**Problem statement**

XYZ is a start-up, an online movie ticket booking platform, providing with B2B and B2C solution for its theatre partners and end customers to onboard theatre and book online tickets.

B2B Theatre partners can onboard their theatres over this platform to go online and get access to bigger customer base and become online.

B2C end customers can browse platform to get access to movies across different cities, languages and genre to be able book ticket in advance and have a seamless experience.

**Technologies recommended**

* Language -Java and other add on languages
* Frameworks- Any
* Database - Any
* Integration technologies- Any
* Cloud technologies- Any
* Use your own editor to build and present solution

**Evaluation criteria**

* Code Implementation and completeness (APIs and Design Patterns)
* Design Principles to address functional requirement and Non-functional requirement
* Platform Solutions detailing
* Solution completeness, presentation and discussion.
* Solution coverage uniqueness and extensibility.

Note: Incomplete solution component would be discussed during discussion round.

*You can skip solution areas that you are not comfortable by making a note of it.*

**Functional features to implement** (Mandatory - Code Implementation):

**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
* Booking platform offers in selected cities and theatres
  + 50% discount on the third ticket
  + Tickets booked for the afternoon show get a 20% discount

**Anyone of the following write scenarios:** (Only Service implementation needed-No UI required)

* Book movie tickets by selecting a theatre, timing, and preferred seats for the day
* Theatres can create, update and delete shows for the day.
* Bulk Booking & Cancel booking
* Theatres can allocate seat inventory and update them for the show

**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.

**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 KPIs overall.
* Create a high-level project plan and estimates breakup.

Disclaimer:

*This document is meant to assess your technical skills and is classified as "Sapient confidential". This document by any means shall not be used/shared without permission from Sapient, non-adherence to this can get your candidature blocked for employment with Sapient.*

**TOC**

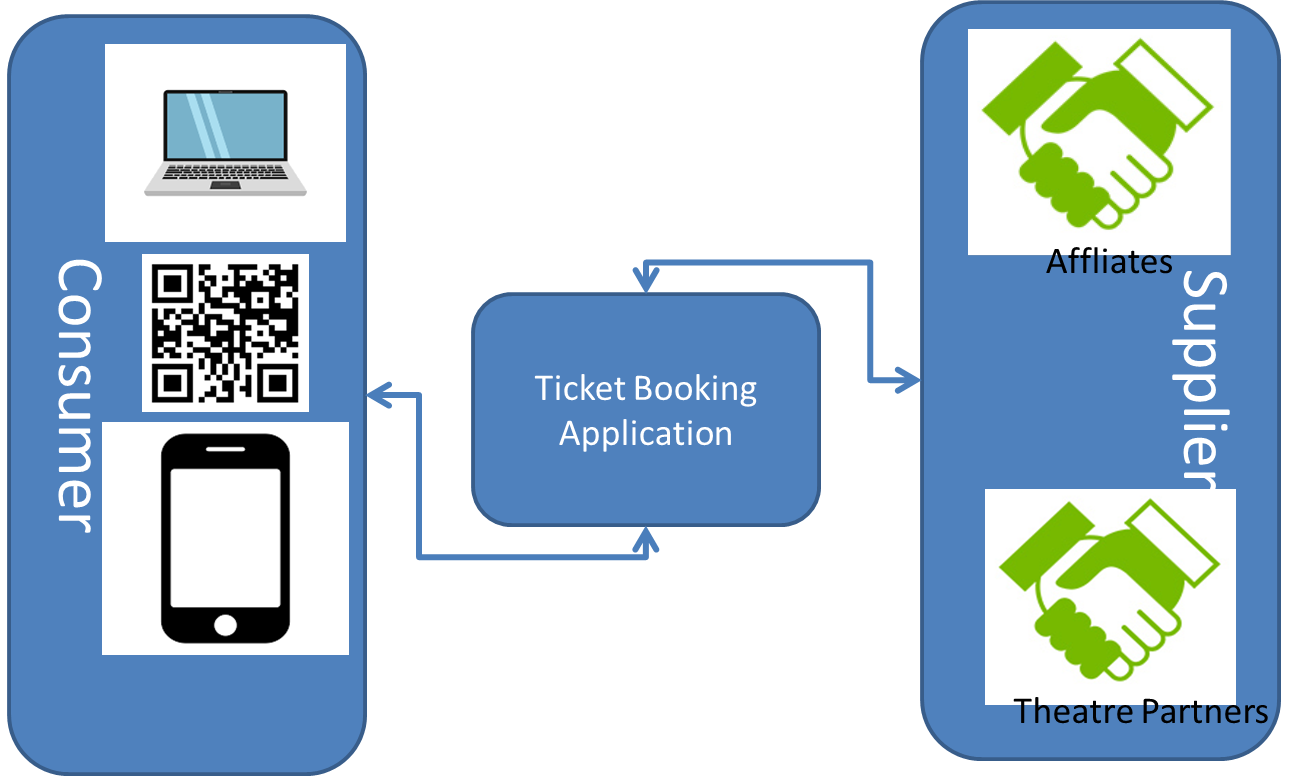
* High Level Design of BookMyTicket Application
* Technology Stack
* Deployment Model
* Application Monitoring
* Non Functional Requirements
  + Performance
  + Web Application Security
  + Load Balancer for Horizontal Scaling
  + MicroServices architechture for High Availability
* Functional Requirements
  + Partners Onboarding Service
  + Customer Onboarding Service
  + Ticket Booking Service
  + QRScanner Service
  + Notification Service
  + Asynchronous Partner Onboarding (covers integrating partner infrastructure)
* Low Level Design for Functional Requirements
* CI/CD Pipeline (Git/BitBucket/Unit Test/Maven/SymLink)
* OWASP top 10 threats
* Monetize
  + Ads
  + Affliate Program
* High Level Project Plan
* KPI

Please note that there are many other layers to think about given the time trying best to cover the most usual cases thereby may be missing finer details, which can be up for discussions as well.

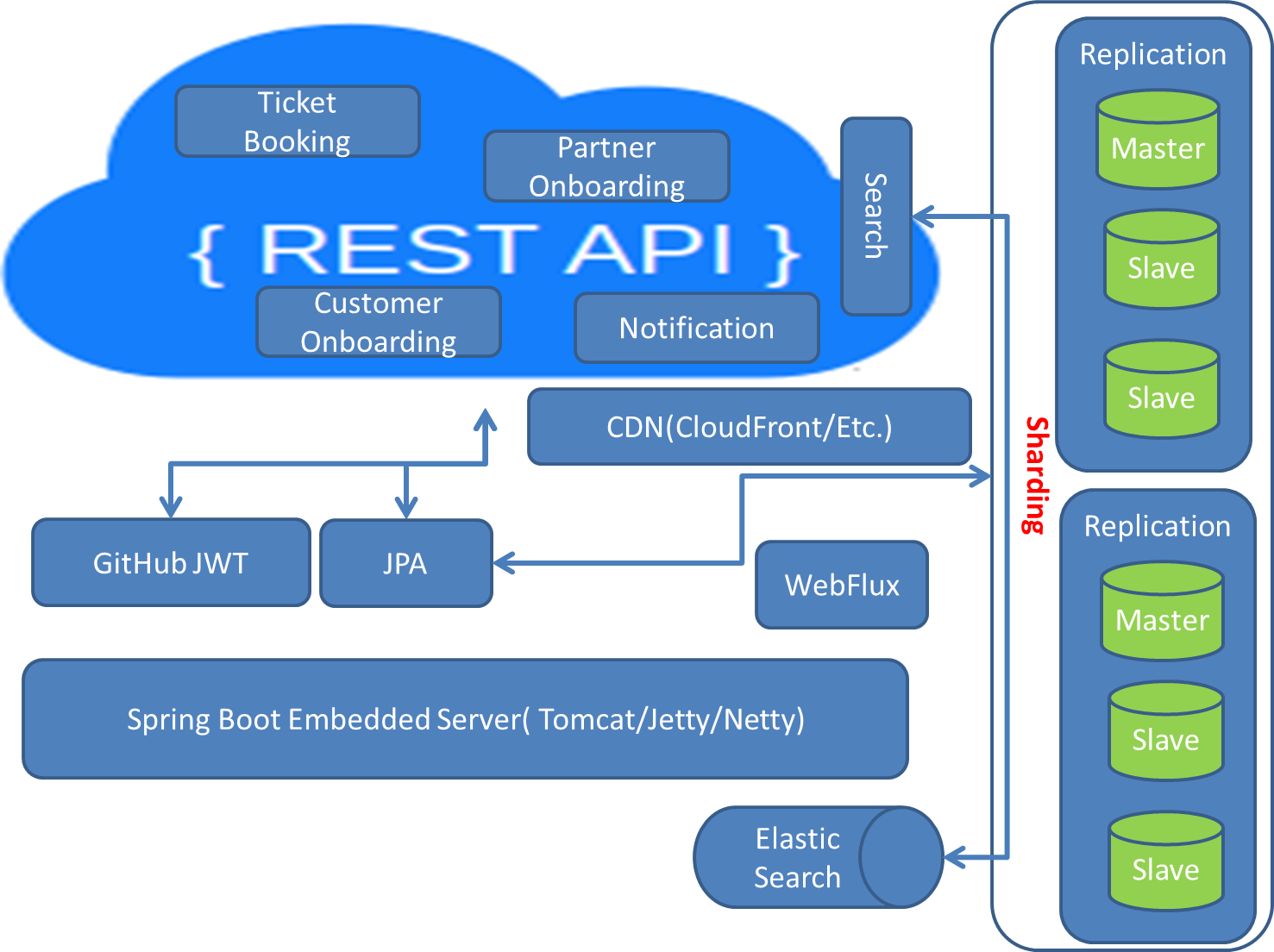
**HIGH LEVEL DESIGN OF THE BOOKING SYSTEM**

There are 3 types of consumers in this application. They are:

* Desktop/Laptop User – Generally searching the movies or else buying a ticket.
* Mobile User – Same as Desktop/laptop user.
* Data Providers
  + Theatre Partners
  + Affiliates – Review writers, blog writers etc.

****

**TECHNOLOGY STACK**

****

**GitHub JWT**

To enable security both at the user access (Partners as well as Consumers) one can use any OAuth2 provider service like Gihub or Google OAuth2 for web security. The other option as well is to have own implementation of resource server using Spring Boot Authorization Server.

**WebFlux**

This one we can use for async notification to customers about offers upcoming bookings etc. With the use of reactive web we can achieve that. Note reactive web is based on reactor framework.

**CDN**

This will be used for images for high performance.

**MySQL Cluster**

While all the booking data, partner data, payment data and customer data needs transactional behaviour and hence RDBMS is a good choice for storing them. It a good idea to create replica mysql cluster for high availability of data just in case one of the master or slave goes down. Imaging the consumer is about to get to the screen the watch the movie and the database is down the verify the QRCode provided by the Booking Service, this will be huge customer dissatisfaction which can be avoided using replication. If the data set grows very large one can go over data sharding where data is distributed across the sharded clusters.

**Elastic Search**

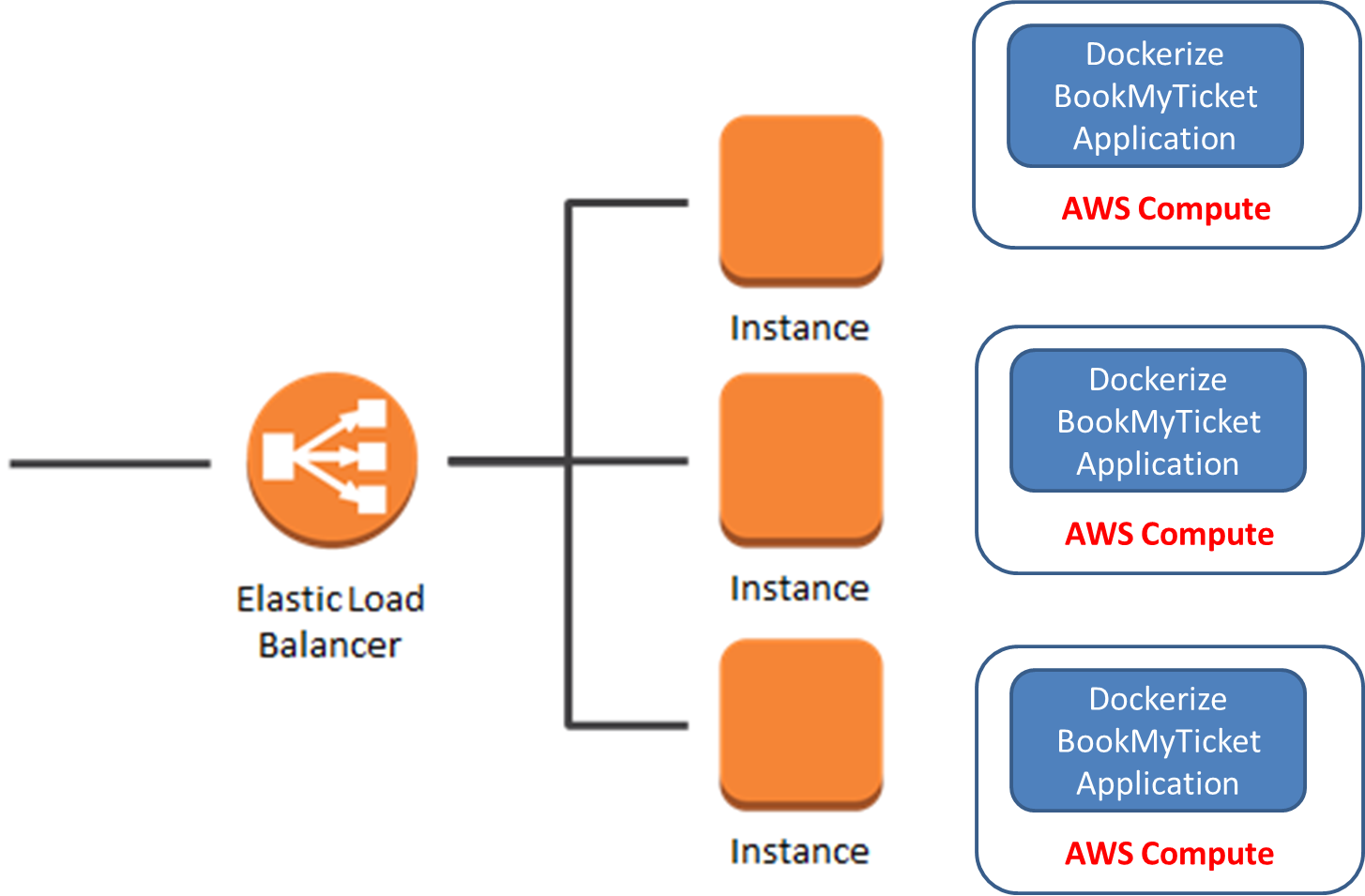
This will be used to store all movies data, reviews etc. User search will be fetched from Elastic search.

**Payment Gateway**

This can be achieved via registering your application with Payment Gateway Providers and when the user is prompted for payment, the user is taken to the gateway provider website for payment and on success redirected back to the main application for confirmation/rejection etc.

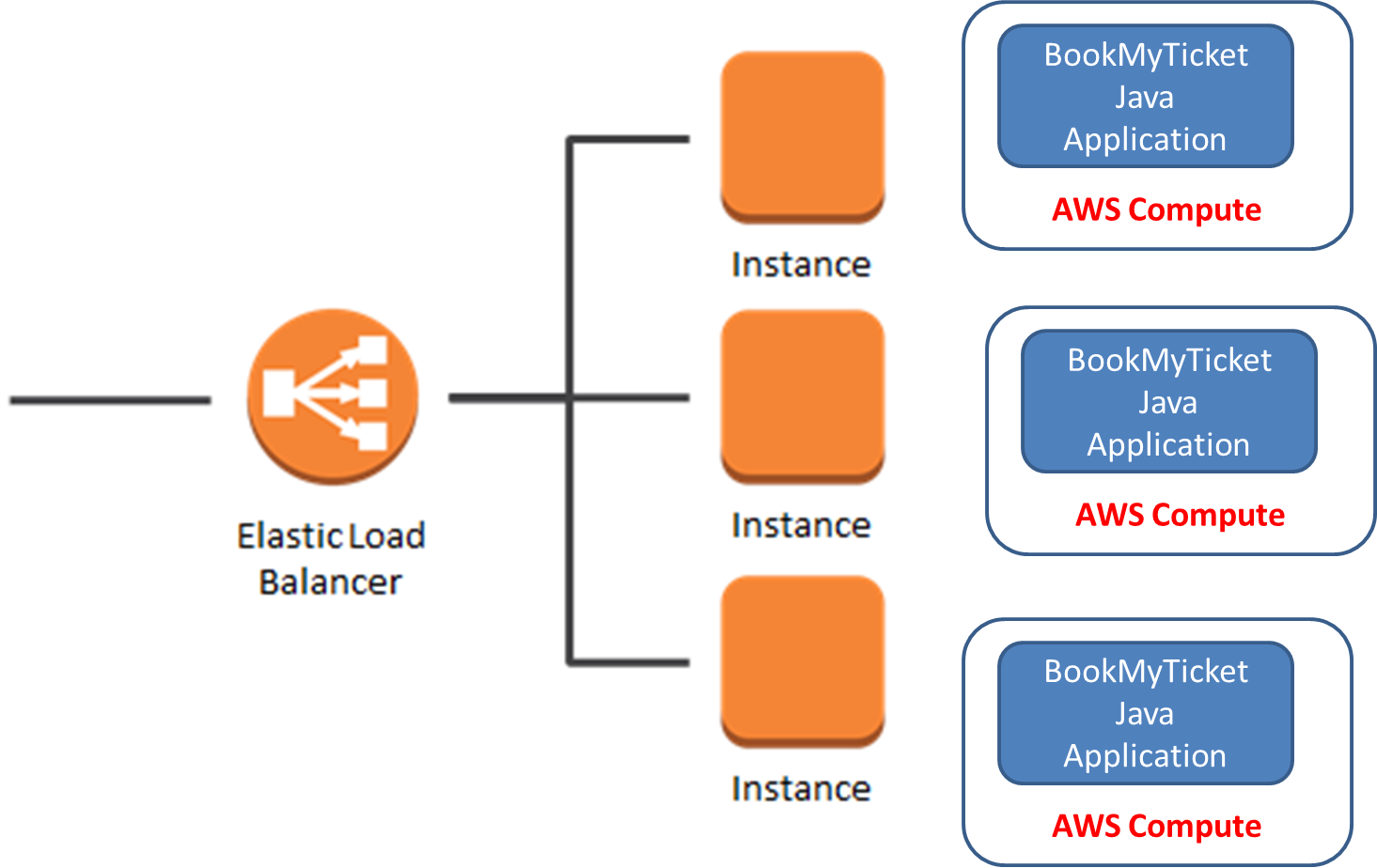
**Application Deployment Model**

**Option 1 – Dockerize and Deploy**

****

**If you dockerize the application, we can get all docker advantage.**

**Option 2 – Java Application Deployment (Why Not Dockerize – Knowledge/Resource/DevOps)**

****

**Other Deployment Options**

* **In house deployment (Cons – Infrastructure Management, Pros – Cost Advantage)**
* **Hybrid Model**

**Application Monitoring**

* **AWS Deployment**

AWS provide utilities to monitor your cloud infrastructure which is cloudwatch. It comes with all application monitoring tools required.

* **Custom Deployment**

If on custom deployment following would help with devops team.

1. Health checks in place for all services running and notification on failure. Ex: Cronjobs etc.
2. SOP’s for devops for helping through potential failures and resolutions.
3. Spring boot actuators.

**Non Functional Requirements**

* Performance
  + Load Balancing
  + Cache (CDN)
  + Small html footprint.
  + Optimize Images (WebP)
  + GZip Compression.
* Security
  + Customer & Partner Login by ROLE.

Authentication ensures that the person who he claims to be is infact the same person, this as example we achieved through GitHub Authenticator. Offcourse other options include Spring Boot Resource manager, Google OAuth2, Etc.

Authrorization ensures that the user access only those resources which the user is entitled to. This is achieved through defining role for each user. Following possible roles are there in our application,

* CUSTOMER
* PARTNER
* OPERATOR
* Etc.

This requires that our application manages user to role assignment along with the authentication from other resources.

* + Rest API Security

The http request must contain the Auhorization header with JWT token, this will enable rest resources to only those request which has the token other requests will be rejected with 401 error.

* Horizontal Scaling

1. Amazon AWS Provides Elastic Loadbalancer which we can use for this purpose.
2. If not in AWS, we can use NGINX as loader balancer across no of the server instance we have deployed.

* Microservices

We are going the design and implement our services as microservices. You can either imagine each of the front facing services like booking service, customer service, partner services as separate microservices or if they are not huge it can be bundled as single microservice.

Design Consideration

* + - Stateless implementation
    - Single code multiple deployment.
    - Replacing one node with another in a very less time.

**Functional Requirements**

**TODO**

**Low Level Design for Functional Requirements**

**TODO**

**CI/CD Pipeline (Git/BitBucket/Unit Test/Maven/SymLink)**

TO BE DISCUSSED

**OWASP top 10 threats**

* Broken Access Control

As discussed in the section GitHub JWT we are going to secure our web application through github authentication as well as protect all rest resources throw token based access otherwise all access to rest resource are restricted. Roles are either managed by a third party solution or self manged and is controlled at the server side.

* Injection
  + Input Validation
  + Prepared Statement and Stored Procedures
* Cryptographic Failure
  + Use Https
* More to do….

**Monetize**

Two approaches that can help monetize the website along with the actual functional reason the project was build for. They are:

* **Advertisements**

This is fairly common and every site we visit today based on user preference and browsing habits show item of their interest for sale and every sale done through this method attracts a commission. Ticket booking application can tie up with other booking applications like travel, hospitality etc and earn some commission on sale.

* **Blogging & Affliate Marketing**

We can invite affiliates such that any sale of movies ticket happens through there social circle would increase the sale of movie tickets from our site and inturn we provide a commission to the affliates. This is a win-win for both parties.

**High Level Project Plan**

Across all of my previous few projects we have been following scrum based project management methods.

* Breakdown the design into smaller tasks.
* Create JIRA for each task.
* Spring Planning – take tasks based on the priority for the next sprint. Consider backlogs.
* Assign JIRA to developers.
* KickOff the sprint.
* Monthly Sprints
* Measure Backlog at the end of the sprint and repeat the cycle.

Example Plan for a Sprint as below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SL NO** | **TASK** | **JIRA** | **Owner** | **Estimate** | **Start Date** | **Expected End Date** | **Current Status** | **Sprint Name** |
| 1 | GitHub JWT Integration | 1234 | Developer 1 | 2 Days | 20-Oct | 22-Oct | IN PROGRESS | October-Spring |
| 2 | Data Mode for Customer | 1235 | Developer 2 | 5 Days | 21-Oct | 26-Oct | IN PROGRESS | October-Spring |
| 3 | Ticket Booking REST API | 1236 | Developer 1 | 3 Days | 23-Oct | 26-Oct | IN PROGRESS | October-Spring |
| 4 | Ticket Booking Repository Implementation | 1237 | Developer 2 | 3 Days | 26-Oct | 29-Oct | IN PROGRESS | October-Spring |

**KPI**

* Non Functional
  + Monitor Google Site Analytics platform to check performance, visitors, impressions , click through rate(CTR= click / impressions)etc.
* Functional
  + % failure on system against % success
  + System Downtime
  + Customer Retention
  + Performance Metrics
  + Load Test Metrics
* Business
  + Customer Issue Resolution Metrics.
  + Partner Onboarding Metrics
  + Customer Onboarding Metrics