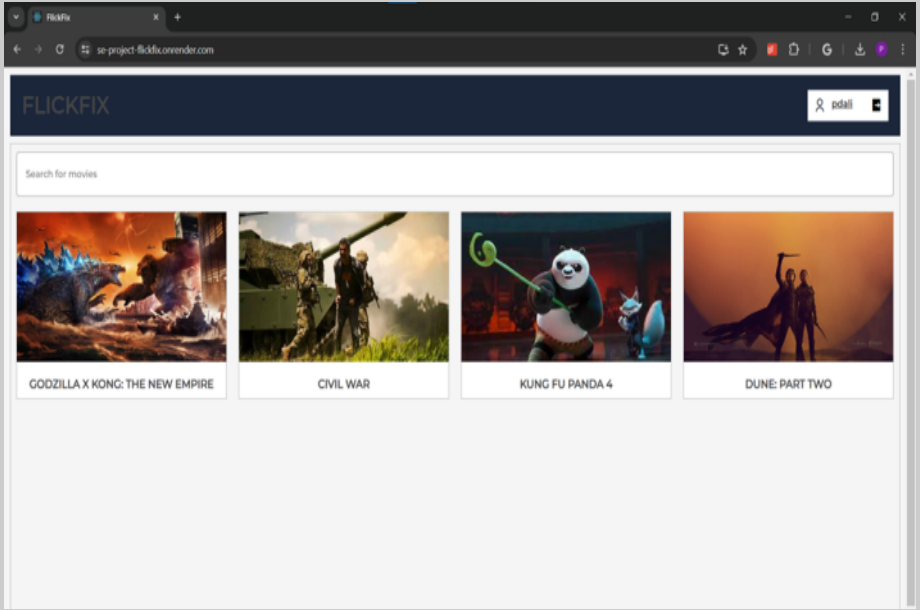
Parth Dali
Archita Palkar
Hemesh Malathi
Sarthak Choudhary
Ashutosh Jagdale

Project Concept

- FlickFlix is the ultimate movie booking platform that transforms how you experience movies at your local theatres.

- FlickFlix offers a seamless portal to explore and book the latest films at cinemas near you.

- Not just for movie-goers, FlickFlix also provides a platform to theater owners to grow their business.



Objective

To redesign and enhance a movie theater ticket booking system, making it streamlined and user-friendly.

Key Features:

- Customer-Centric
- Admin Control dashboard
- Integrated Payment Gateway
- Real-time booking
- User Interfaces for Multiple Roles

Tech in Use

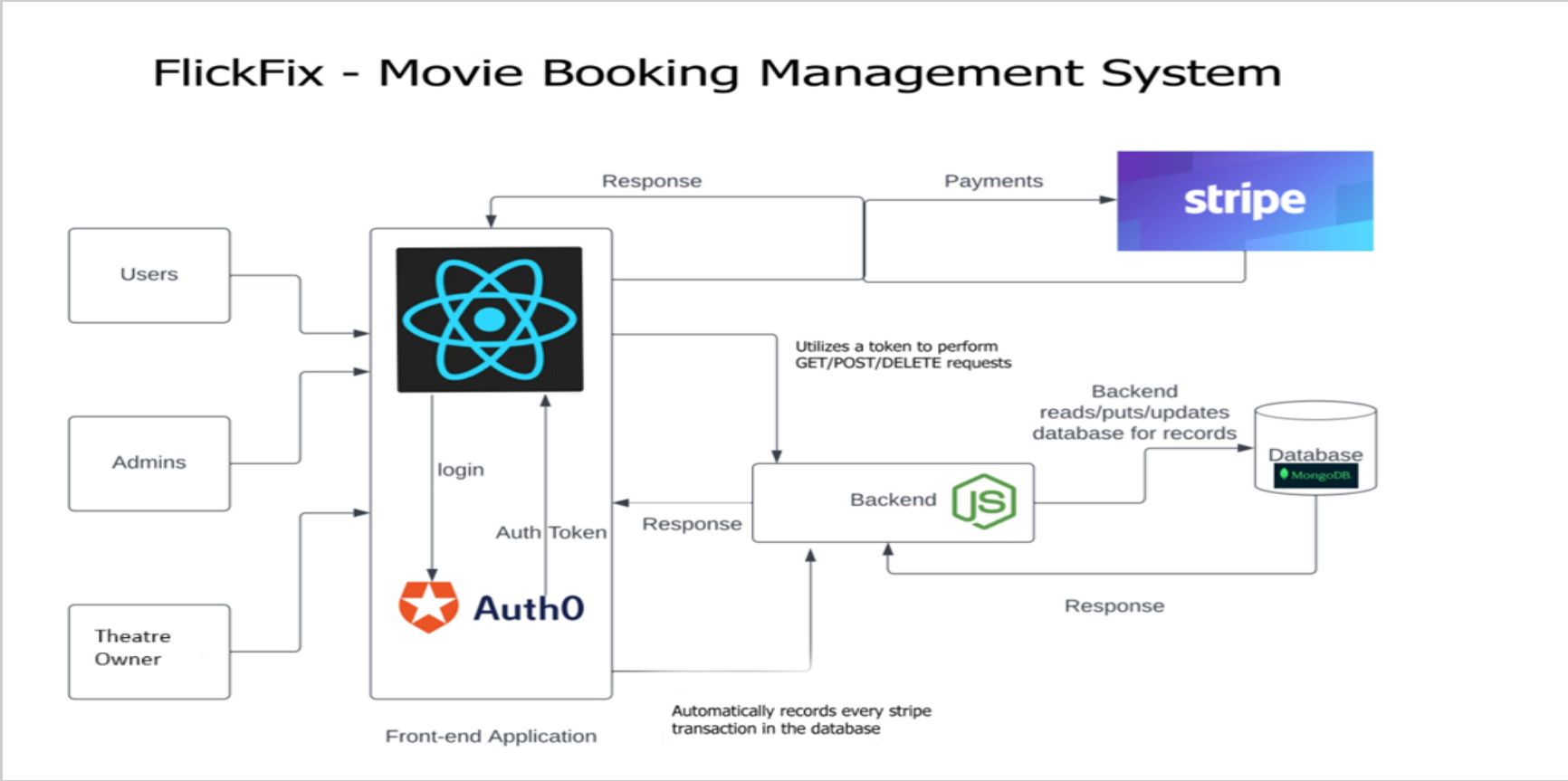
Backend: Node JS & Express JS

Frontend: React

Database: MongoDB

Other tools: Visual Studio code, Git

Application Architecture Overview



System Architecture

For our Movie Tickets Booking Application, we've implemented a hybrid architecture that combines elements of microservices and monolithic models, emphasizing a service-based framework:

- **Frontend Service(s):** Built with React.js, focusing on reusable, modular UI components for a dynamic user experience.
- **Backend Service(s):** Utilizes Node.js and Express.js for RESTful API interactions and modular business logic processing.
- **Database(s):** MongoDB is used for its flexibility and scalability, fitting well with our JavaScript-based backend due to its document-based storage.

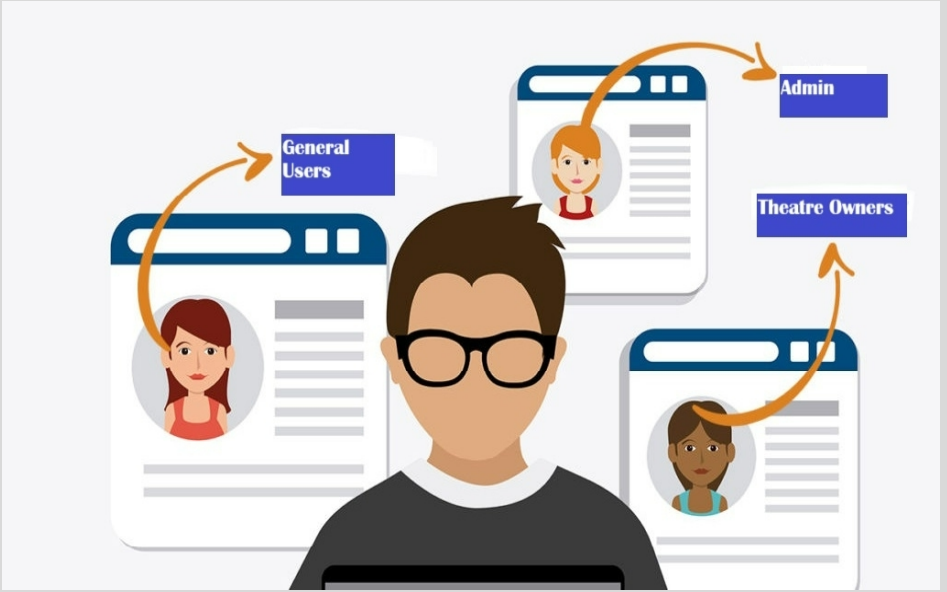
This structure allows us to leverage the strengths of both architectural models to enhance scalability and maintain development efficiency.

Application Structure

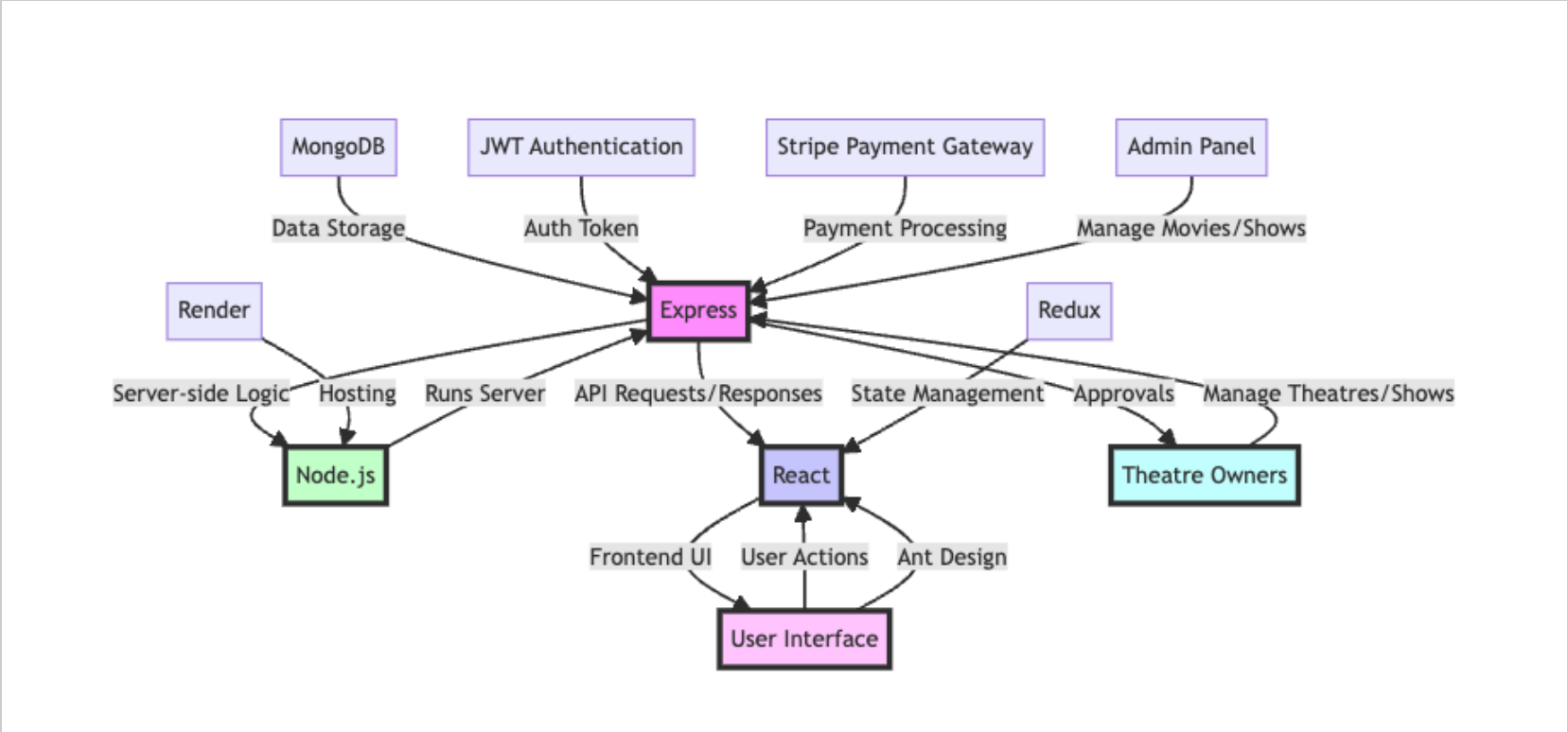
- **Client-Side:** Built with React.js, utilizes Ant Design for UI components and manages state with Redux and Redux Toolkit.
- **Server-Side:** Uses Express.js running on Node.js to handle API requests, connect with MongoDB, and manage server logic.
- **Database:** MongoDB for data storage with complex queries, schemas, and models designed to handle movie, theatre, and booking data.

User Roles and Interfaces

- **General Users:** Interface for browsing movies and booking tickets.
- **Admin:** Interface to upload movies, manage theatre owners, and oversee application settings.
- **Theatre Owners:** Interface to add theatres and manage shows after receiving admin approval.



Application Architecture Diagram



Architecture Style

Hybrid architecture in software design blends elements from various architectural styles. This approach leverages the strengths of each style while mitigating their limitations.

Architecture Style:

- Combines event-based and request-based paradigms.
- **Frontend:** Uses traditional request-response cycles for communication.
- **Backend:** Incorporates event-driven patterns for asynchronous operations like notifications and background jobs.

Key Characteristics:

- **Performance:** Non-blocking I/O with React.js and Node.js ensures high throughput.
- **Scalability:** Modular design and MongoDB's horizontal scaling enhance the system's ability.
- **Security:** Robust security with layered authentication and authorization.
- **Responsiveness:** React.js provides fluid and interactive user interfaces.
- **Fault Tolerance:** Redundancy and failover mechanisms minimize downtime and maintain service continuity.

Software Design Principles

- **Modularity:**
  - **Frontend:** Reusable React.js components, utilizing Ant Design for UI consistency.
  - **Backend:** Express modules manage distinct functionalities like user authentication and data processing.
- **Separation of Concerns:** Clear functional division enhances code maintainability and readability.
- **Bounded Context:** Well-defined component responsibilities, ensuring efficient management of user roles and application features.
- **CI/CD Pipelines:** Automated building, testing, and deployment using Docker and GitHub workflows.
- **Iterative Development:** Agile development with regular sprints and stakeholder feedback for continuous improvement.

Challenges Faced

1) Handling concurrent bookings where multiple users attempt to book the same seat simultaneously.
2) Ensuring that users can securely log in and access only their bookings and profiles, while providing different access levels for administrators and regular users.
3) Integrating third-party payment gateways.

Solution:

1) Used a locking mechanism
2) Implemented JWT (JSON Web Tokens) for secure, stateless authentication
3) Used secure, well-documented APIs provided by payment gateways

Lessons Learned

- **Real-Time Data Handling:** Learned the importance and techniques of managing real-time data interactions, crucial for features like live seat selection.

- **Agile methodology:** Gained insights into the critical role of sprint planning, retrospectives and testing in application development. Learned to incorporate user feedback early and often through iterative sprints, which significantly improved the usability and aesthetics of the application.

- **Security Practices:** Learned to implement and prioritize security measures, especially concerning user data and transaction security. This included using JWTs for secure authentication and HTTPS to protect data in transit.

Improvements

1) **Refined User Interface**
   Including more interactive elements could enhance the user interface and make the booking experience more engaging.

1) **Microservices Approach:**
   This would allow for better scalability, easier maintenance, and faster deployment of new features. Each service could be developed, deployed, and scaled independently.


THANKYOU