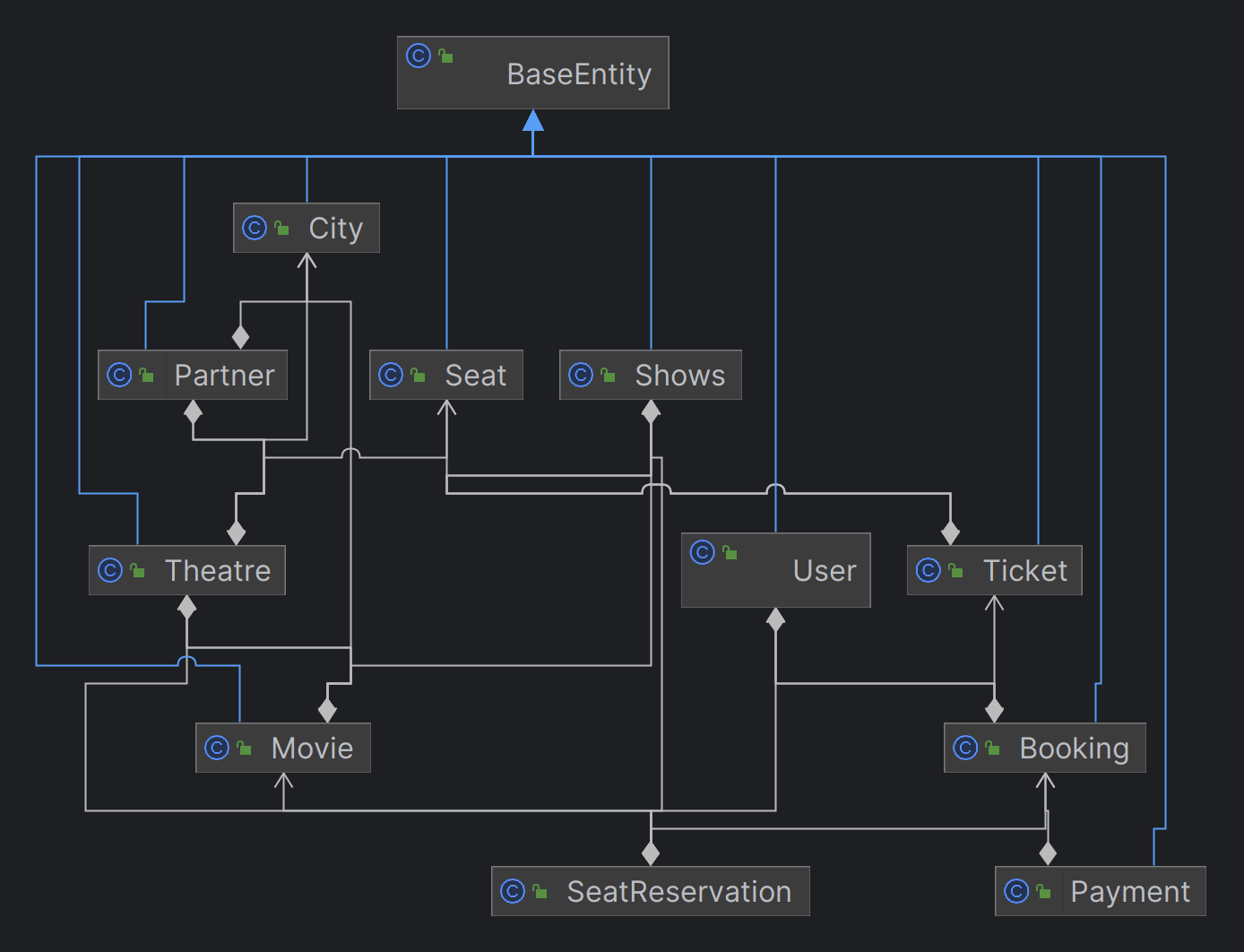
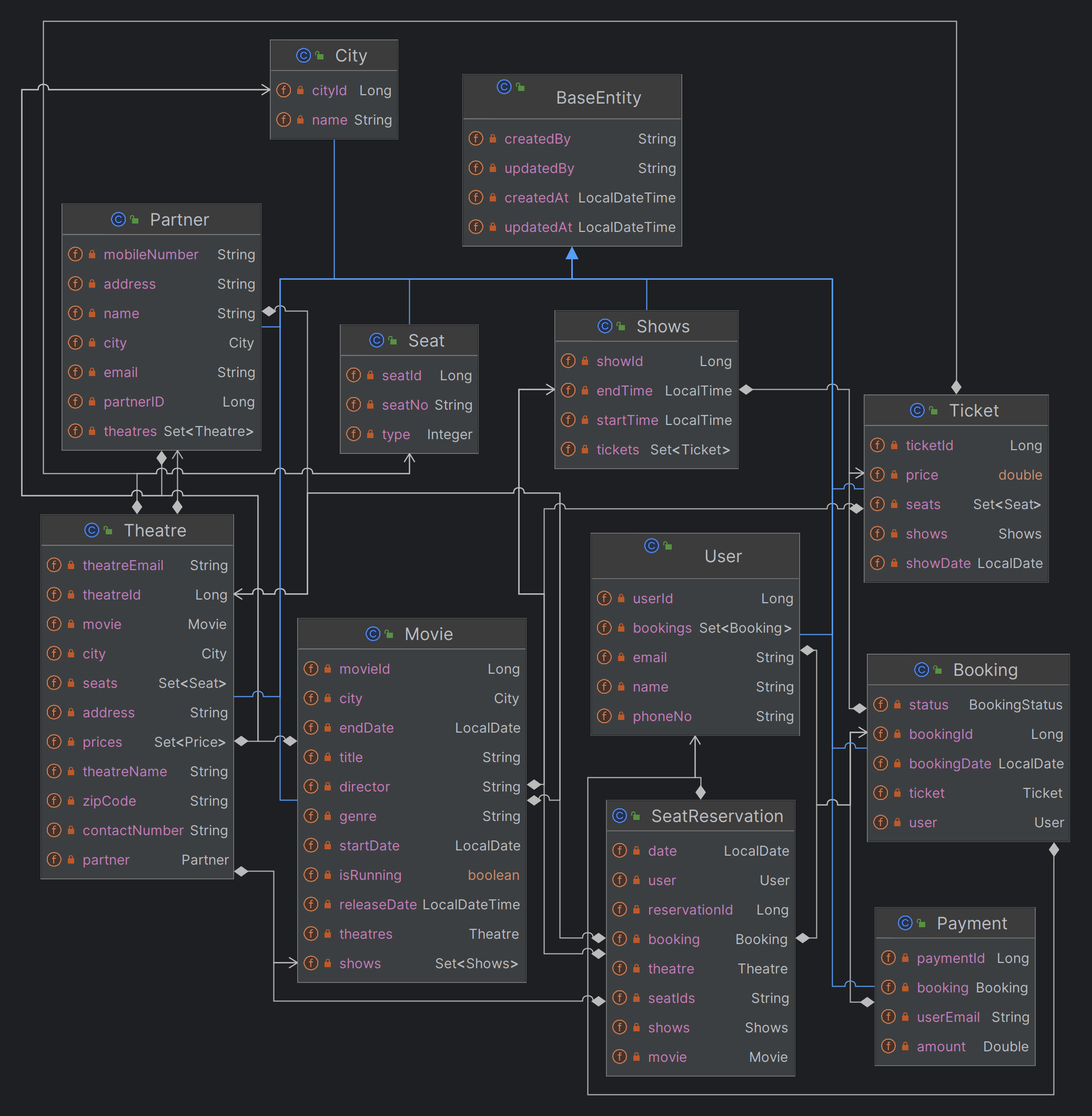
**BookMyMovie architecture and design decisions**

**Data Model**



**Data Model with Attributes**



**Basic Platform Architecture.**

APIs

**Controller**

Application Logic

Request Mapping

Services

Repositiry

Mapper

DB

DTO

Model

**APIs Contract**

**Partner Onboarding APIs**

1. Method -POST

<http://localhost:8080/bookmymovie/app/partner/onboarding>

Creates a partner that can later be associated with theatre and movies

1. Method -GET

<http://localhost:8080/bookmymovie/app/parners>

fetches the list of partners

1. METHOD- GET

[http://localhost:8080/bookmymovie/app/partners/{id}](http://localhost:8080/bookmymovie/app/partners/%7bid%7d)

fetches the partner with the corresponding id.

**Theatre Management APIs**

1. Method - POST

<http://localhost:8080/bookmymovie/app/theatre>

Creates a Theatre

1. Method- GET

<http://localhost:8080/bookmymovie/app/theatres>

fetches the list of all theatres

1. Method -GET

[http://localhost:8080/bookmymovie/app/theatre/{id}](http://localhost:8080/bookmymovie/app/theatre/%7bid%7d)

fetches the details of theratre with corresponding id

**Movie Management APIs**

1. Method - POST

<http://localhost:8080/bookmymovie/app/movie>

Creates a Movie in a theatre with all the shows

1. Method- GET

<http://localhost:8080/bookmymovie/app/movies>

fetches the list of all Movies

1. Method -GET

[http://localhost:8080/bookmymovie/app/movies/{id}](http://localhost:8080/bookmymovie/app/movies/%7bid%7d)

fetches the details of movie with corresponding id

1. Method- GET

<http://localhost:8080/bookmymovie/app/movies/search>

Searches the movies based on movie name,date and city name passes as path parameters

**Booking Management APIs**

1. Method - POST

<http://localhost:8080/bookmymovie/app/booking>

Books a movie based on movieId,theatreId,,showId,userID,list of seatsId and booking date

1. Method- POST

<http://localhost:8080/bookmymovie/app/booking/ticket>

creates ticket for the booking

1. Method -GET

[http://localhost:8080/bookmymovie/app/booking/status/{id}](http://localhost:8080/bookmymovie/app/booking/status/%7bid%7d)

fetches the status of the corresponding bookingid

**Functional Features**

**Read Scenario**

**Anyone of the following read scenarios:** (Only Service Implementation needed/ No UI required)

* Browse theatres currently running the show (movie selected) in the town, including show timing by a chosen date

API for browsing the theatres currently running the show.

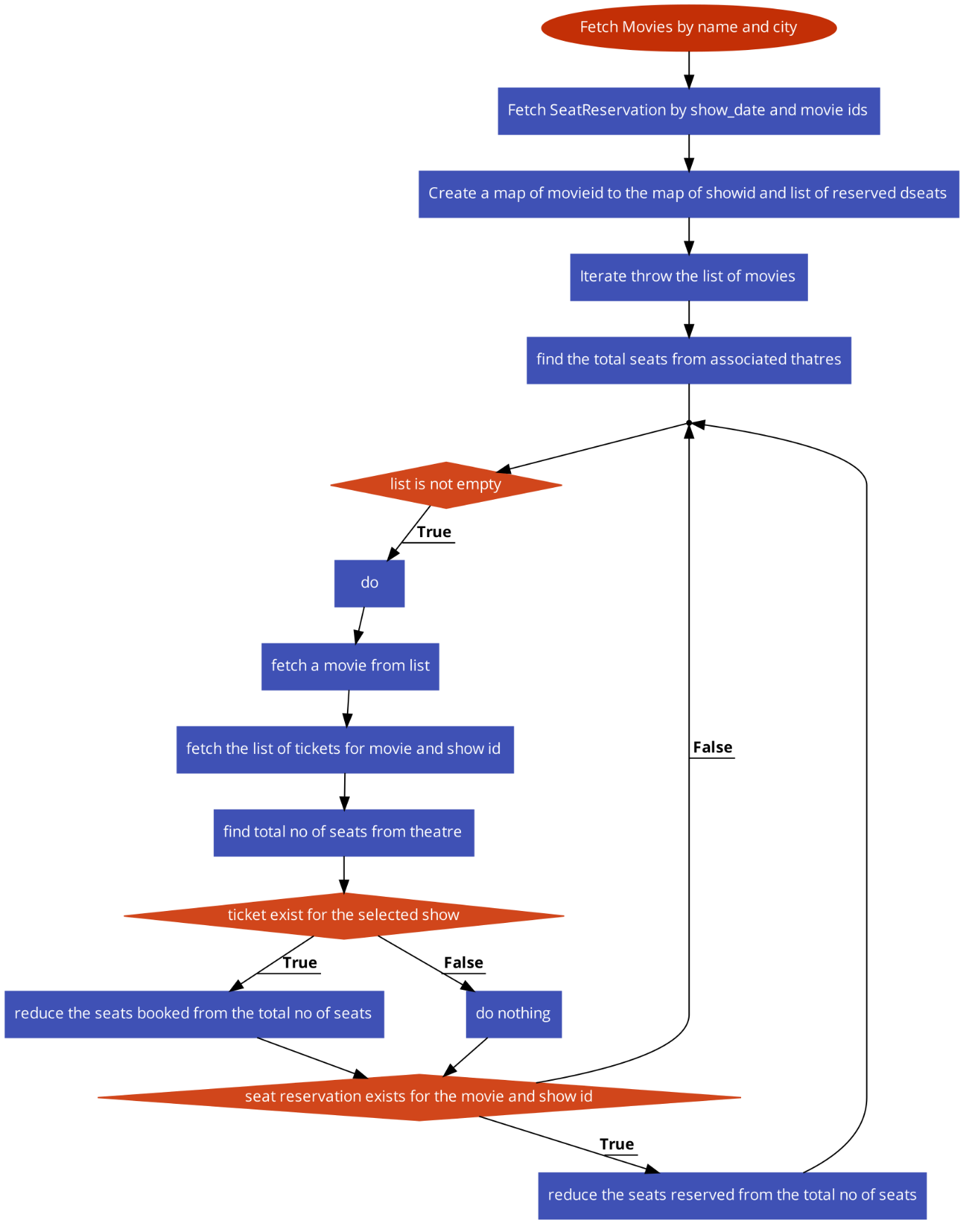
/bookmymovie/api/movies/search?name=’incep’&city=’noida’ & date=’2023-12-01’

Sequence Diagram :

A diagram of a project

Description automatically generated with medium confidence

Flow diagram of logic:



**Anyone of the following write scenarios:** Good to have - Code Implementation (write scenario):

* Book movie tickets by selecting a theatre, timing, and preferred seats for the day

Booking of Movie ticket happens in multi steps

1. Client calls the booking api
2. Booking api resrves the seats, generates an event in RabbitMQ for notification to the payment service and immediately returns the response to the client.
3. On other side notification subscriber consumes the event and calls the payment service.
4. Payment service process the payments and publishes an event for the booking consumer.
5. Booking consumer consumes the event and calls the booking service .
6. Booking service creates a ticket if the payment is successful.
7. If the payment is not successful then updates the booking status as failed .

Sequence diagram:

A diagram of a movie ticket

Description automatically generated

Sequence diagram of BookingController

A diagram of a application

Description automatically generated

Sequence diagram of Payment api

A diagram of a payment process

Description automatically generated

**Discussion topics & Logical View:**

**Non-functional requirements-(**Mandatory **-**Design/Arch solution & Optional Implementation**):**

* Describe transactional scenarios and design decisions to address the same.
* Integrate with theatres having existing IT system and new theatres and localization(movies)
* How will you scale to multiple cities, countries and guarantee platform availability of 99.99%?
* Integration with payment gateways
* How do you monetize platform?

How to protect against OWASP top 10 threats

* **Describe transactional scenarios and design decisions to address the same.**

**Booking Tickets:**

**Transactional Scope**: When a user books tickets, it involves multiple operations, such as reserving seats, updating seat availability, and recording the booking details.

**Design Decisions:**

Use database transactions to ensure atomicity, consistency, isolation, and durability (ACID properties).

Employ optimistic concurrency control or locking mechanisms to avoid double booking of seats.

Incorporate compensating transactions to handle failures during the booking process.

A diagram of a software application

Description automatically generated

**Cancellation of Tickets:**

**Transactional Scope**: Cancelling a booking requires seat inventory to be updated, and the cancelled seats made available again.

**Design Decisions:**

Use transactions to ensure that cancelled seats are released and made available for new bookings.

Implement logic to prevent cancellation of tickets after a specific time limit before the show.

**Bulk Booking and Cancellation:**

**Transactional Scope**: Bulk operations involve a larger number of transactions to book or cancel multiple tickets simultaneously.

**Design Decisions:**

Implement batch processing with proper transaction boundaries to handle bulk booking or cancellation requests.

Ensure optimized database queries and processing to handle larger transactions efficiently.

**Allocation and Update of Seat Inventory:**

**Transactional Scope**: Managing seat inventory involves updating seat availability for each show in different theatres.

**Design Decisions:**

Employ consistent transactional updates across distributed systems for seat allocation and updates.

Use distributed caching mechanisms for frequent seat inventory access to minimize database hits and enhance performance.

**Show Creation, Update, and Deletion by Theatres:**

**Transactional Scope**: Theatres creating, updating, or deleting shows involves modifying the show schedules and seat allocations.

**Design Decisions:**

Implement ACID transactions to maintain data integrity while creating, updating, or deleting shows.

Use event sourcing or logging mechanisms to track show changes for auditing purposes.

**Payment Processing:**

**Transactional Scope:** Processing payments for booked tickets involves ensuring successful financial transactions.

**Design Decisions:**

Utilize secure and reliable payment gateways to handle payment transactions.

Implement idempotent payment processing to handle duplicate or failed transactions without side effects.

* **Integrate with theatres having existing IT system and new theatres and localization(movies)**

1. **API-based Integration:**
   * Develop APIs and Provide well documented APIs for functionalities like ticket booking, cancellation, seat inventory etc.
   * Support various data formats like JSON,XML and provide authentication mechanism like OAuth,API key etc.
2. Legacy System Compatibility:

* Analyze and understand the existing data formats
* Implement middleare or adapters to translate data between different formats.
* Data Migration strategies

1. Customization and Flexibility:

* Provide configuration options to map data fields and accommodate specific requirements of each theatre's IT system.
* **How will you scale to multiple cities, countries and guarantee platform availability of 99.99%?**

**Scaling to Multiple Cities and Countries:**

**1.Microservices Architecture:**

* Implement a microservices-based architecture that enables independent deployment and scaling of services.
* Decompose functionalities into smaller, manageable services like booking, theatres, payments, and notifications.

**2.Horizontal Scaling:**

* Utilize container orchestration tools (e.g., Kubernetes) to scale services horizontally based on demand.
* Distribute services across multiple nodes or clusters to handle increased traffic from various cities or countries.

**3.Global Content Delivery:**

* Employ Content Delivery Networks (CDNs) to cache static content like movie information, images, and UI elements closer to users globally.
* Utilize edge caching to reduce latency and deliver content efficiently across different regions.

**4.Distributed Data Management:**

* Use geo-distributed databases or database sharding techniques to manage data across multiple regions.
* Perform sharding based on city id, partnerID
* Leverage Caching Solution. Create a dependency graph like DAC graph and find out which set of data can be fetched concurrently. Distribute the data on cache clusters pool to serve different set of key value pair

**Ensuring Platform Availability (99.99%):**

**1.Redundancy and Fault Tolerance:**

* Avoid Single point of failure
* Deploy services across multiple availability zones (AZs) in cloud providers to mitigate failures in a single zone.
* Implement redundancy for critical components (load balancers, databases) to ensure fault tolerance.

**Automated Monitoring and Healing:**

**1.Comprehensive Monitoring:**

* Implement robust monitoring tools (e.g., Prometheus, Grafana) to track system performance, latency, and availability across multiple locations.
* Set up alerts and automated responses for detecting and mitigating issues in real-time.

**2.Scheduled Maintenance and Updates:**

* Plan scheduled maintenance windows during off-peak hours to perform updates and maintenance tasks.
* Use blue-green deployments or canary releases to minimize downtime during updates.

**3.Fault Isolation and Recovery:**

* Implement circuit breakers and graceful degradation techniques to isolate failures and prevent system-wide crashes.
* Design services to automatically recover from failures or switch to alternative resources.

**4.Disaster Recovery and Backup:**

* Set up disaster recovery mechanisms to restore the system in case of catastrophic failures.
* Regularly back up critical data and configurations to ensure quick recovery in case of data loss or corruption.

**5.Continuous Testing and Deployment:**

* Implement Continuous Integration/Continuous Deployment (CI/CD) pipelines for automated testing and deployment.
* Conduct load testing, stress testing, and failover simulations to validate system performance and resilience.

**6. Proactive Maintenance and Updates:**

* Schedule maintenance windows during low-traffic periods to apply updates, patches, and perform system optimizations.
* *Use blue-green deployments or canary releases to minimize downtime during updates.*

**Possible basic AWS architecture of application:**

A diagram of a diagram

Description automatically generated

* **How do you monetize platform?**

**1.Commission on Ticket Sales:**

* Transaction Fees
* Tiered Pricing
* Exclusive Deals

**2.Premium Features and Services:**

* Subscription Models
* VIP Services

**3.Advertising and Partnerships:**

* Sponsored Content:
* Partnerships with Brands:

**4.Data Monetization and Insights:**

* Analytics Services:
* Targeted Marketing

**5.Ancillary Revenue Streams:**

* Convenience Fees
* Affiliate Programs:

**6.Platform White-Labeling:**

* Platform as a Service (PaaS):
* Customized Solutions:

**7.Subscription-Based Business Models:**

* Subscription for Theatre Partners
* Enterprise Solutions:

**8.Value-Added Services:**

* Customer Insights Reports
* Marketing Campaigns:

**Discussion topics & Logical View:**

**Platform provisioning, sizing & Release requirements:** (Mandatory-Architecture artifacts)

* Discuss your technology choices and decisions through key drivers
* Discuss database, transactions, and data modelling.
* Discuss enterprise systems that you may need to manage specific areas.
* Discuss hosting solution and sizing (Cloud / Hybrid/ Multi cloud)- Any
* Discuss release management across cities, languages etc
* Provide details on monitoring solution
* Discuss overall KPIs
* Create a high-level project plan and estimates breakup.
* **Discuss your technology choices and decisions through key drivers**

**Key Drivers for Technology Choices:**

**Scalability**:

* Key Driver: Need for a scalable architecture to handle varying loads during peak booking times or new theatre onboarding.
* Technology Choices: Microservices architecture, cloud-based hosting (AWS, Azure, Google Cloud), Kubernetes for container orchestration.

**Performance and Responsiveness:**

* Key Driver: Requirement for a responsive and high-performing platform to ensure a seamless user experience.
* Technology Choices: Java with Spring Boot for backend services, React or Angular for frontend development, caching mechanisms (Redis or Memcached), CDN integration.

**Flexibility and Agility:**

* Key Driver: Need for flexibility to adapt to changing user demands and market dynamics.
* Technology Choices: Agile development methodologies, DevOps practices, CI/CD pipelines for rapid and continuous deployment.

**Security and Compliance:**

* Key Driver: Ensuring data security, privacy, and compliance with industry regulations.
* Technology Choices: Encryption protocols, OAuth for authentication, HTTPS for secure communication, compliance with GDPR, PCI DSS, or other relevant standards.

**Interoperability and Integration:**

* Key Driver: Integration with theatres' existing systems and other third-party services.
* Technology Choices: RESTful APIs, message brokers (Kafka or RabbitMQ) for asynchronous communication, data serialization formats like JSON or XML.

**User Experience and Innovation:**

* Key Driver: Focus on enhancing user experience and offering innovative features.
* Technology Choices: Machine learning or recommendation systems for personalized movie suggestions, chatbots for customer support, progressive web app (PWA) for improved mobile experience.

**Technology Decisions:**

**Backend Development:**

* Decision: Java with Spring Boot for robust and scalable backend services.
* Rationale: Strong ecosystem, mature frameworks, scalability, and extensive community support.

.**Database:**

* Decision: Relational database (e.g., PostgreSQL) for structured data and scalability.
* Rationale: ACID compliance, relational data modeling, support for complex queries.

**Cloud Hosting:**

* Decision: AWS for cloud hosting due to its wide range of services and scalability options.
* Rationale: Elasticity, global reach, managed services, and availability of serverless computing (Lambda).

**Development Methodology:**

* Decision: Agile methodology (Scrum or Kanban) for iterative and adaptive development.
* Rationale: Flexibility, collaboration, incremental delivery, and responsiveness to changes.

**Security Measures:**

* Decision: Implement OAuth for authentication, encryption for sensitive data, and regular security audits.
* Rationale: Strong authentication, data protection, and compliance with security standards.
* **Discuss hosting solution and sizing (Cloud / Hybrid/ Multi cloud)- Any**

**Cloud Hosting (Single Cloud Provider):**

**Advantages:**

**Scalability**: Easily scale resources up or down based on demand.

**Global Availability**: Cloud providers offer data centers in multiple regions for global reach.

**Managed Services**: Access to managed services for databases, containers, and various tools.

**Cost Savings:** Pay-as-you-go pricing model helps optimize costs.

**Considerations:**

**Vendor Lock-in**: Potential dependency on a single cloud provider.( Kubernetes can be a solution?)

**Performance Variability**: Network latency can affect performance in different regions.(CDN can be a solution?)

**Security and Compliance**: Ensure adherence to security standards across regions.

**Sizing Considerations:**

**Resource Allocation:**

* Compute: Allocate sufficient resources for handling peak loads during ticket sales or promotions.
* Storage: Plan for scalable storage solutions to accommodate growing data volumes.
* Networking: Ensure robust and scalable networking infrastructure to handle user traffic.

**Performance and Scaling:**

* Auto-scaling: Implement auto-scaling mechanisms to dynamically adjust resources based on demand.
* Load Balancing: Employ load balancers for even distribution of traffic across instances.
* Caching Strategies: Use caching mechanisms to improve performance and reduce load on databases.

**Disaster Recovery and Redundancy:**

* Backup and Recovery: Implement regular backups and disaster recovery mechanisms.
* Redundancy: Ensure redundancy across regions or clouds to mitigate failures.

**Cost Optimization:**

* Resource Monitoring: Regularly monitor resource utilization and optimize for cost-effectiveness.
* Reserved Instances: Utilize reserved instances or long-term commitments for cost savings.
* **Discuss release management across cities, languages etc**

**1.Release Planning:**

Feature Prioritization:

Prioritize features or updates based on user needs, market demands, and business objectives.

Categorize features as critical, major, or minor for better planning.

2.**Roadmap Creation:**

Develop a release roadmap outlining the timeline for each feature or update.

Consider seasonality, movie release schedules, and peak booking times for optimal release planning.

**City-wise Release Management:**

**1.Rollout Strategy:**

Implement phased or staged rollout across different cities or regions.

Start with a limited release in specific cities to monitor performance and gather feedback.

**2.User Engagement:**

Communicate upcoming releases or updates to users in targeted cities through notifications or emails.

Encourage user participation and feedback during beta releases in specific cities.

**Release Coordination and Communication:**

**Cross-functional Teams:**

Establish cross-functional teams involving development, QA, operations, and marketing for coordinated releases.

Conduct regular sync-ups and meetings to align on release plans and progress.

**Clear Communication:**

Maintain transparent communication with stakeholders, including theatres, users, and internal teams, about release timelines and features.

Provide release notes or documentation detailing changes for clarity.

**Monitoring and Feedback:**

**Performance Monitoring:**

Monitor platform performance, stability, and user feedback after each release.

Utilize analytics tools to track user behavior, adoption rates, and any issues reported post-release.

**Iterative Improvements:**

Iteratively improve the platform based on user feedback and performance metrics.

Release incremental updates or patches to address issues or introduce enhancements.

**Compliance and Rollback Plans:**

**Compliance Checks:**

Ensure compliance with regulations in different cities or countries before releasing updates.

Validate that updates adhere to local data protection and privacy laws.

**Rollback Plans:**

Prepare rollback strategies or contingency plans in case of unforeseen issues or failures post-release.

Implement versioning and backup systems to facilitate quick rollbacks if required.

* **Provide details on monitoring solution**

**Components of Monitoring Solution:**

**Monitoring Tools:**

* Prometheus: For collecting metrics and alerting based on predefined rules.
* Grafana: Visualization and dashboards for monitoring various metrics.
* ELK Stack (Elasticsearch, Logstash, Kibana): Log management and analysis for error logs and exceptions.
* New Relic or Datadog: Comprehensive application monitoring and performance analytics.

**Key Metrics to Monitor:**

* System Metrics: CPU, Memory, Disk Utilization.
* Network Metrics: Latency, Throughput, Errors.
* Application-Specific Metrics: API response times, Database query times, Booking success rates.
* User Metrics: Active users, Concurrent sessions, User behavior.

**Alerting Mechanism:**

* Configure alerts for critical thresholds using Prometheus Alertmanager or similar tools.
* Set up notifications via email, SMS, or collaboration platforms like Slack or Microsoft Teams.

**Logging and Tracing:**

* Centralized logging using ELK Stack to aggregate and analyze logs for troubleshooting.
* Implement distributed tracing (e.g., Jaeger or Zipkin) to trace and analyze requests across microservices.

**Security Monitoring:**

* Implement Intrusion Detection Systems (IDS) to monitor for suspicious activities or security breaches.
* Perform regular security audits and compliance checks.
* **Create a high-level project plan and estimates breakup.**

**Project Phases:**

Discovery and Planning (2-4 weeks):

* Define project scope, goals, and requirements.
* Conduct market research and competitor analysis.
* Create a detailed project plan and roadmap.

Architecture and Design (4-6 weeks):

* Define system architecture and technology stack.
* Develop database schemas, API contracts, and system design.
* Create wireframes or prototypes for user interfaces.

Development (Varies based on scope and features):

* Implement core functionalities: user authentication, theatre registration, movie listings, ticket booking, etc.
* Develop APIs, microservices, front-end interfaces, and back-end systems.
* Conduct iterative development and testing cycles.

Testing and Quality Assurance (2-4 weeks):

* Perform unit testing, integration testing, and user acceptance testing.
* Address bugs, performance issues, and usability feedback.

Deployment and Release (2-4 weeks):

* Prepare for production deployment.
* Roll out the platform in phases or cities.
* Monitor and stabilize the platform post-release.

Post-launch Optimization and Support (Ongoing):

* Monitor system performance, user feedback, and analytics.
* Implement improvements, updates, and additional features based on user insights.

**Estimated Breakdown:**

Resource Allocation:

* Project Manager: 1
* Developers ( Backend): 4-6
* Quality Assurance/Testers: 2-3
* DevOps Engineers: 1-2 (for infrastructure setup and maintenance)

Timeframe Estimates:

* Discovery and Planning: 2-4 weeks
* Architecture and Design: 4-6 weeks
* Development: 12-16 weeks
* Testing and QA: 2-4 weeks
* Deployment and Release: 2-4 weeks
* Post-launch Optimization and Support: Ongoing

Dependencies and Milestones:

* Milestone 1: Completion of Architecture and Design Phase
* Milestone 2: Completion of Development Phase
* Milestone 3: Successful Deployment and Release
* Milestone 4: Post-launch Optimization and User Feedback Integration

Budget Estimates:

Labor Costs:

* Personnel costs based on resource allocation and duration.
* Estimate based on hourly rates or salaries.

Infrastructure Costs:

* Cloud hosting expenses (AWS, Azure, Google Cloud).
* Licensing fees for third-party tools or software.

Contingency and Miscellaneous Costs:

* Buffer for unforeseen expenses or scope changes.
* Miscellaneous expenses (training, documentation, etc.).

**Product management and Stakeholder management**

* Please talk about stakeholder management instances
  + What decisions and actions were taken for decision closure?
* Overall technology management
* Enabling team and introducing efficiencies
* Delivery planning and estimates
* **Please talk about stakeholder management instances** 
  + **What decisions and actions were taken for decision closure?**

**Stakeholder Engagement Instances:**

**Theatre Partners:**

* Instance: Onboarding theatre partners onto the platform.
* Action Taken: Conducted meetings and workshops to understand their requirements, addressed concerns about integration, and provided training/support for platform usage.

**End Users (Customers):**

* Instance: Introducing a new booking feature.
* Action Taken: Conducted surveys, gathered user feedback, and organized focus groups to understand user preferences. The decision was made to implement a more user-friendly booking interface based on this feedback.

**Development Team:**

* Instance: Choosing the technology stack.
* Action Taken: Facilitated discussions, presentations, and workshops to evaluate different technologies. After careful consideration of scalability, maintainability, and team expertise, the decision was made to use Java with Spring Boot for the backend and React for the frontend.

**Management and Investors:**

* Instance: Deciding on the budget allocation for additional features.
* Action Taken: Prepared cost-benefit analyses, presented potential ROI, and demonstrated the value of implementing certain features. Agreed upon an increased budget for implementing premium membership features based on projected revenue.

**Decision Closure Actions:**

**Clear Communication:**

* Ensured all stakeholders were informed about the decision outcomes promptly.
* Provided detailed rationale and justifications behind each decision to maintain transparency.

**Documentation:**

* Documented decision summaries, including reasoning, alternatives considered, and implications.
* Maintained decision logs to track the decision-making process and outcomes.

**Follow-up Actions:**

* Aligned resources and initiated necessary actions based on the decisions taken.
* Monitored the progress and impact of decisions to ensure they aligned with expected outcomes.

**Feedback and Evaluation:**

* Encouraged stakeholders to provide feedback on decisions' effectiveness.
* Conducted periodic reviews to evaluate the impact of decisions and made adjustments if needed
* **Delivery planning and estimates**

**Delivery Planning:**

**Task Breakdown:**

* Identify and break down project tasks into manageable units.
* Categorize tasks based on development, testing, deployment, documentation, etc.

**Dependency Mapping:**

* Determine task dependencies and sequences.
* Ensure tasks are organized logically and can be executed in parallel whenever possible.

**Resource Allocation:**

* Allocate resources based on task requirements and skill sets.
* Balance workload across the team to ensure efficiency.

**Sprint or Iteration Planning:**

* Divide the project timeline into sprints or iterations.
* Assign tasks to each sprint based on priority and complexity.

**Estimation Techniques:**

**Time-Based Estimates:**

* Use historical data and experience to estimate time for similar tasks.
* Break tasks into smaller units and estimate time for each unit.

**Effort Estimation:**

* Estimate effort in person-hours or person-days required for each task.
* Consider complexity, expertise, and potential challenges.

**Story Points (Agile Approach):**

* Assign story points to tasks based on complexity, risk, and uncertainty.
* Use relative sizing to compare and estimate effort required for different tasks.

**Buffer Allocation:**

* Include contingency buffers for unforeseen challenges or scope changes.
* Allocate contingency as a percentage of overall project effort.

**Techniques and Tools:**

**PERT Analysis (Program Evaluation and Review Technique):**

* Use PERT to estimate task duration based on optimistic, pessimistic, and most likely scenarios.
* Calculate the Expected Time using PERT formula: (Optimistic + 4 \* Most Likely + Pessimistic) / 6.

**Gantt Charts:**

* Create Gantt charts to visualize task timelines, dependencies, and milestones.
* Use tools like Microsoft Project, Asana, or GanttPRO for planning.

**Agile Estimation Techniques:**

* Conduct Planning Poker sessions for team-based estimations in Agile projects.
* Use relative sizing (Fibonacci sequence) for story point estimation.

**Delivery Timeline:**

**Milestone Definition:**

* Define key milestones such as architecture design completion, MVP release, beta testing, etc.
* Assign specific deliverables and dates for each milestone.

**Timeline Validation:**

* Validate the delivery timeline considering resource availability, dependencies, and potential risks.
* Perform a critical path analysis to identify tasks critical for meeting deadlines.

**Iterative Approach:**

* Plan for iterative development cycles to accommodate changes and improvements based on feedback