Online Movie Ticket Booking Platform

Online movie ticket booking platform that caters to both B2B (theatre partners) and B2C (end customers) clients

Brief steps towards design

Being Proactive

Step that involves exhaustive discussion for requirements and thinking wide across all corners of all possible use cases

Requirements

Enable theatre partners to onboard their theatres over this platform and get access to a bigger customer base while going digital.

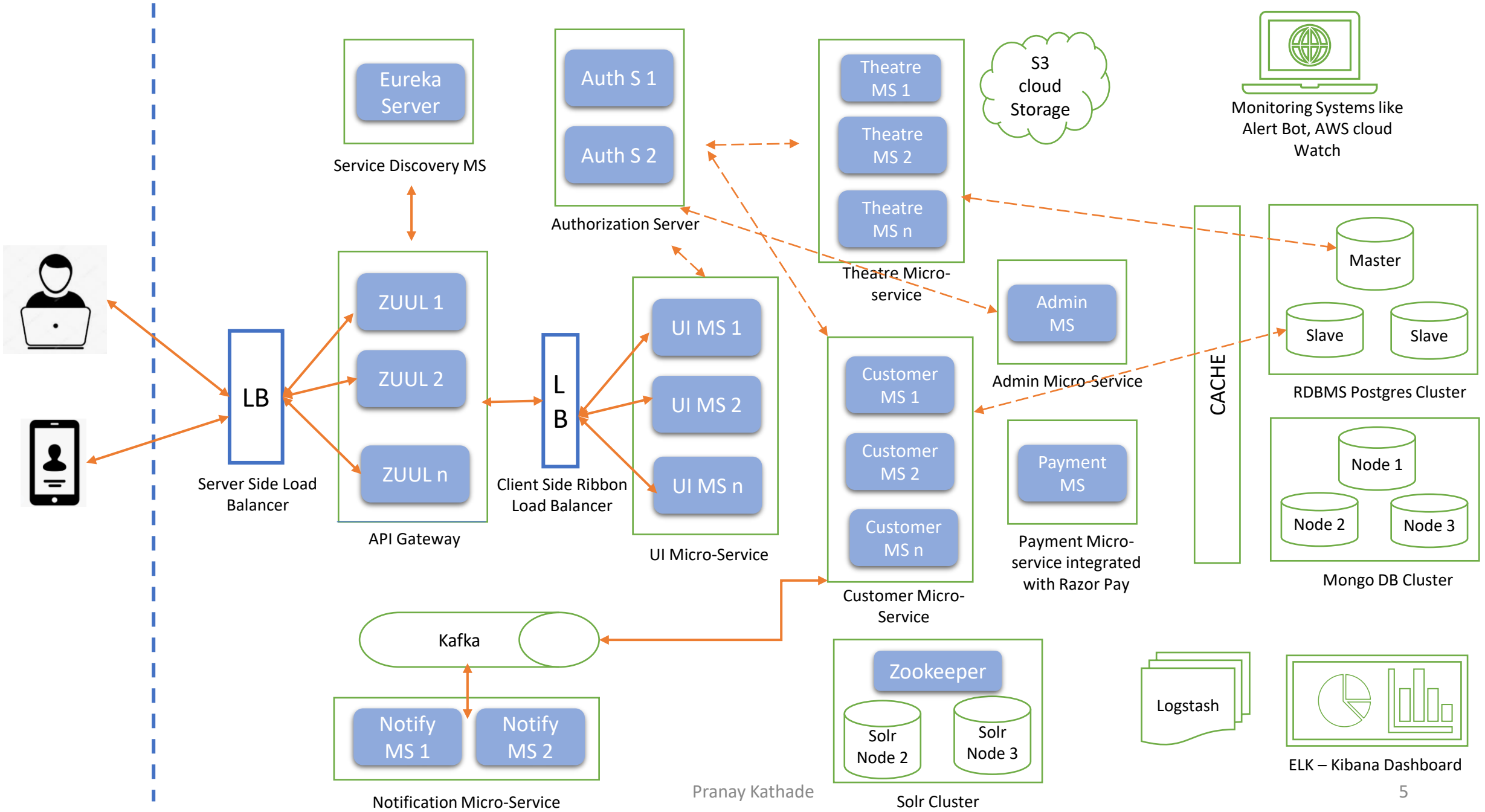
- Theatre partners shall be able to login to the system and should have privileges to create, update, read and delete their data like theatre, screens, shows, upcoming shows, seating categories, offers, etc.
- They shall be able to publish the future engagements as well as track history for their transactions on platform.
- They shall be able to update seat availability and their prices at runtime.
- System should have ability to integrate APIs exposed from existing IT systems of the existing theatres.

Enable end customers to browse the platform to get access to movies across different cities, languages, and genres, as well as book tickets in advance with a seamless experience.

- Any user shall be able to visit platform from any device like mobile, laptop, desktop over internet.
- They should be able to browse all the shows, theatres by location, show timings, etc. search criteria without needing to login to the system.
- They should be able to select the movie show for which they want to book the tickets, should be able to register themselves into system and book the ticket by paying online.
- They should be able to like, comment, share on the show also they should be able to see their previous history on the platform.

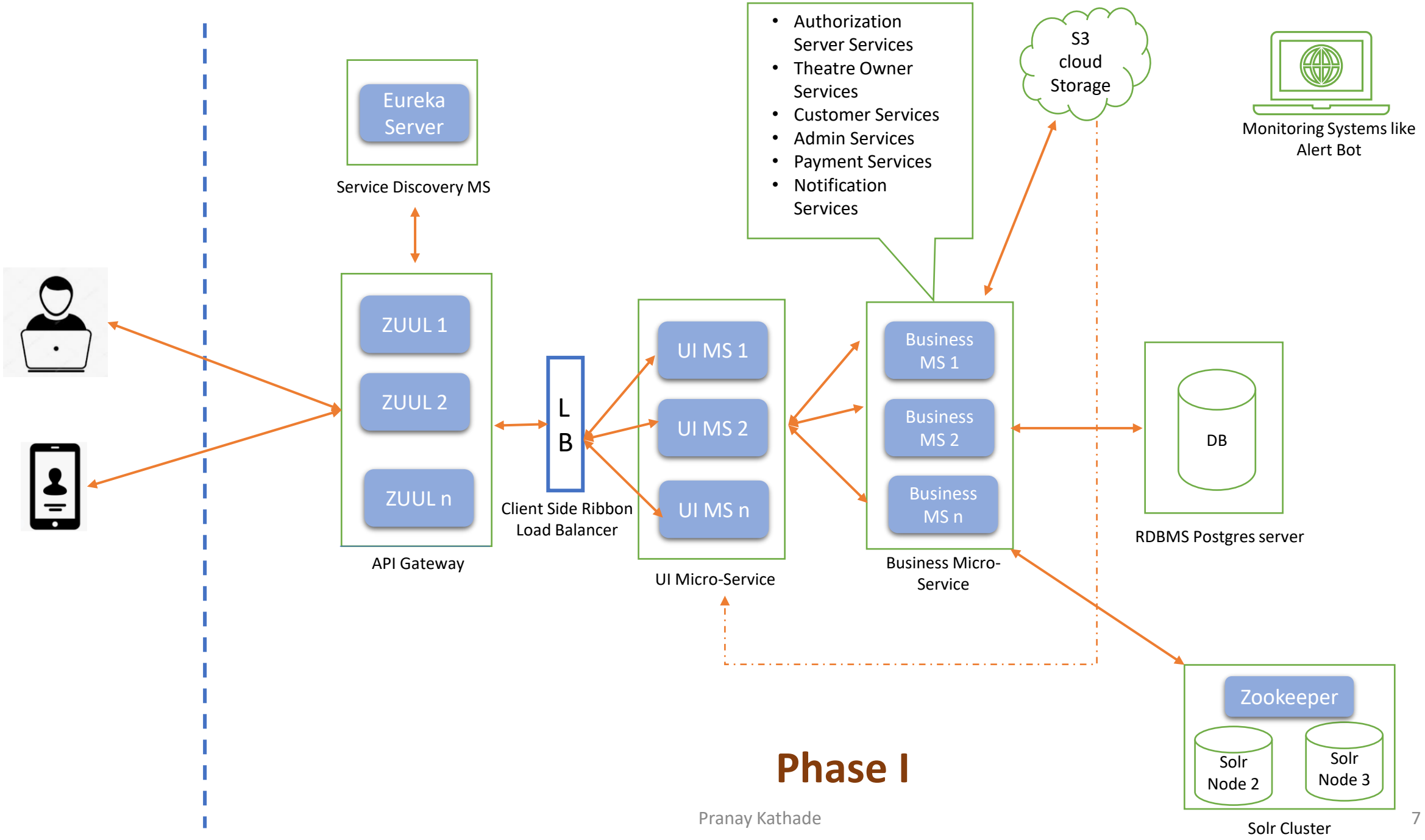
Begin with End In Mind

Step that defines our final milestone


Eureka Server
Service Discovery MS
Auth S 1
Auth S 2
Authorization Server
Theatre MS 1
Theatre MS 2
Theatre MS n
Theatre Micro-service
S3 cloud Storage
Monitoring Systems like Alert Bot, AWS cloud Watch
LB
Server Side Load Balancer
ZUUL 1
ZUUL 2
ZUUL n
API Gateway
L B
Client Side Ribbon Load Balancer
UI MS 1
UI MS 2
UI MS n
UI Micro-Service
Admin MS
Admin Micro-Service
CACHE
Master
Slave
Slave
RDBMS Postgres Cluster
Customer MS 1
Customer MS 2
Customer MS n
Customer Micro-Service
Payment MS
Payment Micro-service integrated with Razor Pay
Node 1
Node 2
Node 3
Mongo DB Cluster
Kafka
Notify MS 1
Notify MS 2
Notification Micro-Service
Zookeeper
Solr Node 2
Solr Node 3
Solr Cluster
Logstash
ELK – Kibana Dashboard

Put First Things First

Step that defines achievable milestone within given timelines


Eureka Server
Service Discovery MS
ZUUL 1
ZUUL 2
ZUUL n
API Gateway
L B
Client Side Ribbon Load Balancer
UI MS 1
UI MS 2
UI MS n
UI Micro-Service
Authorization Server Services
Theatre Owner Services
Customer Services
Admin Services
Payment Services
Notification Services
Business MS 1
Business MS 2
Business MS n
Business Micro-Service
S3 cloud Storage
Monitoring Systems like Alert Bot
DB
RDBMS Postgres server
Zookeeper
Solr Node 2
Solr Node 3
Solr Cluster

Phase I

Technology and Frameworks

Spring Boot

Spring Cloud Netflix OSS

- Eureka Server
- ZUUL API Gateway
- Config Server
- Ribbon Load Balancer
- Hystrix Circuit Breaker
- Feign Client

Spring Framework

- Spring MVC
- Spring Security
- Spring AOP
- Spring thymeleaf template Engine

User Interface

- HTML
- CSS
- JavaScript
- jQuery
- Bootstrap 4

Database

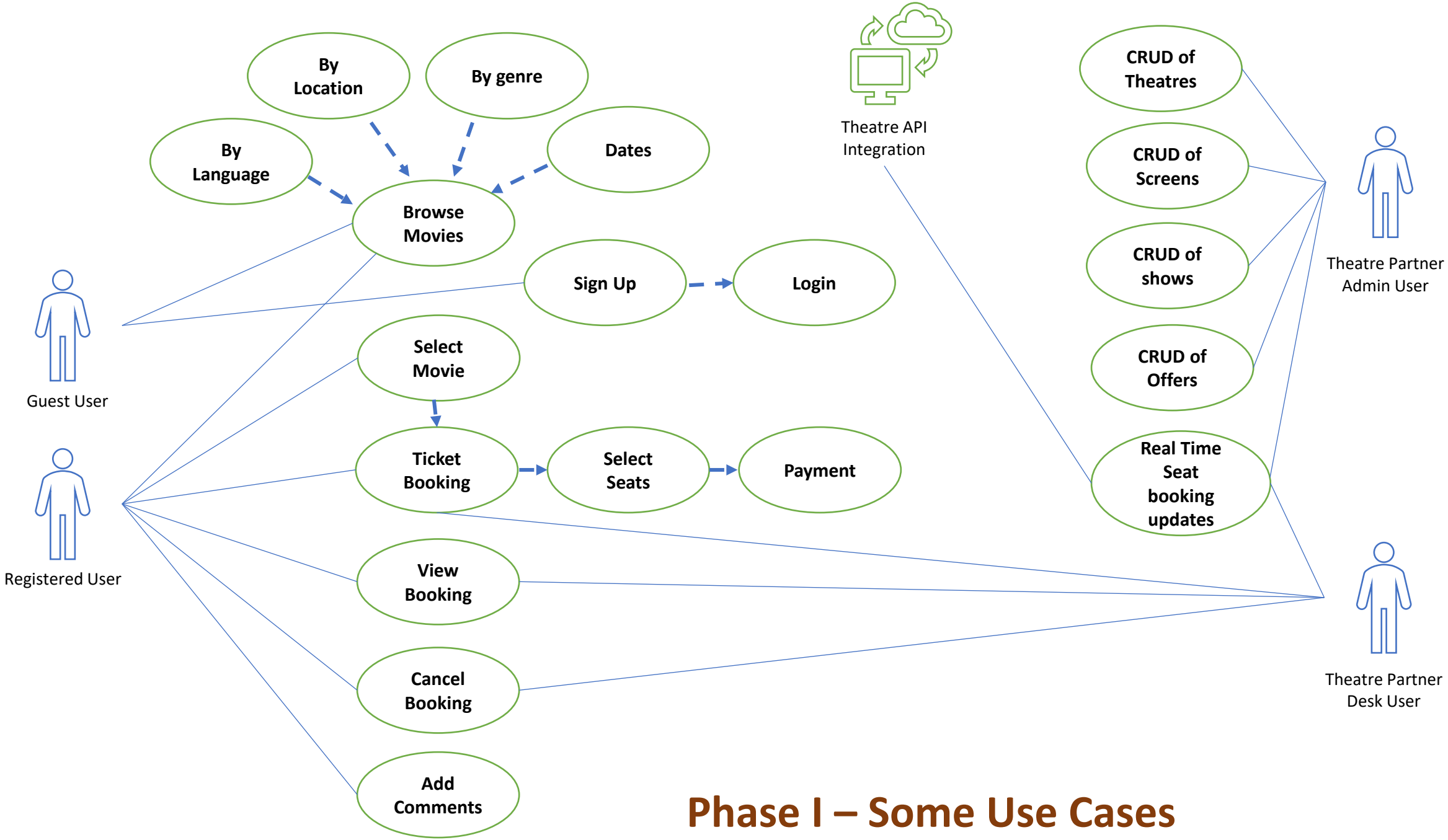
- Postgres SQL

Cloud Resources

- Simple Storage Service (S3)
- EC2 Instances
- RDS Postgres Instance

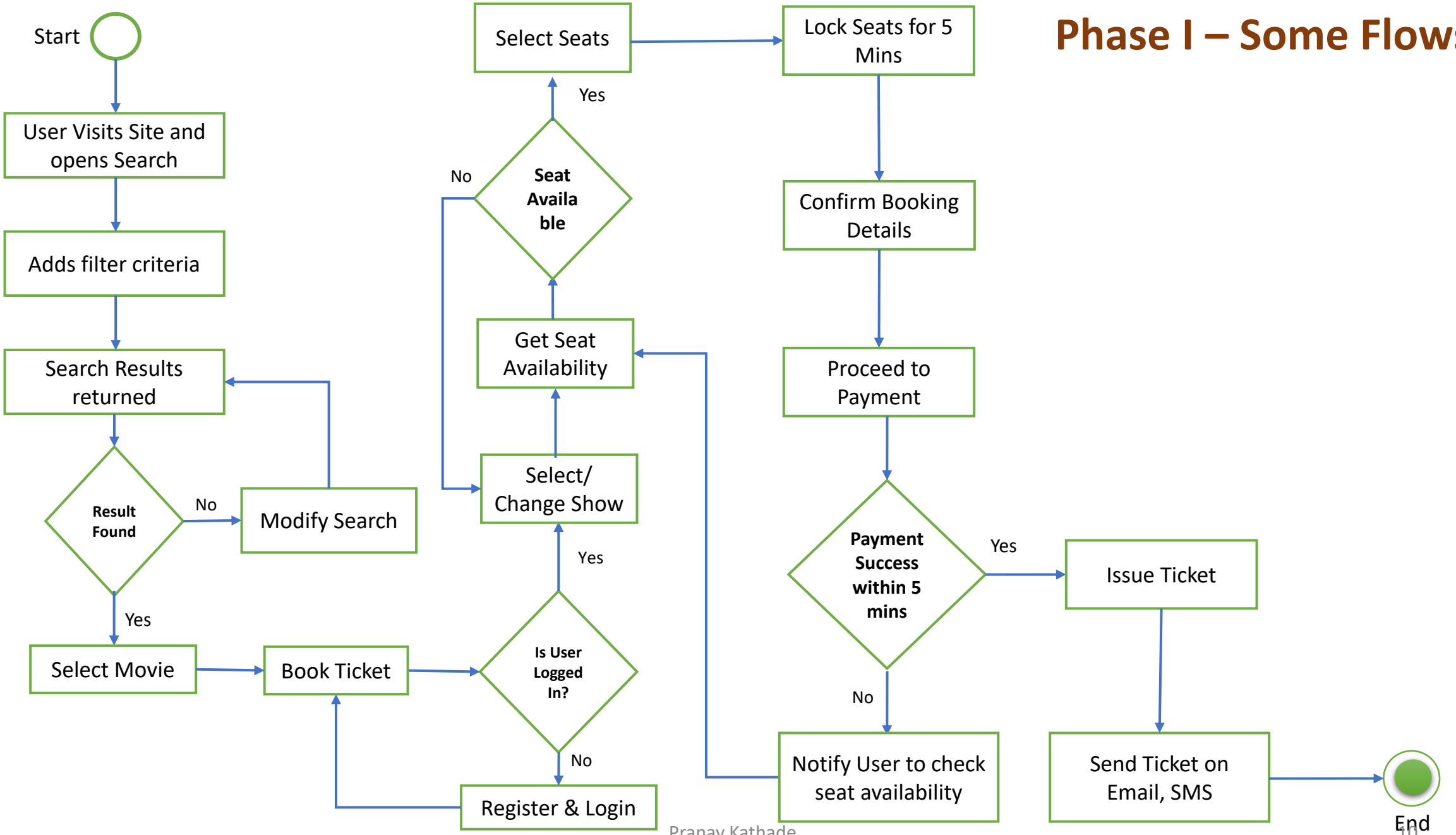
Build Tool

- Gradle
- Batch Files
- SH files



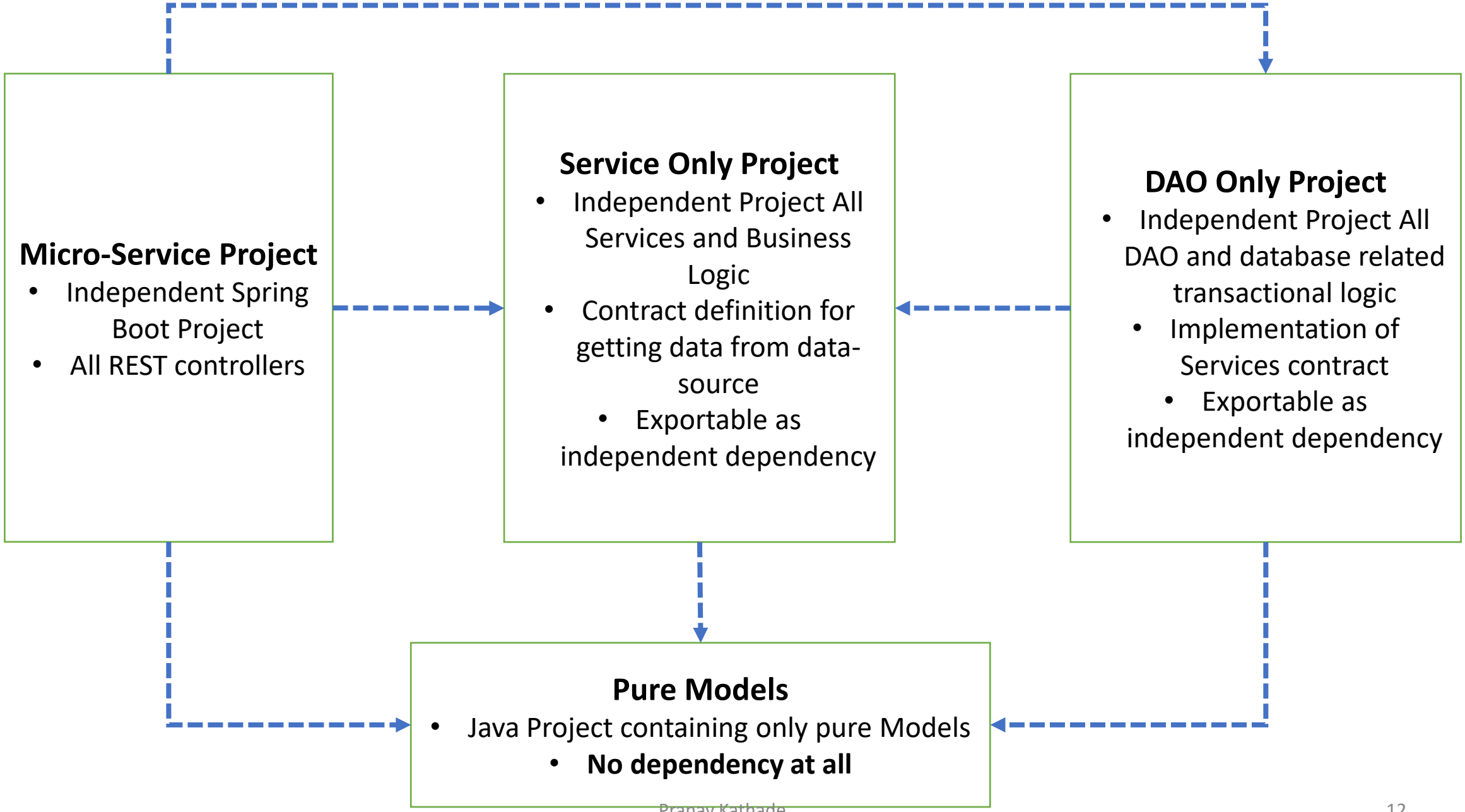
Phase I – Some Use Cases

Phase I – Some Flows



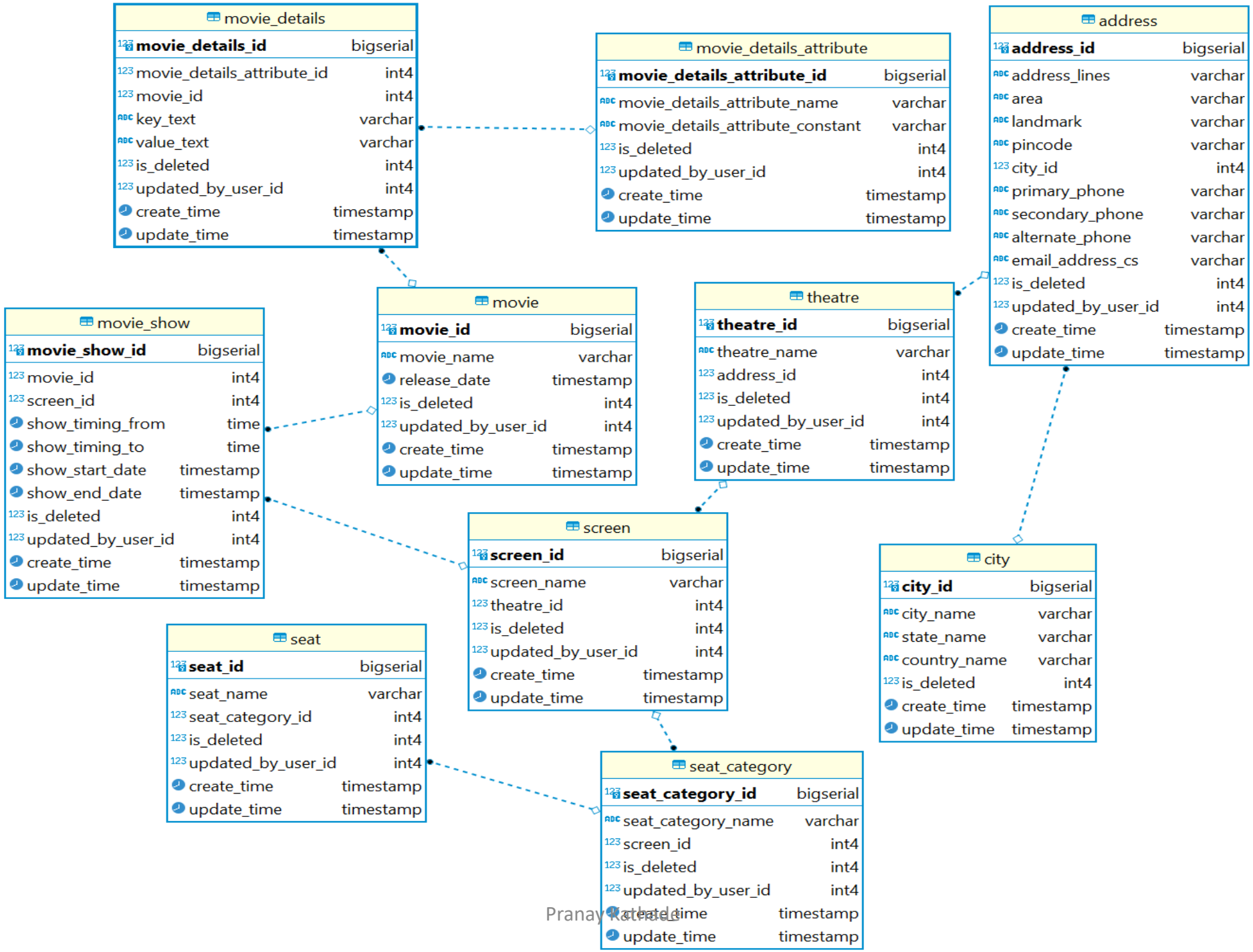
Micro-Modules with Purity of Models

Step that introduces unique concept and set of principles for designing a single unit of Micro-service


Micro-Service Project
• Independent Spring Boot Project
• All REST controllers
Service Only Project
• Independent Project All Services and Business Logic
• Contract definition for getting data from data-source
• Exportable as independent dependency
DAO Only Project
• Independent Project All DAO and database related transactional logic
• Implementation of Services contract
• Exportable as independent dependency
Pure Models
• Java Project containing only pure Models
• No dependency at all

Core DB design

Core business related tables


movie_details
movie_details_id bigserial
movie_details_attribute_id int4
movie_id int4
key_text varchar
value_text varchar
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
movie_details_attribute
movie_details_attribute_id bigserial
movie_details_attribute_name varchar
movie_details_attribute_constant varchar
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
address
address_id bigserial
address_lines varchar
area varchar
landmark varchar
pincode varchar
city_id int4
primary_phone varchar
secondary_phone varchar
alternate_phone varchar
email_address_cs varchar
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
movie_show
movie_show_id bigserial
movie_id int4
screen_id int4
show_timing_from time
show_timing_to time
show_start_date timestamp
show_end_date timestamp
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
movie
movie_id bigserial
movie_name varchar
release_date timestamp
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
theatre
theatre_id bigserial
theatre_name varchar
address_id int4
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
screen
screen_id bigserial
screen_name varchar
theatre_id int4
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
city
city_id bigserial
city_name varchar
state_name varchar
country_name varchar
is_deleted int4
create_time timestamp
update_time timestamp
seat
seat_id bigserial
seat_name varchar
seat_category_id int4
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp
seat_category
seat_category_id bigserial
seat_category_name varchar
screen_id int4
is_deleted int4
updated_by_user_id int4
create_time timestamp
update_time timestamp

Non Functional Requirements

Some Transactional Scenarios

Booking the same seat by more than One User

- This is most common concurrency issue for which we will have to lock the seat selected by first user while booking for five minutes.
- If seat is booked within 5 minutes, then it’s confirmed otherwise release the lock and make it available for other users.
- Fetching most real time seat availability and if acquiring lock is successful then only can proceed with actual booking.

Heavy booking for some show

- We might create a virtual queue and assign a token to the user. While at back end, maintaining a queue for transaction would be very important.
- We might restrict seat selection by user to dynamic allocation.

Cancellation

- One might book bulk tickets and cancel just before the show. So, there should be strong cancellation policy as well as booking policy for number of tickets in one transaction.

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

- Write compatible REST End Points to communicate to external IT system.
- Fetch real time seat availability by polling information constantly.
- Expose End Points to be consumed by external parties by using client credentials. Assign client credentials to the 3rds party who wants to integrate our API.
- Localization in case of movies can be sorted out by adding attribute to movie details table and adding preferences for user. Site availability in multi-language can be done using message resources properties and use of locale

How will you scale to multiple cities, countries and guarantee platform availability of 99.99%?

- Deploying ombp platform on any cloud like AWS would make it easier to ensure high availability.
- Any containerization tool like Docker in combination with orchestration tool Kubernetes can make sure of this.
- Existing cloud capabilities like AWS cloud can be used to scale app horizontally.

Integration with Payment Gateways

- Register with payment Gateway like Razor Pay.
- Obtain credentials as identity.
- Configure call back URLs of our platform into Payment Gateway.
- Configure Payment Gateways URLs into page and redirect to those after verification.

How do you monetize platform?

- By building strong Supply Side with theatre partners to onboard for free.
- Selling theatre partners **S**oftware **A**s A **S**ervice for their inhouse ERP stuff and changing them monthly.
- Addition of some commission per ticket for ticket booked on a platform from theatre partner/ charging service charge (convenience charges) of platform from user booking ticket.
- Hiring a great sales team to answer this question

How to protect against OWASP top 10 threats.

- Take a Zero-Trust Approach to Security
- Use own Authorization server for validating each and every request
- Data at Rest to be ENCRYPTED all the time
- String password Policy and Multi-factor authentication
- Strong access management framework, preferably written inhouse
- Strong Logging and Monitoring mechanism
- Strong input validation on UI as well as in backend code
- Add security filters like XSS filters which helps in converting any sort of data to encoded format to save into database.
- Take help of tools like HCL app scan for dynamic scanning and conduct penetration testing as well.
- Following best code practices.
- Upgrading Open source dependencies as soon as any vulnerability is detected.
- All headers should be parsed and they must meet rules defined in filters.
- Extensive use of **obscurification**. Minify resources like js and css. Change the names of resource files with every new deployment dynamically to enforce cache refresh at client side.
- Strong CORS policy.

Thank you!

Code available at
https://github.com/pranaypkathade/ombp