Created by: Raditya Saskara

TixHub System Design Document

Project Overview

Abstract

There is already an app for booking tickets and creating events out in the open world available to use but it has a difficult user experience. Tixhub is a booking ticket similar to the ones that are already existed, but it boasts fast, reliable user experience. With microservices approach, this app will be highly available and should have no down time in the production.

Objective

My goal for this project is to deliver a fully working microservices web application. The app will be highly available with Kubernetes clusters managing its lifetime and then connected to a messaging stream called NATS-Streaming Server so that even if a replica of a service is down all of the information being transmitted does not disappear. This app is made to solve a pain point that event bookers have, which are slow and down all the time booking application.

System Level Design

System Requirements

- **App**

The app will include a web app that will be similar in terms of functionality and features of other apps that existed. One of my main goals is to make it as seamless as possible and fast at all times. The app will need to communicate to a third-party payment service called Stripe.

Functional Requirements

- **Synopsis**

Allow users to purchase any tickets that are available on the website and pay using Stripe API.

- **List of requirements**

Must show a list of a tickets that are available to buy. The user must also have the ability to buy, get details, and create their own ticket show.

- **Data Collection**

The following data points will be collected when the user has signed up for the app.

1. Full name
2. Address
3. Telephone Number

- **User Input**

The app will facilitate user input of the following data

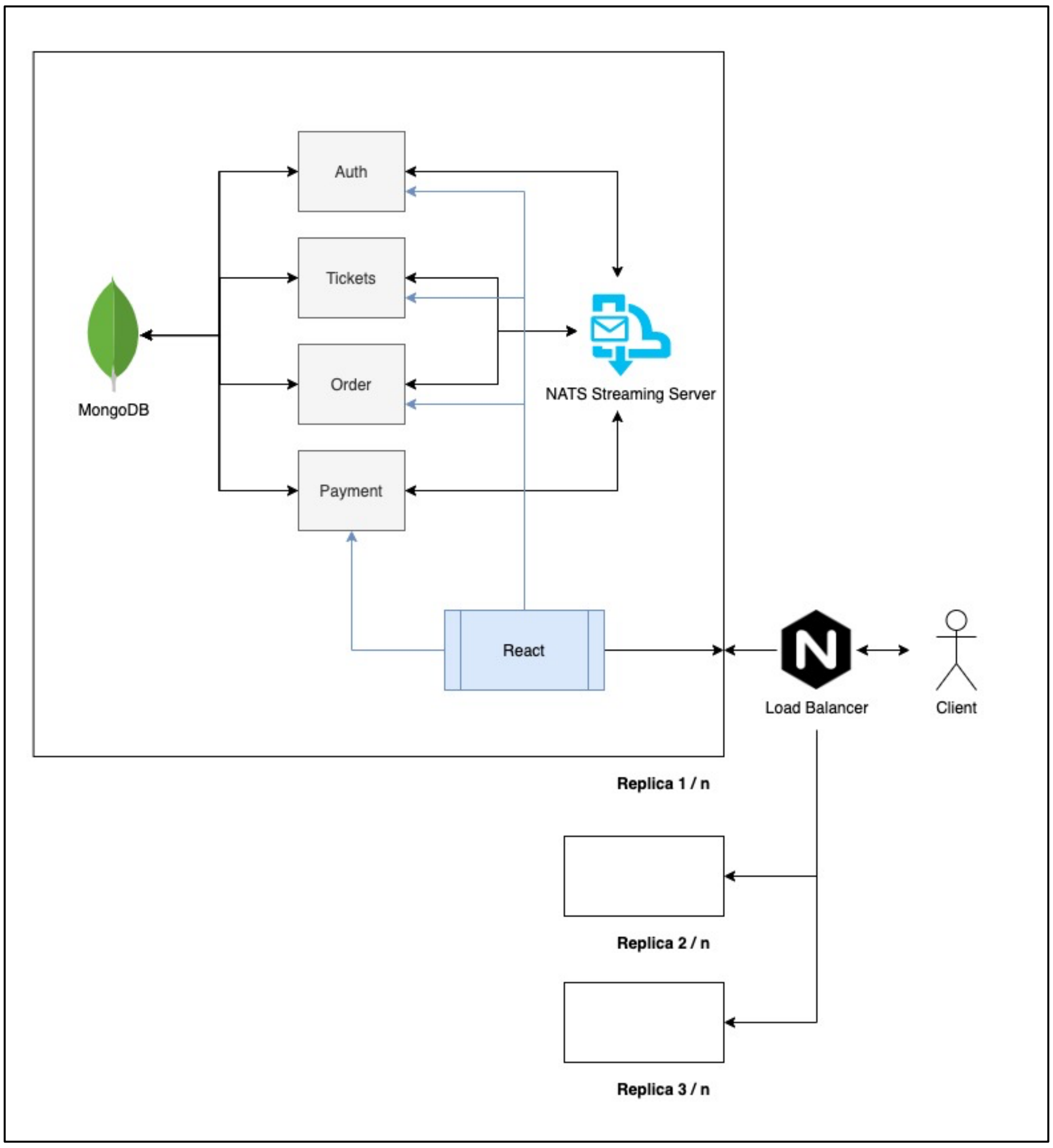
1. Ticket title
2. Ticket description
3. Ticket price
4. Ticket image
5. Billing info to Stripe

Nonfunctional Requirements

- **Web Application**
  1. Performance: The application must not take a long time to load or process information, and shall not take much of processing power from the client
  2. Ease of use: Users should require no training or go through some confusions to navigate the app.
  3. Security: The authentication system must protect users billing info and personal info and other private information.
  4. Graceful failure: In the event of the application failure, all information that are currently on process by the app should not be lost, and the user data should be saved.
  5. Form factor adaptability: The application should look as good and as functional with each and every size of the user screen.
  6. Software adaptability: The app should have support for older browser of a couple previous generations.
  7. Transactional data submissions: In the event of a lost network connection or otherwise failed data transmission, the app should not lose the data that are currently exchanged.
  8. Minimal resource usage: The app will minimize user side rendering as much as possible
  9. Security: The app should not leak users account information, or store the user information in plain text

- **Web Server**
  1. Availability
     a. The database should handle thousands of requests per day
     b. The database should be able to store up to 30,000,000 records
  2. Security
     a. The server should not have unauthorized person to access the app's database
  3. Maintainability
     a. The database should allow incremental new data fields

Architecture Diagram



Functional Decomposition

- User Interface

The application will provide beautiful interface to the user of the application using React framework. The interface allows the user to create ticket, update, and purchase a ticket. My main focus with the UI is to make first time user to be able to navigate the app easily.

- Web Server

The web server will be using API to communicate to each of the services using NATS Streaming server as its message bus.

- **Database**

The databases that are used is MongoDB, with each services have separate database for data consistency

- **Web Application**

The web application was developed in JavaScript and Express.Js for its backend framework. The application provides users with the ability to perform queries to the database through the UI.

- **Payment Service**

The application will be using third-party application called Stripe and communicate using API

- **Auth Service**

This service will be responsible to handle user creation, user login, and token creation to the browser cookie using JWT.

- **Ticket Service**

This service will be responsible to handle ticket creation, ticket edit, and ticket listing or remove it from the list.

- **Order Service**

This service will be responsible to handle ticket purchasing, order expiration creation.

- **Message Bus Service**

This service is responsible for all the communication within the app between each service and manages all the information that it receives.

System Analysis

- **Stripe**

The app uses stripe for payment because it is easier to setup and easier for customer to have a trust in and also stripe provides a better user experience.

- **Web Service**

I chose the web for the first step to solve the problem is because the web is not limited to device or the latest software. The web has a broader customer range and it is available anywhere without any strings attached to the device.

Detail Description

I/O Specifications

Arrows indicate the direction of information flow

- **Client Device → Ingress**

The ingress will load balance the users request to whichever replicas of a service has the least amount of load.

- **Ingress → Web Server**

The ingress will pass the users request to the web server that which hosts the applications content.

- **Web Server → Database**

The web server will then connect with the mongo database.

- **Payment Service → Stripe**

The payment service that are used by order service will communicate with stripe using a secret key to process its payment and using its API.

User Interface Specifications

Using the UI, the user should be able to create account, edit its information, create a ticket, and also purchase an existing ticket. With several different people expected to use the app, ranging from enthusiast to casuals, I will need to create an app that is easy to use while supporting a wide range of users. The app also needs to be responsive regardless of the user screen size.

Software Specifications

- Express Web Server version **4.17.1** or newer
- Mongoose version **5.12.13** or newer
- Typescript version **4.2.3** or newer
- Json web token version **8.5.1** or newer
- Node-nats-streaming version **0.3.2** or newer
- React version **17.0.2** or newer
- NextJS version **11.0.0** or newer
- Bootstrap version **5.0.2** or newer
- Stripe version **8.165.0** or newer
- Ingress (**networking.k8s.io/v1**)

Database: I will be using MongoDB for database

Implementation Issues and Challenge

One of my biggest challenges is getting familiar enough with microservices architecture approach to have an app that is ready to use and good enough to be an actual solution. Another one of my biggest challenges is the consistency of data flow that are passed through each of the service and creating versions of database with each service. Other challenge is to make sure no message is lost in the event of a service failure. Challenge that may arise is when I need to revamp or do a big update, since the code is not as clean code as it could be.