PUBLICIS SAPIENT

BOOKMYMOVIE

HIGH LEVEL DESIGN DOCUMENT

Version: 1.0

Effective Date: July-24, 2024

Statement of Confidentiality

The information contained herein shall not be disclosed, duplicated, or used in whole or in part for any purpose other than to evaluate the proposal, provided that if a contract is awarded to this offer as a result of, or in connection with, the submission of these information, the recipient shall have the right to duplicate, use or disclose the information to the extent provided in the agreement. This restriction does not limit the right to use information contained in the data if it is obtained from another source without restriction.

Contents

About This Guide

This document is intended for understanding the details (Architectural/Technical Design, basic Features) of all the components of Bookmymovie Application. Purpose of this High Level Design (HLD) Document is to guide represent a suitable model for development. This document is also intended to help detect contradictions prior to coding and can be used as a reference manual for the preparing Low Level Design (LLD).

Audience

This guide is intended for the Bookmymovie Application Technical team.

Related Documents

Document title	Owner/Link	Version
Scope Of Work Document (SOW)	Front Runners	V2.0
Business Need Document (BND)	Product Owner	V1.0

Document History

Created on	Created by	Changed on	Changed by	Reviewed by	Version
24/07/2024	Palash Chanda	24/07/2024	Palash Chanda	TBD	V1.0
		26/09/2024	Palash Chanda	TBD	V2.0

Overview

The High Level Design (HLD) provides the detailed description of all the Components/Modules of this Bookmymovie Application/System. Technically this System can be considered as a collection of several Microservices/Infra on high level.

Clients

Client means who can use and utilize this Application/System. Here Client can be Public Users as well as Organizational Users.

Major Functionalities

Core Features of Bookmymovie Application available in the System:

- Viewer/User can view/update their Profile.
- Load/Create City (name, code) in the System.
- Load/create Theater (name, city, address-line, available facilities) in the System.
- Load/Create Screen (name, theater, show times, total rows, number of seats in each row, row type, row price) in the System.
- Load/Create Seats (seat number, price, type) in the System. Seats distribution (for a Theater) will be done automatically (based the defined Algorithm) when action is triggered.
- Load/Create, Update, Remove Movie (name, category, description, trailer, certificate, duration, casts, crews) for the Viewers in the System.
- Load/Create, View Movie-Show (movie name, city, theater, screen, showtimes) for the Viewers in the System.
- Display Movies (currently running the Show) based on selected City.
- Display Movies (currently running the Show) based on selected City and Category (comedy, action, thriller).
- Display Theaters (currently running the Show) based on selected Movie and City.
- Book Movie by selecting a Theater/Showtime/Seat for a day. Viewer can book maximum 10 Seats at a time. Offers can be applied as below:
  1. 50% discount on the third ticket.
  2. 20% discount on afternoon show booked ticket.

Module Details

Module/Component represents (Technically) each Microservice in this System:

- Orchestrator Microservice: Role: System Entrypoint.
- Viewer Microservice: Role: Load/Display Viewer's Profile.
- Theater Microservice: Role: Load/Display City/Theater/Screen/Seats.
- Movie Microservice: Role: Movie and running Show details.
- Order Microservice: Role: Order processing for booking a Movie.
- Payment Microservice: Roles: Payment processing for booking a Movie.
- Notification Microservice: Role: Sending notifications (email/sms) for booking a Movie.
- Utility Microservice: Roles: Offer details, Ratings (User/Critics), Generate offers automatically based on the DOB, Marriage Anniversary.

Development Methodology

Here we are following Agile Methodology where Software is Developed and Delivered in multiple Phases (known as Milestone).

Development Guidelines

Steps are being followed for Segment Sell Development:

- Developer creates a new branch from master branch.
- QA starts preparing Automation Scripts and push Scripts in acceptance-test folder.
- Developer clones Code from new branch.
- Developer writes Code.
- Developer confirms that local Code Build and Run is successful.
- Developer checks in Code in new branch.
- TL clones Code changes and review.
- TL checks in Code changes after review.
- Architect reviews Code changes.
- Developer creates PR.
- PR Build (all steps in CI) should besuccessful.
- PR is merged with master branch.
- Developer's Code changes are deployed in DEV using automated CI/CD Pipeline.

Dependencies & Risks

Concurrent seat Booking

Multiple Viewers are trying to book the Seat at same time:

- If Requests are coming to same Pod (Tomcat Server) it can be handled by the Java Synchronization.
- If Requests are coming to different Pods (Tomcat Servers) it can be handled by DB side.

Architecture Details

This chapter describes the Application/System Architecture and communication among Components/ Microservices.

Architectural Style

An Architecture can have multiple Architectural Styles. Here we have selected Microservice.

Architectural Pattern

An Architectural Style can have multiple Architectural Patterns. Here we are focused on Backend API.

Technical Flow

Technical Flow of details: TBD.

Design Principles and Patterns

Design Principles/Patterns should be used:

- 12 Factor App Principles.
- Microservice Design Patterns:
  1. Decomposition Pattern.
  2. API Gateway Pattern.
  3. Integration Pattern: Agreegrator Pattern, Choreography Pattern.
  4. Centralized/Externalized configuration.
  5. Service Registration & Discovery.
  6. Database Pattern.
  7. Deployment Pattern.
  8. Error Approach.
  9. Transaction Approach.
  10. Testing Approach.
  11. Security Approach.
  12. Scaling Approach.
  13. Reliability Approach.
  14. Monitoring Approach.
- OOPs Features and OOPs Design Patterns (like SOLID, DRY Principles etc.).
- Java Design Patterns: TBD.
- Dependency Injection Pattern so that Object Relationships become Loosely Coupled.

Decomposition Pattern

Microservices can be identified and decomposed as:

- Decomposed by Functionality: Theater, Movie, Vierer, Order, Payment etc.
- Decomposed by Roles/responsibilities: Auth, Gateway/Orchestrator, Utility etc.

API Gateway Pattern

API Gateway Patten can be implemented as:

- Istio Service Mesh will be acting as an API Gateway for the Entire System.
- Gateway Microservice will be acting as an API Gateway for the Microservice Ecosystem.

Choreography Pattern

Choreography Pattern implements Microservice-to- Microservice communication. Here Microservices can talk to each other directly (Asynchronous mode) without any centralized Orchestrator.

Centralized/Externalized Configurations

Configurations can be Centralized and Externalized as:

- Business Configurations (Rules/Validations) should reside in DB.
- Technical Configurations in GKE ConfigMap

Service Registration & Discovery

Service Registration/Discovery is required for the Microservices communication. Prerequisite is Istio Daemon should be installed on the Kubernetes Cluster. Approach:

- Microservice should be registered in Istio Data Plane/Layer (Service Entry).
- Microservice should be discovered by Istio Control Plane/Layer (Sidecar Envoy Proxy).

Database Pattern

Database Pattern should be Single DB per Microservice. Details:

- GCP Firestore (Datastore mode) Service will be used. Reasons behind selecting NoSQL DB:
  1. Required Document-Data-Model not Relational-Data-Model.
  2. Scalibility of NoSQL DB is better than Relational DB in high traffic environment.
  3. GCP Datastore is a Multi Regional Service. Disaster Recovery & backup not required.
  4. ACID property support not required.
- GCP Datastore Database Kind/Collection Relationships:
  1. One City can have multiple Theaters.
  2. One Theater can have multiple Screens.
  3. One Screen can have multiple Seats.
  4. One Screen can have multiple Movie-Show.

Deploment Pattern

Deployment Pattern can be:

- Single Container per Pod for each Microservice.
- Blue-Green Rollling Upgrade Strategy where exitsting Workload will not be touched.

Error Approach

Error management approaches are:

- Transient Errors: Retry immediately. Details:
  1. Exception/Errorcode/Errormessage mappings should be configured.
  2. Exceptions should be cought only in the Service Layers and thrown from the other layers.
  3. Populate Error block (error code/message) in Response based on the Exception occurs.
- Non-Transient Errors: Replicate, Analyze, Find root cause and provide the fix.

Testing Approach

Testing Approaches are:

- Unit Testing using JUnit/Mockito/SquareTest.
- E2E/Integration Testing using Web/Mobile Applications.
- Automation Testing: BDD approach (using Cucumber, Gherkin language).
- Performance Testing using JMeter: Load, Stress Testing.
- Smoke Testing: Validate that new Build is ready for the Testing.

- Sanity Testing: Validate that Bugs are fixed or not.
- Regression Testing: Validate existing functionalities working or not after the Bug fixed.
- Notes: For the Defects/Bugs (Severity: Fatal/Major/Minor) management JIRA will be used.

Security Approach

As per the existing implementations, following Security mechanisms are available in the System:

- Authentication/Authorization: Determines Who/What can be accessed. Approaches:
  1. For the API used by Public Users in B2C, Authentication is enabled using Mobile/OTP.
  2. For the API used by Public Users and Organization Users in B2B/B2C, Authentication/ Authorization are enabled.
- Cloud shared Security model:
  1. Cloud will secure: Runtime, OS, Server, Networking.
  2. Customer will secure: Data, Application, IAM, Configurations.
- Future Scope/Enhancement: OAuth2.0 External Authorization (Google/Apple/Facebook etc) implementation can be planned as a part of.
- Future Scope/Enhancement: Encryption can be enabled when System Users (Public Users and Organization Users) are consuming API from the external Client (like app, web). Here encrypted Data can be decrypted at Server end.

Reliability Pattern

Reliability Approaches ensures Fault Tolerant/High Availability:

- Create multiple Regions.
- Create multiple AZ in a Region.
- Create multiple Clusters in an AZ.
- Create multiple Servers in a Cluster.
- Create multiple Application instances in a Server.
- Scaling: Kubernetes HPA (CPU based Autoscaling).
- Kubernetes DR/Issues:
  1. ETCD Backup.
  2. Multiple Cluster set up: Multiple Kubernetes Clusters should be there in different Regions and can be accessed using LB/GSLB (in Acrive-Acrive or Acrive-Passive mode).
  3. Enable Graceful shutdown.
- Implement using Resiliense4J/Hystrix: Circuit Breaker, Rate Limiter, Retry, Bulkhead.
- Cache:
  1. Server/Client side Caching: NA.
  2. CDN: Frontend (static components) caching is not enabled as this Application is not global.

Observability Pattern

Observability/Monitoring Approaches can be:

- Logging/Log-Agreegation: GCP Log.
- Kubernetes Metrics creation using Micrometer API.
- Kubernetes Alerts creation and configure Email/SNOW.
- Distributed Request tracing: GCP Trace.
- Kubernetes Health check: Liveness Probe, Readiness Probe.
- JVM Performance Tuning: Heap/Stack memory set, GC set, GC Logging set.
- JVM Performance Profiling/Monitoring:
  1. Download/Analysis Heap Dump from Kubernetes Container into a local machine.
  2. Download/Analysis Thread Dump from Kubernetes Container into a local machine.
  3. Analysis GC: GC Log, GC is running frequently.

Technology and Tool Details

Cloud Provider

GCP Cloud Provider is selected and Resources Selection is done based on the Requirements/Budget:

- GKE: Kubernetes Cluster for Microservices Containerized Deployment.
- GCR: Docker Images storage.
- Firestore DB (Datastore mode): NoSQL Database and supports write operations with no limit.
- GCP Operations/Stackdriver: Logging, Monitoring (Metrics/Alerts), Trace.
- GCP PubSub: TBD.

Code Repository

Code Repository will be used:

- Distributed VCS: Bitbucket Repository Monorepo.
- Single Code Repository for all the Environments (DEV/INT/CERT/PROD).

CI/CD

CI/CD purpose Jenkins Pipeline Job (Declarative Pipeline) will be used to integrate following phases/stages:

- Set up Environment.
- Source Code Checkout.
- Unit Testing.
- Build Artifact.
- Automation Testing.
- SonarQube Rules Validations.
- Veracode Code Scanning.
- Publish Artifact to Nexus repository Manager.
- Generate Image.
- Publish Image to GCR.
- Trigger Deployment for the Microservice Deployment in GKE.

Tools/Frameworks Used

Languages/Frameworks/Tools selected:

- Languages/Frameworks: Java17.0 (OpenJDK), Spring (Core/Web/Data/Boot), REST.
- Container Technologies: Docker, Kubernetes, Helm, Minikube.
- Code-analysis/Code-security: SonarQube/Veracode.
- Build/Deployment: Maven/Jenkins.
- Web Servers/IDE/Tools: Apache Tomcat7/IntelliJ/GCloud-CLI/Postman/JMeter.
- Databases: GCP Firestore (Datastore mode).
- Infrastructure as Code Tool: TerraForm.
- Diagram as Code Tool: PlantUML (C4-Diagrams).
- Generative AI: Spring-AI/OpenAI (AI Service provider).

Coding Standards

Industry standard best practices will be used during the Development:

- OOP/Java Design Patterns.
- Code Design approach.
- Code general Conventions.
- Notes: For more details refer to the Coding_Guideline_V1.0 specification.

Functionalities Overview

Booking Process

Functional Flow Sequence diagram: TBD.

Flow details: Prerequisites: Viewer will be loaded in the System after providing the required details.

- Viewer/User will login in the System.
- Viewer/User will search and select the City.
- Movies will be displayed. Viewers can search Movie by name/category.
- Select Movie and get details (category, description, trailer, certificate, duration, casts, crews).
- Proceed for the Booking.
- Get Movie-Show details (theater available for the city, screens available for the theater, seats available for the screen, showtimes).
- Select Theater/Screen/Seats/Showtime. Viewer can select maximum 10 Seats at a time. Offers can be applied as below:
  1. 50% discount on the third ticket.
  2. 20% discount on afternoon show booked ticket.
- Proceed for the Payment.
- Provides the Payment details.
- Finally receive the Booking details (movie name, theater, screen, seats, showtimes, total amount paid, booking id) on successful Payment.
- Notes: For more details refer to the BND_V1.0 (Business Need Document) specification.

Compliance & Security

As a Compliance Certificate, Veracode Code Scanning Report will be provided at the end of Milestone.
No Compliance will not be provided for the Infrastructure, Cloud Services.

Future Scopes

Following Features and Enhancements can be considered for the Future Scopes:

- Enable Bulk Operations (add/update) for the City/Theater/Screen.
- Enable Movie-Show update (seat/show-time) after successful Seat Booking.
- OAuth2.0 External Authorization (Google/Apple/Facebook etc) implementation.
- Neo4J Graph Database integration for Rating/Review/Notifications etc.
- GCP Metrics creation for Monitoring using Micrometer Framework.
- GCP Alerts creation for Monitoring.
- Implement Fault Tolerant/High Availability using Resiliense4J/Hystrix: Circuit Breaker, Retry.
- AI based Chatbot integration to enable Automation: check City/Theater/Movies/shows etc.

Bibliography

https://in.bookmyshow.com/

https://docs.spring.io/spring-framework/reference/index.html

https://www.youtube.com/

https://www.google.co.in/