CS731 COURSE ASSIGNMENT-PROJECT

MyTicket.com Blockchain Ticket Managment System

Divyansh Chaurasia
241110022

Ansh Makwe
241110010

Introduction

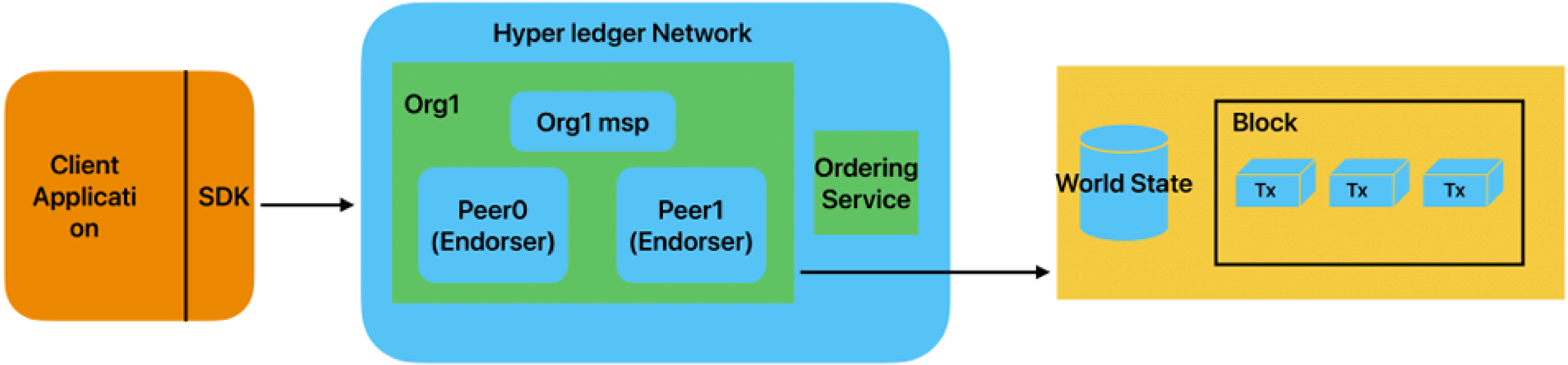
- MyTravel.com is blockchain based ticket management built for customers and travel agencies
- The system integrates Hyperledger Fabric technology as the core backend

HYPERLEDGER FABRIC

Objectives

- Provides secure and transparent ticket booking with dynamic pricing.
- Prevents common pitfalls like overbooking and double bookings.
- Incorporates blockchain features such as identity management, ledger consistency, and block confirmations.
- Ensures privacy between customers and service providers.
- Simulates a dummy payment system to validate transactions on the blockchain.

System Architecture



- There are two peers in our architecture namely peer0 and peer1 with peer0 being the anchor peer, peer1 also maintains the replicated copy of ledger and can also endorse transactions, ultimately improving fault tolerance
- Also, there is a single orderer sufficient for single organization architecture

Channel Configuration

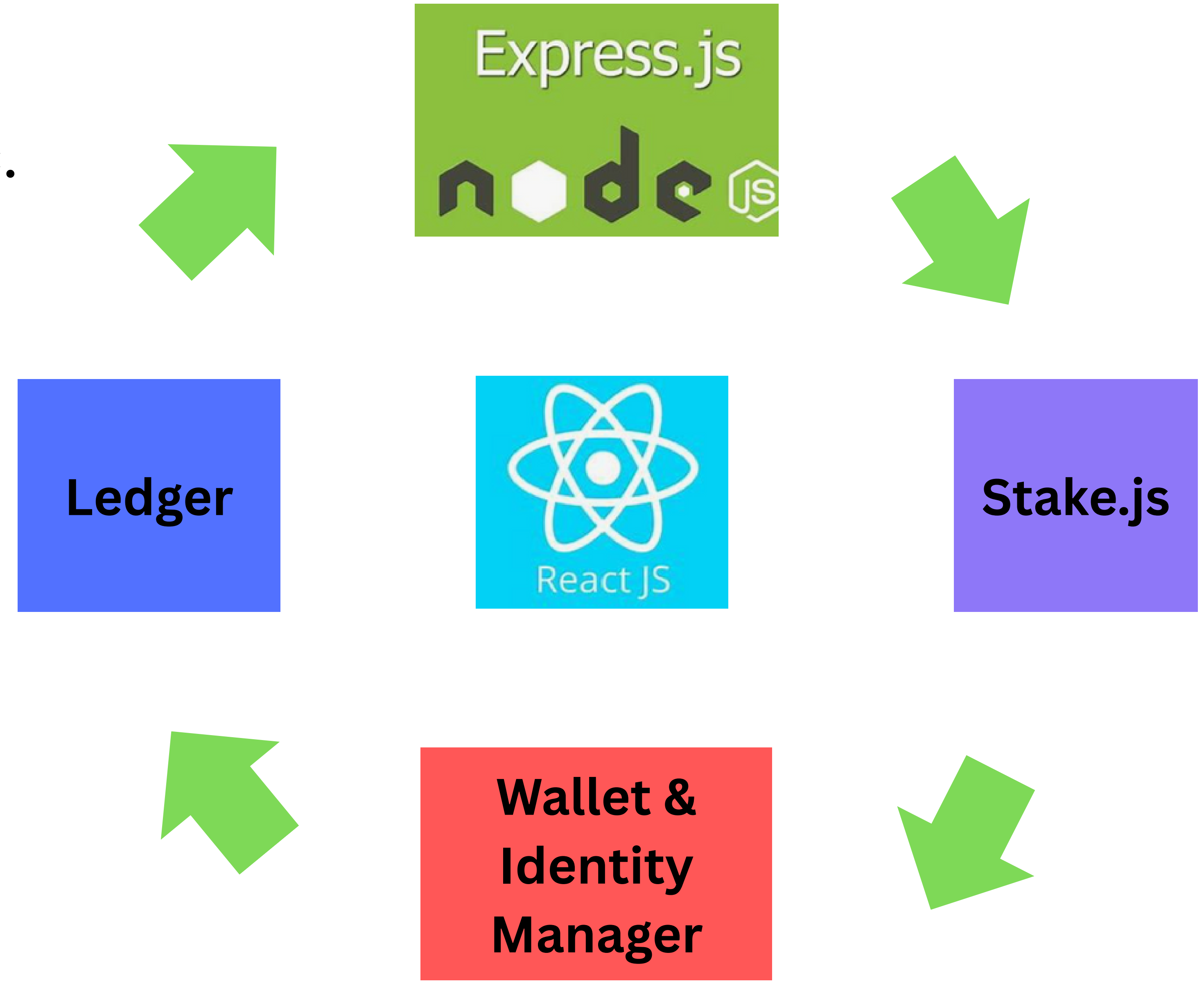
- Single channel named *mychannel* is used for all transactions
- Both customers and providers interact through this channel

Chaincode

Implemented in JavaScript within the file named *stake.js. The chaincode includes all the logic for :*

- Customer Registration
- Provider Registration
- Travel Option Creation
- Ticket booking, cancellation, rescheduling
- Payment simulation and dynamic pricing, etc.

System Architecture: platform is composed of



Requirements Mapping

Functional Requirements	Non-functional Requirments
Ticket Booking (Source to Destination)	Consistency, Robustness, Transparency
Cancel an Existing Ticket	Scalability and Performance
Change Date of Travel (Reschedule)	Double Booking and Overbooking Prevention
Show 3+ Options	Show 3+ Options
Provider Onboarding and Deletion	
Sorting by Provider Rating, Mode, Price	
Booking Details Verification	
Dynamic Pricing	
Update Passenger and Provider Details	

Deliverables

Minimum Deliverables	Extra Points
User Creation	Filtering by Price, Availability and Provider
Minimal Web UI	System Deployed Online
Provider Creation	Ease of Ticket Verification
Invisible Bookings	Seat Selection for Customer
Ticket Creation	Rating Option
Authenticating service providers	Automatic Ticket Confirmation
Payment + Ledger Update	
Ticket Deletion	
List Self Tickets	
List Transport Options	
Provider Info Privacy, Verifying Booking Details	
Customer Info Privacy, Tackling Bots	

Business Logic Implementation

Dynamic Pricing:

$$\text{dynamicPrice} = \text{basePrice} \times \left(1 + \frac{\text{bookedCount}}{\text{seatCapacity}} \times 0.5\right)$$

- Dynamic Price is capped to a maximum of 1.5 x base price
- And the final charged cost = dynamic price + 5rs, where extra 5 rs is charged as platform fees

Ticket Confirmation and Block Based Logic:

- **confirm ticket:** a ticket gets the status “CONFIRMED” when two blocks gets added into the chain after its own block.
- **autoConfirmScheduler:** automatically confirms a ticket if no two block gets added, departure time is within 2 min and seats are available.

Cancellation of Tickets:

Refunds are processed on the time difference from the departure time

- For full refund, time diff should be within 48 hrs
- For 80% refund time diff should be between 24 to 48 hrs
- Otherwise, there will be no refund processed

What if Departure Date is Passed?

Customers’ Perspective:

- ListTravelOptions: will jot show tickets with departure time < current time
- Customer will also not able to book a ticket if departure time < current time

Provioders’ Perspective:

- AddTravelOptions: Customer will not be able to add travel option with departure time < current time
- CancelTravelListing: If a provider cancels a travel listing after departure date, no refund will be processed
- DeleteProvider: If a provider deletes its profile, no refund will be processed for the listing with departure time < current time

Non Functional Requirements in Depth

Showing Three Plus Options

- In the demo, we have a setup where we have 400 travel options that are in generated random configuration and cover this constraint.
- However, in our Business logic we have not forced a provider to add three plus options.

Scalability and Performance

- Our product is indeed scalable for 1000+ daily active users with 2500+ bookings.
- Detailed Testing and Analysis will be discussed in further slides.

Consistency, Robustness and Transparency

- Consistency: Endorsement policies and chaincode logic ensure each transaction is validated prior to commit.
- Robustness and Transparency: Full transaction history is on the immutable ledger; tickets are confirmed only after ledger finality (two subsequent blocks).
- Also, the dynamic pricing details are displayed in the ticket for transparency.

Double Booking Prevention

Per-seat existence check:

- On every bookTicket(...), the chaincode iterates existing tickets for that travelOptionId (via getStateByPartialCompositeKey(ticketPrefix, [travelOptionId]))..
- It throws if any non-cancelled ticket already has the requested seatNumber.

In-memory bookedSeats guard:

- After reading the stored travelOptionData, we also keep a bookedSeats array.
- Before issuing a new ticket, we check if (bookedSeats.includes(requestedSeatNumber)) throw ....

MVCC concurrency control:

- Fabric’s MVCC ensures that if two clients try to book seat say 28 at the same time, only one transaction can commit; the other sees a version conflict and must retry.

Atomic composite-key uniqueness:

- Each ticket composite key includes travelOptionId, customerId, and a timestamp, so writes don’t stomp each other.

Overbooking Prevention

AvailableSeats counter:

- At booking start we check if (availableSeats <= 0) throw ....
- On a successful booking, we decrement availableSeats by one—atomically, in the same transaction that creates the ticket.

Auto-confirmation with capacity enforcement:

- As departure nears (within 2 minutes), the scheduler calls autoConfirmTicketsForTravelOption, which:
    - Confirms up to the remaining AvailableSeats.
    - Cancels (and refunds) any extra pending tickets beyond capacity, so you never end up with more “confirmed” seats than exist.

Capacity never goes negative:

- Because the availableSeats > 0 check and the decrement happen inside one Fabric transaction, you can’t push availableSeats below zero.

What if changes one tries to Redeploy an upgraded Chanincode?

Persistence:

We provide two scripts to start the system, first.sh and
second.sh where scond.sh will be used for redeploying an upgraded chaincode

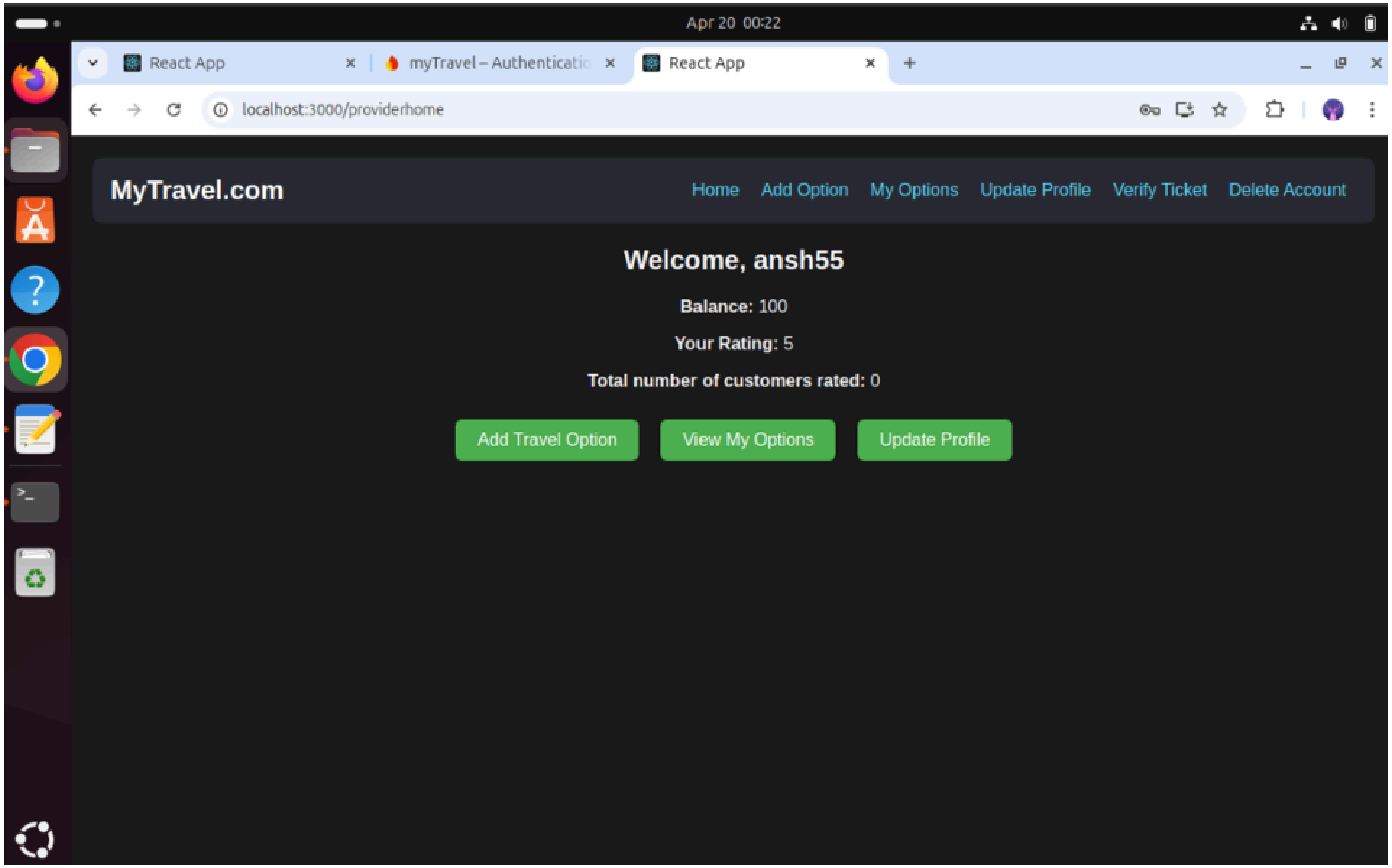
Data preservation:

Upgrades preserve existing identities and on-chain
data, first-time wipes state.

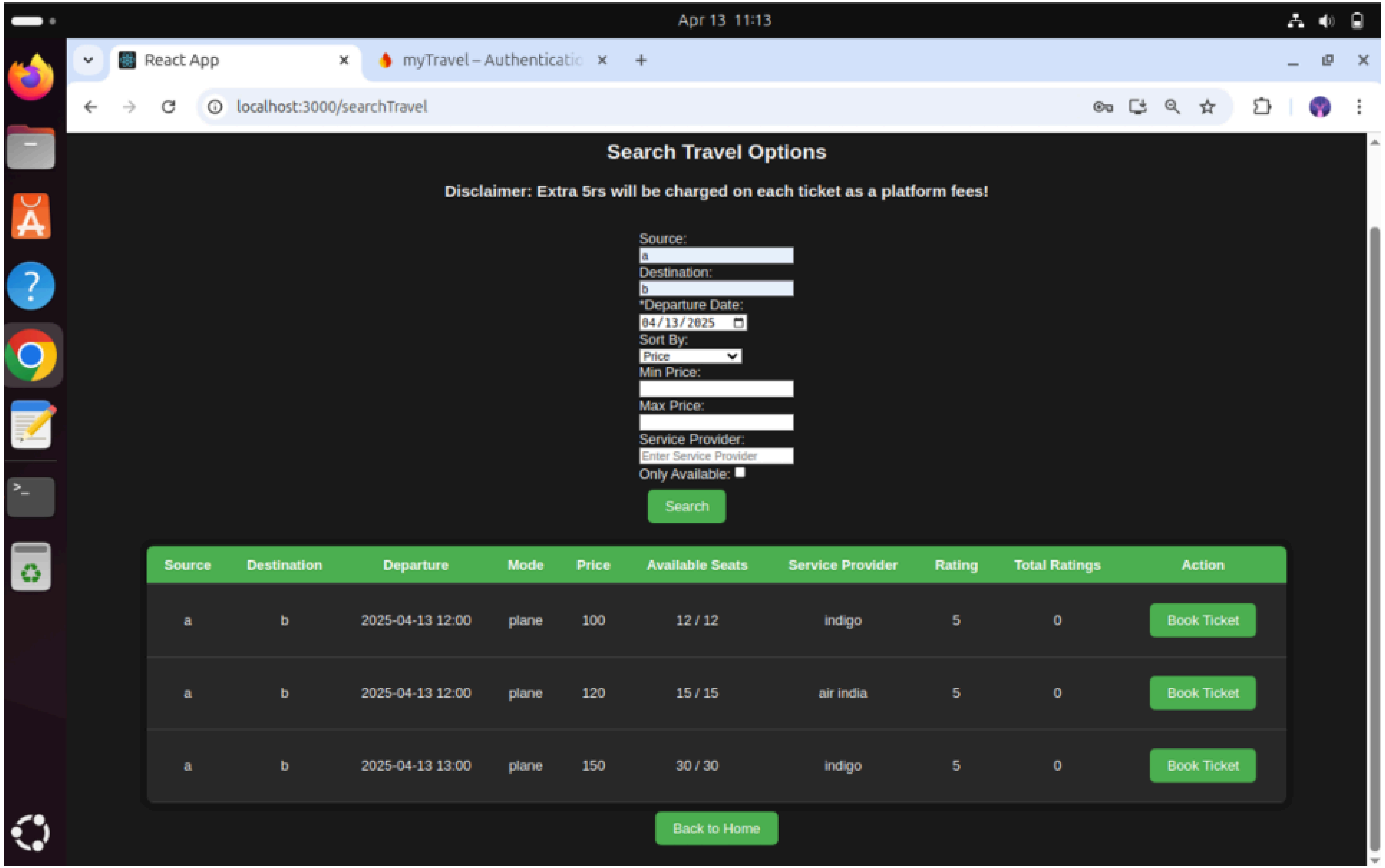
Efficiency:

Reduces deployment time after chaincode update.

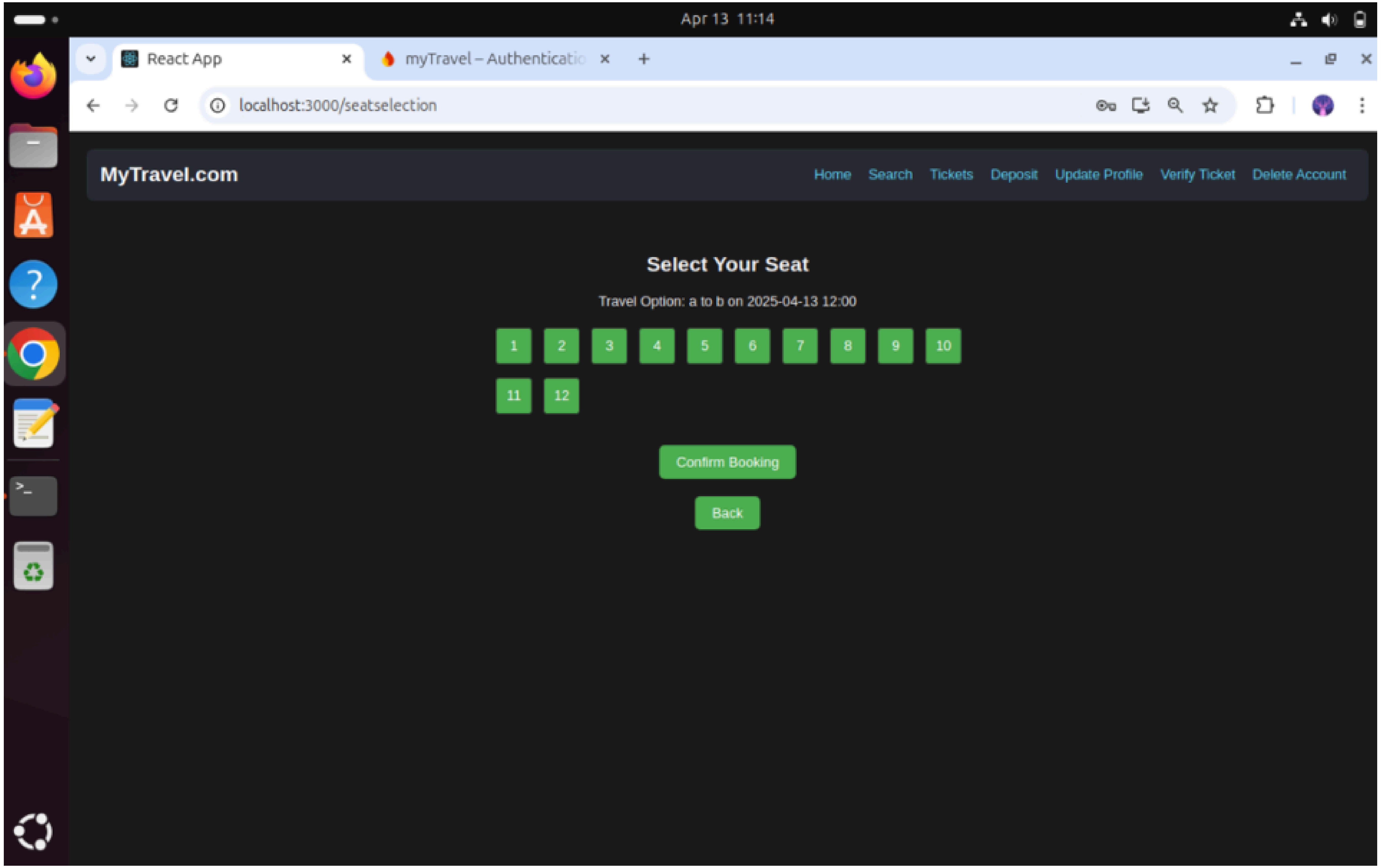
Frontend Componenets Overview



Home Screen



Search Travel Options



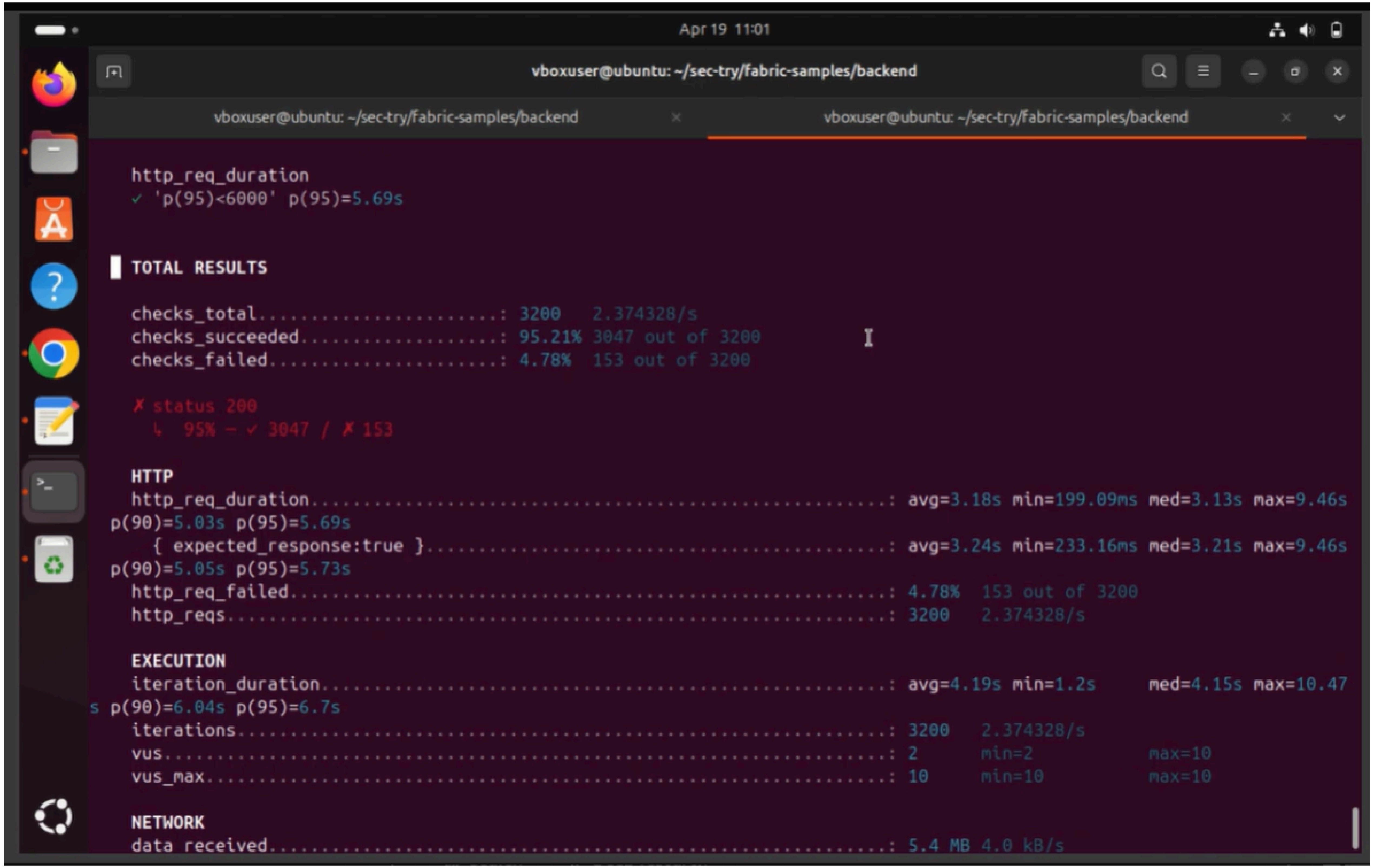
Seat Selection Screen

Testing Results

For testing normal endpoints, normal node scripts were used to call the end point,s and their behaviour was observed to confirm that they are working correctly.

Testing Objectives:

• Throughput: 2500 booking operations within 24 h
• Concurrency: up to 10 virtual users (VUs)
• Latency SLA: 95th-percentile http_req_duration < 6000 ms
• Reliability: 99% successful bookings (i.e. HTTP 200)



Had it been a one day scenario

- Observed sustained rate = 2.37 req/s
- Seconds per day = 86400 s
- Total requests per day = 2.37 × 86400 ≈ 205,000 req/day
- Success rate = 95%
- Successful bookings per day = 0.95 × 205,000 ≈ 195,000 bookings/day

Current Limitations

1. Scalability to Very High Loads– The chaincode is not heavily optimised for extremely large queries. If we had 1 million daily bookings, we might need advanced indexing or partitioning.

2. Latency- The test on our laptop showed high latency; a better machine might provide better latency, but this shows a direction of potential optimization. One could use dynamic load balancers.

3. Simplicity of the Payment Model– We store balance fields in the ledger for each user. In production, we might integrate a real payment gateway or use a token-based approach.

Potential Future Enhancements

- **Multi-Org Deployment– Allow different providers from different organizations to manage their own peers, increasing trust boundaries.**

- **Advanced Dynamic Pricing Strategies– Incorporate real-time demand predictions, surge pricing, or AI-driven seat pricing.**

- **Microservice Payment Integrations– Connect to a real-time payment gateway, bridging off-chain fiat transactions with on-chain booking confirmations.**

- **Comprehensive Analytics– Providers may want dashboards of seat utilization, revenue, or user demographics (with user consent).**

Workload Distribution

241110010 Ansh Makwe	241110022 Divyansh Chaurasia
Enroll admin and authentication	Register user and provider
Update customer/provider details	Get customer/provider details
Deposit balance	Get customer tickets
Delete customer/provider	Book ticket and cancel ticket
Reschedule ticket and cancel ticket	Get ticket details and auto-confirm ticket
Get provider details and travel options	Add travel option/delete travel option
Cancel travel listing	List and filter travel options
Rate provider	Dynamic pricing for tickets
Build user/provider home page	Build login page with Firebase auth integration
Register user/provider page	Form page for adding travel options
Page to book ticket/list tickets	Reschedule ticket interface
Cancel ticket interface	Block confirmation logic integration
Preventing overbooking	Preventing double booking and

Conclusion

This assignment successfully implements a Blockchain-based Ticket Management System using Hyperledger Fabric. Through a single organization setup with two peers and one orderer, we have demonstrated:

- Secure and transparent ledger-based seat allocation.
- Role-based operations for customers (ticket purchase, cancellation) and providers (adding routes, canceling listings).
- Dynamic pricing influenced by seat occupancy rates.
- Refund logic that factors in the remaining time before departure.
- Block-based finality to confirm tickets after at least two subsequent blocks.
- A fully functional front-end (React) and a Node/Express back-end that orchestrates chaincode calls and identity management.
- Other essential and extra functionalities.

Acknowledgments

We express our gratitude towards Professor Dr. Angshuman Karmakar Sir for giving us this opportunity to work on this assignment. Also, we express our gratitude towards Er. Sumit Lahiri Sir for guiding us throughout the development of this assignment and product.