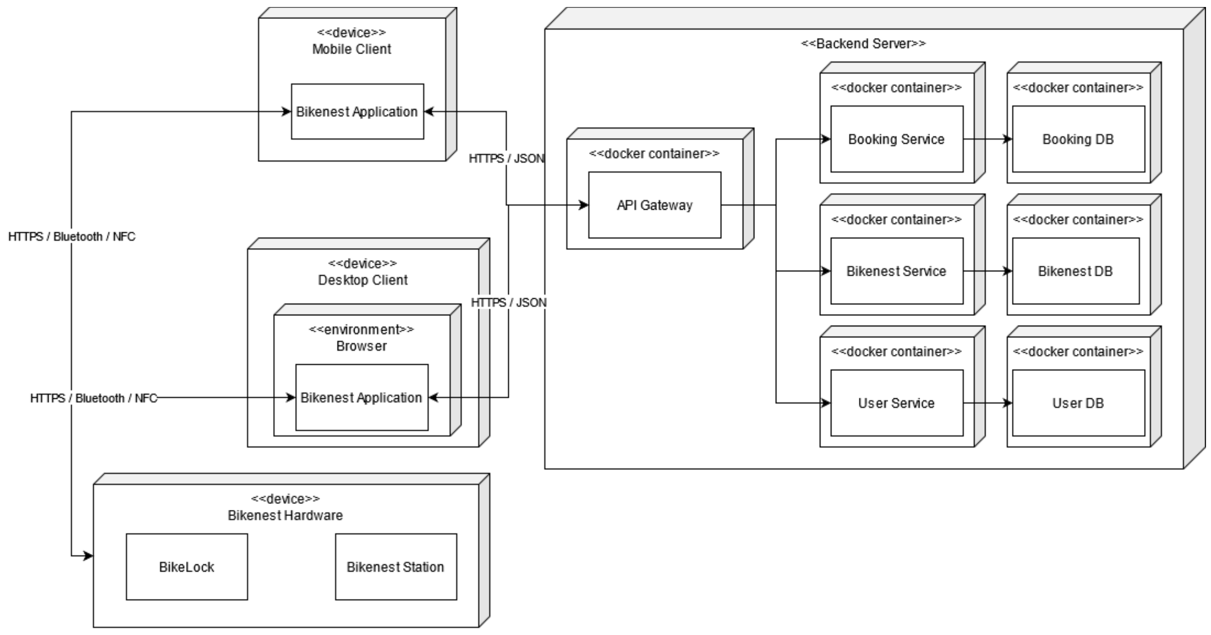
Software Architecture Description

AMOS SS 2021

BIKE NEST

Project 7

Runtime Components

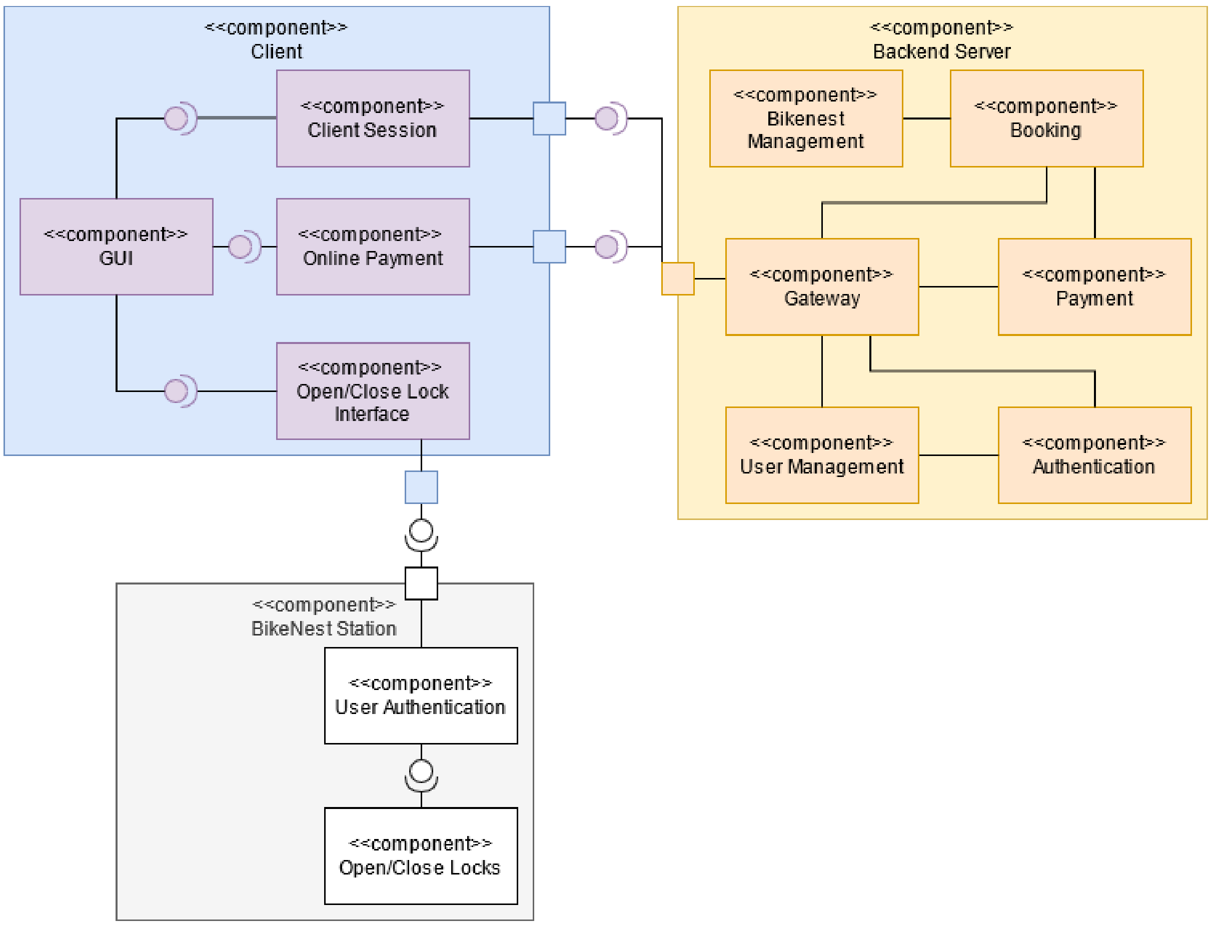


<<device>> Mobile Client
Bikenest Application
HTTPS / JSON
HTTPS / Bluetooth / NFC
<<device>> Desktop Client
<<environment>> Browser
Bikenest Application
HTTPS / JSON
HTTPS / Bluetooth / NFC
<<device>> Bikenest Hardware
BikeLock
Bikenest Station
<<Backend Server>>
<<docker container>> API Gateway
<<docker container>> Booking Service
<<docker container>> Booking DB
<<docker container>> Bikenest Service
<<docker container>> Bikenest DB
<<docker container>> User Service
<<docker container>> User DB

On the Backend Server we will run a Microservice Architecture Style where each Microservice and their databases run isolated inside a docker container.
The individual services are exposed through the API Gateway, which will route requests to the individual microservice.
On the client side we have the Bikenest Application that can run on mobile devices (Android/iOS) as native applications or that can run on desktop clients via browser.
The communication between the Bikenest Application and the Backend will happen via HTTPS using JSON messages.
Some of the available services require authentication via JWT (JSON Web Token).
JWTs will be issued by the Authorization Service. If the API Gateway gets a request that contains a JWT, it will ask the Authorization Service to validate the JWT. If it can not be successfully validated, it is stripped away from the request and the request is forwarded to the chosen Service. The service then decides by itself, if authorization is required, and if it is, it will send back a 401 (Unauthorized) response.

The Bikenest Hardware can be accessed from the Bikenest Application either via HTTPS, Bluetooth, NFC, or other suitable technologies. Using an approach where we are able to deploy our app as a native app is important to have as much functionality (Bluetooth, NFC, etc.) available, because so far the final decision on the Bikenest Hardware has not been made and it's uncertain how exactly we will be able to access it.
If the application should be run in a production environment, it is definitely required that there is a connection between the Bikenest Stations and the Backend Server. Else there would be major security issues, however these are not relevant for prototyping/demo purposes.
For communication between the Microservices we use synchronous API Calls and no messaging bus, because it is less complicated and for demo purposes synchronous calls scale well enough.

Code Components



We group the architecture into Server, Client and Bikenest Station. The implementation of the Bikenest Station is not part of our work, so there are no details provided. It will have to provide some functionality to authenticate the user and to open/close the locks, so that only the owner of the bike will be able to take it. Currently it is planned to have a cage that can be locked and to have individual spots for the bikes that can be locked.
Either the client or the server will be communicating with the BikeNest Station but for now we orient our architecture to the use case of having the communication between the Client and the Station. For this there are multiple options (Bluetooth, Internet, NFC) available, while for the Server the only real option would be a communication channel via Internet.
One of the aims of the Bikenest Hardware is to be as low tech as possible. Therefore it would be favorable to avoid the need for an internet connection.

The communication between the Client and the Server happens via API calls using HTTPS and JSON as message format. The server is split into different components, each taking on a distinct job. Some components need to access other components to fulfill their job. For example the Bikenest Management Component will provide details about the number of spots left in each Bikenest Station and therefore it has to communicate with the Booking Component.

Underlying Technology Stack

Client

- Mobile: iOS, Android
- Web browser

Developing

- UI frameworks and libraries: React Native, Node.Js
- Backend: Java with Spring Framework
- Programming languages: JavaScript, TypeScript, Java
- Other frameworks and platforms: Expo
- Developing tools: Visual Studio Code or IntelliJ recommended but not mandatory
- Databases: MySQL
- Docker Containers for Backend Services
- APIs: Third party providers for login and payment

Description about the technologies

React Native:
React Native is a framework that builds a hierarchy of UI components to build the JavaScript code. It has a set of components for both iOS and Android platforms to build a mobile application with a native look and feel.
We choose it because it has all the needed requirements we have so far and is a popular framework with a lot of community support. Components are reusable, single code as it is a cross platform. In case of need we could also build native code.

Spring Framework:
All in One Solution for Java Development that is often used for Web Applications. Provides ways of accessing databases (connection and OR Mapping), setting up RESTful services, integrating Security, supports Cloud specific tasks like Gateways and much more. One important aspect for us is the huge support for microservice specific tasks like inter service messaging.

Developing tools:
Visual Studio Code extension that provides a development environment for React Native projects is recommended but any other IDE that can work with React Native is also accepted.

Expo:
Expo is a toolchain built around React Native to help you build native iOS and Android projects using JavaScript/TypeScript and React. Expo enables you to build cross-platform native apps using only JavaScript. Benefits of using Expo is no need to use Xcode or Android Studio. You can build managed and bare workflows, Apps are built with the managed workflow using the expo-cli, the Expo Go app available for mobile devices and access to SDK. In case of need we can also opt to go bare workflow that will build native code.

MySQL:

MySQL is one of the most used open-source relational database management systems in the world. It uses the client-server architecture where a client sends requests to the server specifying the data it wants to retrieve. A lot of web services use MySQL to store their data and Spring also supports MySQL so it’s a good choice for our purposes.