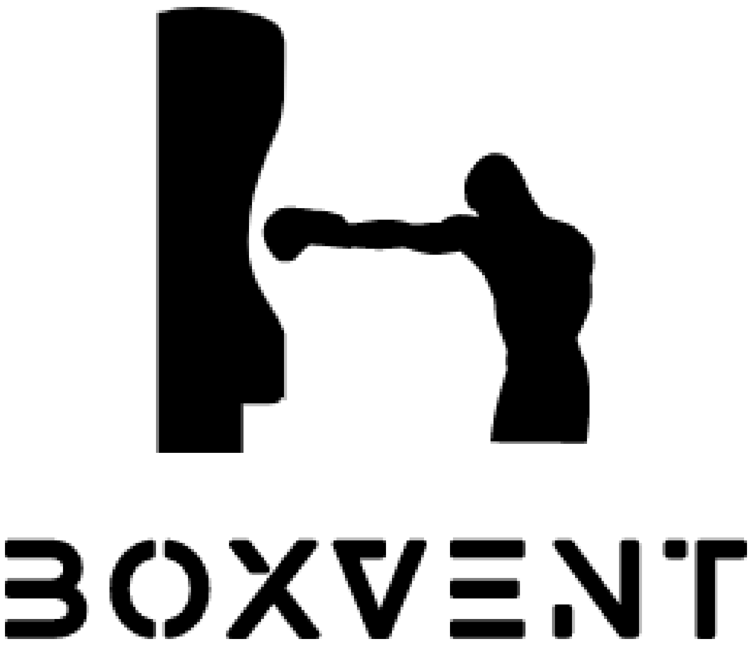
Design document



Vlad Dumitru – 3367231

Student at Fontys University of Applied Sciences – S3CB04

Sprint 3

Document version 2.0

Table of Contents

Product Description

We will be creating a website that will be used by the company BoxVent to display and manage upcoming boxing events, manage fighters and sell tickets. The website will give the possibility to the client to create an account, see upcoming fights and the fighters line up as well as purchasing different types of tickets for any available event. The administrator of the website will have the ability to create new events, add/edit/delete fighters, see different statistics about the events (number of tickets sold/remaining).

High level architecture

During the development of the product the SOLID principles will be applied to ensure the possibility of extending the website, making it more flexible, maintainable and adaptable.

There are five SOLID principles that focus on distinct concerns of object-oriented design which help with extending the system with new functionalities that don't break previous implemented functionalities. I will give an explanation of each principle shorty after which I will give an example of how it's applied in our website.

Single Responsibility Principle (SRP)

The SRP is pretty self-explanatory from the name but what it means basically is that one class/method/component/etc should have only one responsibility. This is done for easier manipulation of the code and readability and it makes the classes independent of each other.

The source code of the product respects this by separating each page and each of that pages component (frontend). For example, for adding a new fighter we have a create fighter page which renders an InputFighter component which has the responsibility of showing an input form and it has multiple methods for handling each part of the input form(boxer name, boxing record, submit). The same principles are applied when it comes to the backend.

Open and Closed Principle (OCP)

The OCP is more or less a set of rules that you have to follow to ensure a functioning application through the development process. This principle means that an entity should be open for extension but closed for modification. Which means that previously written functions, classes, etc. should be extendable but not modifiable. For example in our backend we have a class that manipulates the database, we have a method called GetFighterbyId(long id) if we would want to also get the fighter by his name for example, we wouldn't modify the parameter but create a new method called GetFighterByName(String name).

Liskov Substitution Principle (LSP)

LSP expands on the Open/Closed Principle which says that If class A is a subtype of class B, then we should be able to replace B with A without disrupting the behavior of our program. This is achieved in our backend by creating an interface for which a class should implement it. In our project this is applied to each type of use case, first we create an interface and then we create an implementation for that interface.

Interface Segregation Principle (ISP)

ISP is the separation of interface so that the dependency between them should be as minimal as possible. The goal of this principle is to reduce the side effects and frequency of required changes by splitting the software into multiple, independent parts. In our project so far instead of having one giant interface for all the use cases we made multiple small ones for each use case, in the future we might also have classes that inherit from two different interfaces while other classes inherit from just one of them.

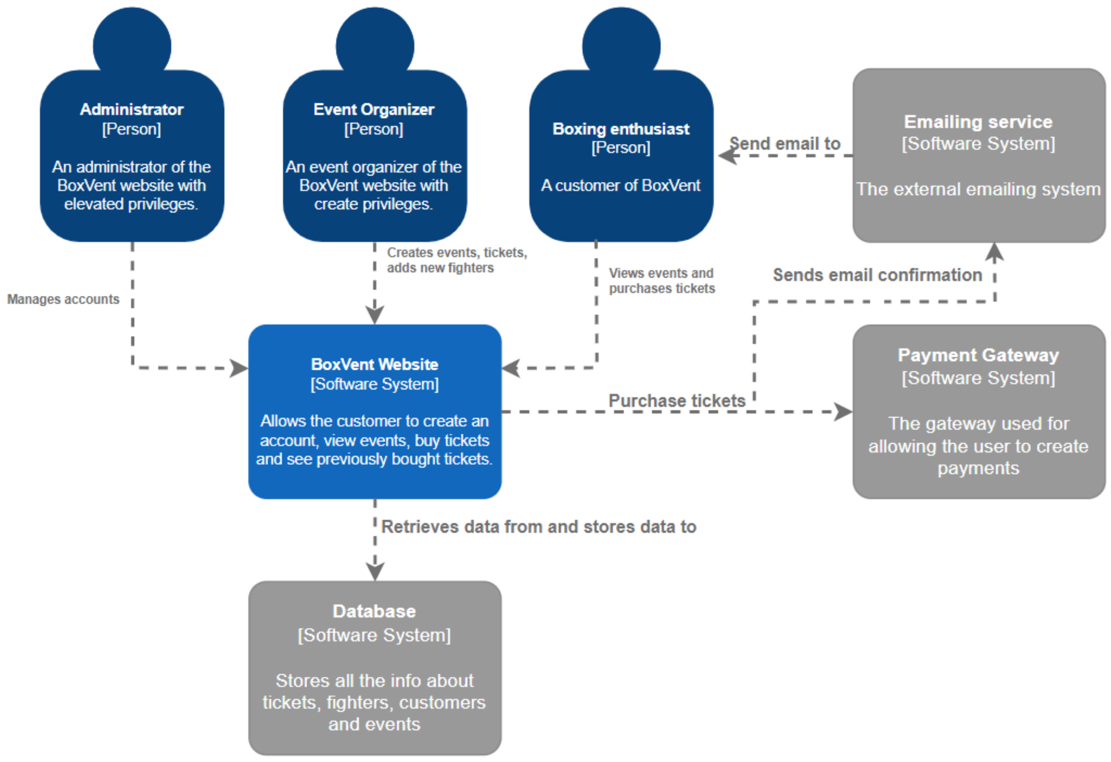
Dependency Inversion (DIP)

DIP is referring to the fact that High-level modules should not depend on low-level modules. They should be minimally affected by their actions, or not at all ideally. This method was described as the Hollywood Method (“Don’t call us, we’ll call you”), this is often achieved by “bridging” higher-layer packages with lower-layer packages. This could be applied in our website for example when buying a ticket, the high-layer logic would have a class, “Transaction” which will implement from an interface called “Action”. When buying a ticket all the necessary information is processed such as how much the ticket cost, where the person should seat. Now, if we wanted to implement a free ticket for a free event and we would implement the “Action” interface we will have some empty methods for handling the payment of the ticket. To fix this we separate the “Action” interface into multiple interfaces and we implement only the ones we require.

C4 architecture diagrams

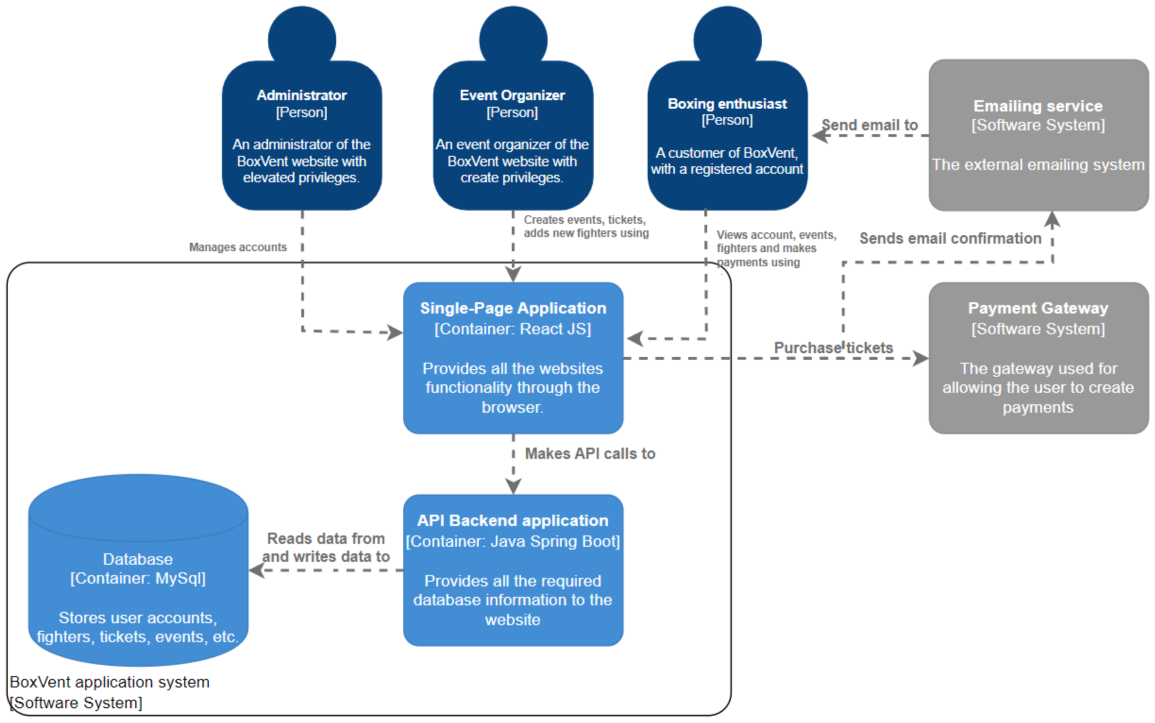
C1 diagram

The C1 diagram depicts the 3 users that will access the software system. The Administrator will have the ability of making/deleting/modifying user accounts and rolls. The Event organizer will have the ability of creating events and tickets as well as the ability to add new fighters. The customer has the ability of viewing the events and purchasing tickets to them. The diagram also depicts the communication with the database from the main system as well as communication with the payment gateway and the external emailing service.


Administrator
[Person]
An administrator of the BoxVent website with elevated privileges.
Event Organizer
[Person]
An event organizer of the BoxVent website with create privileges.
Boxing enthusiast
[Person]
A customer of BoxVent
Emailing service
[Software System]
The external emailing system
Send email to
Manages accounts
Creates events, tickets, adds new fighters
Views events and purchases tickets
Sends email confirmation
BoxVent Website
[Software System]
Allows the customer to create an account, view events, buy tickets and see previously bought tickets.
Purchase tickets
Payment Gateway
[Software System]
The gateway used for allowing the user to create payments
Retrieves data from and stores data to
Database
[Software System]
Stores all the info about tickets, fighters, customers and events

C2 diagram

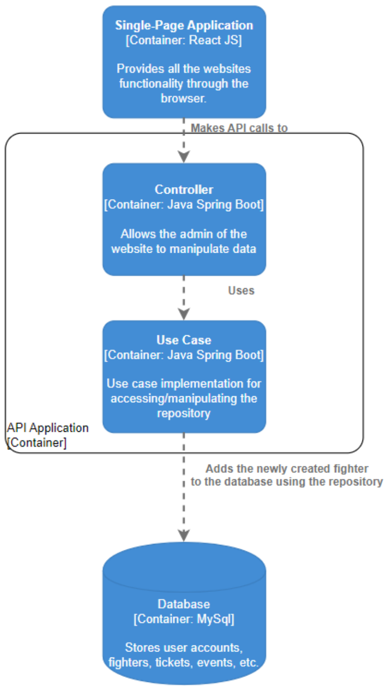
The C2 diagram shows how the Single-Page application will be used by the users and how it will communicate with the backend API to retrieve necessary data. Communication of the API backend and the Database is also depicted, which will allow the backend to read and write information from the database. The connection between the Single-Page Application and the two external systems is also depicted.



Administrator
[Person]
An administrator of the BoxVent website with elevated privileges.
Event Organizer
[Person]
An event organizer of the BoxVent website with create privileges.
Boxing enthusiast
[Person]
A customer of BoxVent, with a registered account
Emailing service
[Software System]
The external emailing system
Send email to
Manages accounts
Creates events, tickets, adds new fighters using
Views account, events, fighters and makes payments using
Sends email confirmation
Single-Page Application
[Container: React JS]
Provides all the websites functionality through the browser.
Purchase tickets
Payment Gateway
[Software System]
The gateway used for allowing the user to create payments
Makes API calls to
Database
[Container: MySql]
Stores user accounts, fighters, tickets, events, etc.
Reads data from and writes data to
API Backend application
[Container: Java Spring Boot]
Provides all the required database information to the website
BoxVent application system
[Software System]

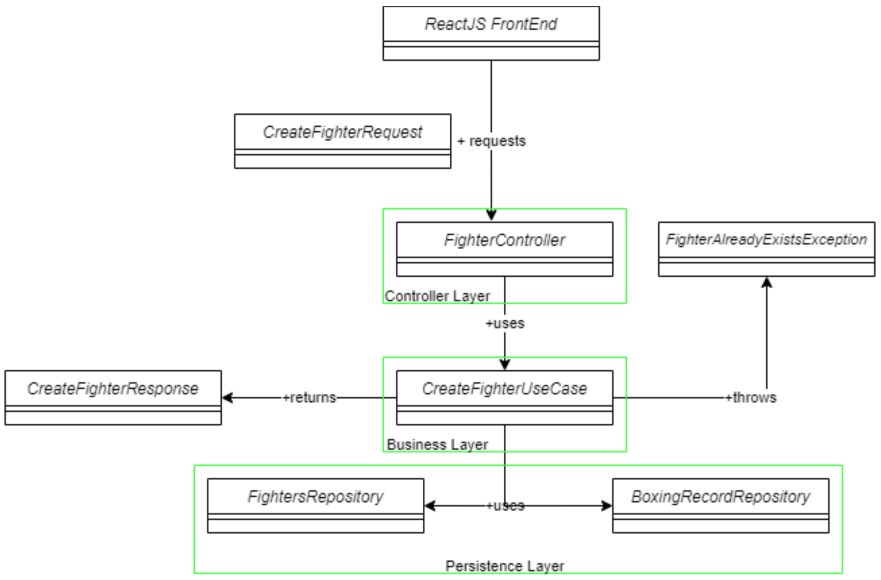
C3 diagram

The C3 diagram shows how the application will communicate with the backend API's controller and how the controller will use a use case for manipulating data, which in turn will use a repository which accesses the external MySql Database.


Single-Page Application
[Container: React JS]
Provides all the websites functionality through the browser.
Makes API calls to
Controller
[Container: Java Spring Boot]
Allows the admin of the website to manipulate data
Uses
Use Case
[Container: Java Spring Boot]
Use case implementation for accessing/manipulating the repository
API Application
[Container]
Adds the newly created fighter to the database using the repository
Database
[Container: MySql]
Stores user accounts, fighters, tickets, events, etc.

C4 diagram

The C4 diagram depicts the functionality of the backend application, clearly showing the separation of layers. The front end application will send a HTTP Request with a JSON Body containing the fighters name and boxing record to the back end controller. The controller takes this request and accesses the required use case with the request. The use case tries to create a new fighter and a new record to associated with the fighter. If this is not possible the appropriate exceptions are thrown. The persistence layer is used by the use case to create the fighter and boxing record. The repositories then send the required information to the database and return the appropriate response.



Design decisions and applied research

The software tools that will be used for developing the website are the following: spring boot, Gradle and MySQL for the backend and React JS with Axios and JSX for front end. For version control we will be using git with Gitlab. The reason why we are using Gitlab is so that we can implement continuous integration in our website.

Spring boot, Gradle and MySQL

Spring boot is a platform for Java used for developing stand-alone and production-grade spring applications, it’s much easier to start without the need for an entire Spring configuration setup. It’s also easy to understand and use, increases productivity and drastically reduces the development time. Another reason why we decided to use spring boot is the fact that it manages REST endpoints very well it’s also very flexible when it comes to database transactions. It’s also easy to start since it’s auto configured and no manual configurations are needed.

For building and automatically performing tests we use Gradle, which is a tool that focuses on maintainability, usability, extendibility, performance and flexibility. It is a better choice since it includes the pros of previous tools such as Ant and Maven while also curbing the cons of both.

We will be using MySQL for managing our database since it’s one of the worlds most popular open source databases which provides adequate support for every application development needed, it also saves time and money.

React JS, Axios and JSX

One of the main reasons we picked React JS is due to its potential to reuse components, which in turn saves time on the development process giving the developer the opportunity to reuse his code if the code applies SOLID principles and best practices of course. When developing an app, changes made in any particular part will not affect other parts of the website application. It also gives us the ability to work in chunks by only re-rendering what has changed and not the whole page, with this functionality it will also be pretty easy to build a single page website.

React JS comes with JSX which combined with reacts components it gives you the ability to write your own custom components. These type of components accept HTML quoting and thanks to JSX they can be combined with the full power of javascript.

Axios is not as much a choice as it is a necessity since it allows our React apps to communicate with APIs easily, although there are other methods we could retrieve this data but Axios provides a little more functionality that goes a long way when using React.

Why are you using spring boot, how do you separate concerns? What framework are you using and why?)

Separation of concerns

Separation of concerns is a key element when building layered applications, at a low level, this principle is related to the SOLID principle known as Single Responsibility Principle. We will achieve this by establishing boundaries, for example using methods, objects, components and services to define the core logic of our application and folder hierarchies for source organization. By applying

SOLID principles and separation of concerns we ensure that our application will be scalable, reusable and maintainable.

For our java application we will be separating the concerns in a horizontal separation which refers to the process of diving the application into controller layer, business layer, persistence layer and database layer. This is also the preferred way of working when it comes to spring boot.

The controller layer is the most top layer of the spring boot architecture, it handles HTTP requests and performs authentication. It's also the layer where the URL mapping of the website is done. The business layer is pretty self explanatory, it contains all the business logic which include service classes, it's also the layer that's responsible for validation and authorization. The persistence layer contains all the database storage logic, it's used for converting business objects to database objects. The database layer contains the MySQL database and is responsible for performing CRUD operations.

References

West, Z. (2021, July 21). *SOLID: Guidelines for Better Software Development*. Alphαrithms. Retrieved October 7, 2022, from https://www.alpharithms.com/solid-guidelines-for-better-software-development-055500/

Jaludi, M. (2021, December 10). *Software Design Principles: SOLID - Mariam Jaludi*. Medium. Retrieved October 7, 2022, from https://mariam-jaludi.medium.com/software-design-principles-solid-ef0c3d5b008a

Spring Boot - Introduction. (n.d.). Retrieved October 7, 2022, from https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

Gaba, I. (2022, August 18). *What is Gradle? Why Do We Use Gradle? Explained*. Simplilearn.com. Retrieved October 7, 2022, from https://www.simplilearn.com/tutorials/gradle-tutorial/what-is-gradle

MySQL :: Why MySQL? (n.d.). Retrieved October 7, 2022, from https://www.mysql.com/why-mysql/

Peerbits. (n.d.). *The benefits of ReactJS and reasons to choose it for your project*. Retrieved October 7, 2022, from https://www.peerbits.com/blog/reasons-to-choose-reactjs-for-your-web-development-project.html

Introducing JSX –. (n.d.). React. Retrieved October 7, 2022, from https://reactjs.org/docs/introducing-jsx.html

GeeksforGeeks. (2022, March 11). *Spring Boot - Architecture*. Retrieved October 7, 2022, from https://www.geeksforgeeks.org/spring-boot-architecture/