IBU | International Burch University

CEN 308 SOFTWARE ENGINEERING

PROJECT DOCUMENTATION

INDIGO EVENTS

Prepared by:
Adi Lagumdžija
Dženana Međedović

Proposed to:
Nermina Durmić, Assist. Prof. Dr.
Aldin Kovačević, Teaching Assistant

24.06.2022.

TABLE OF CONTENTS

Contents

1. Introduction

This document contains all the need-to-know information about our Software Engineering project that is commited to GitHub and deployed to Heroku. We decided to create a web application about which we are going to say more in the next section of this document.

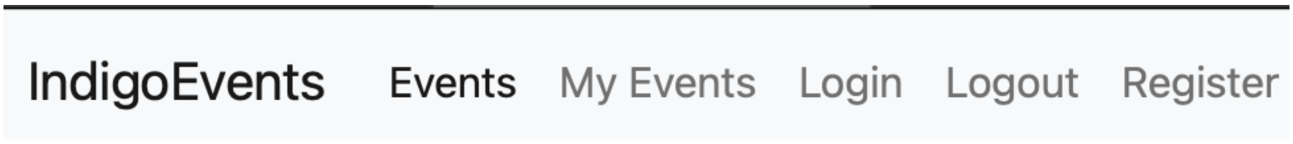
1.1. About the Project

For our project we decided to create an Events Web App. Its purpose is to showcase available events to users in their or any other city. The user will also be able to make a reservation for the event that he/she is interested in and review his/her past reservations. The idea came from looking back at SFF and other very interesting events which people may want to know more about. Those events were packed with all kinds of different content. There was something for everyone. We wanted to give users the ability to review events like that online more easily and make their reservation accordingly. You can find the project at: https://indigo-events-app.herokuapp.com/index.html

1.2. Project Functionalities and Screenshots

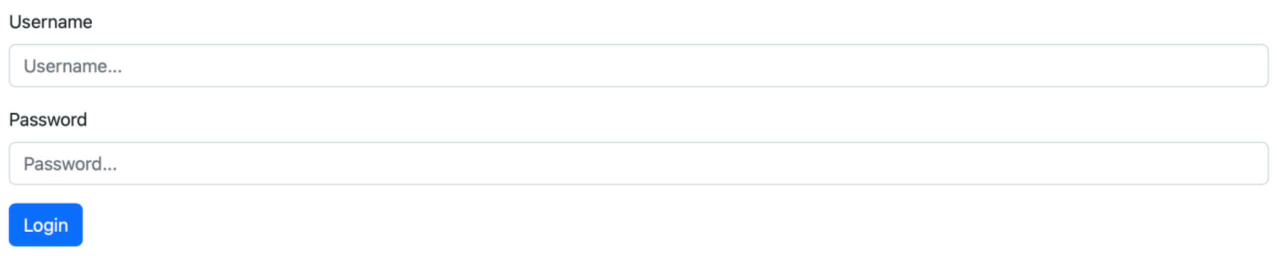
Describe or list the main features of the application and provide a few screenshots of your project.

1.2.1 Menu



Navbar with 5 options.

1.2.2 Login

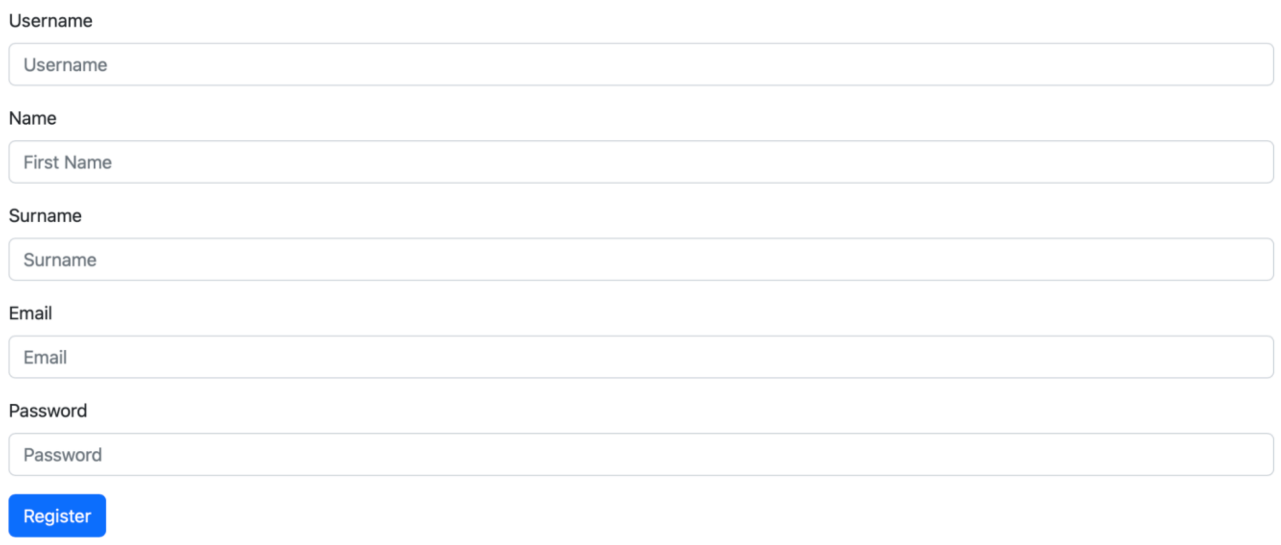


Full-functional login option.

1.2.3 Logout

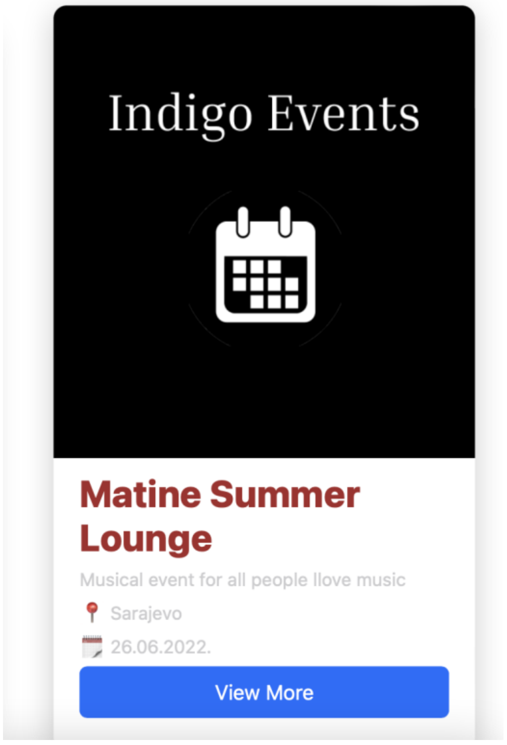
Logout option that clears local storage and redirects to index.html.

1.2.4 Register



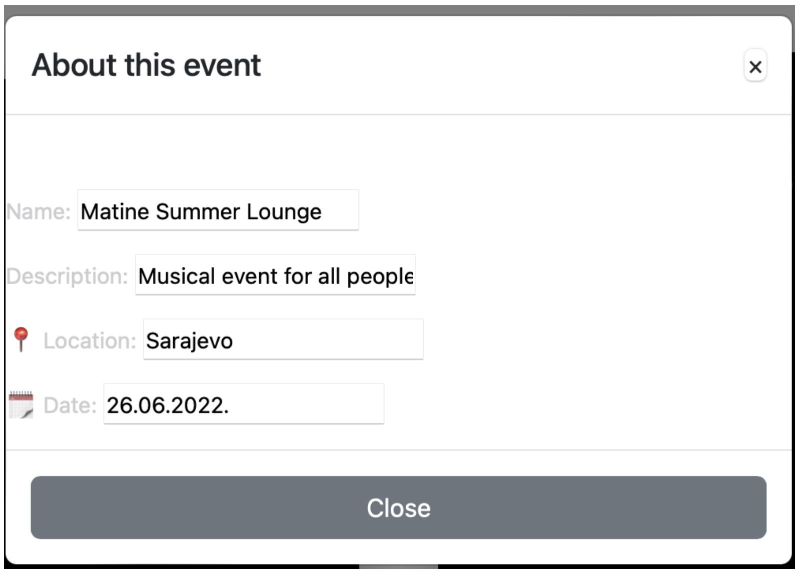
Full functional register option with JWT Token. It has an email confirmation.

1.2.5 Event page



Event page, all events are made from admin.html page, where you can add new events

1.2.6 View more option



Name: Matine Summer Lounge

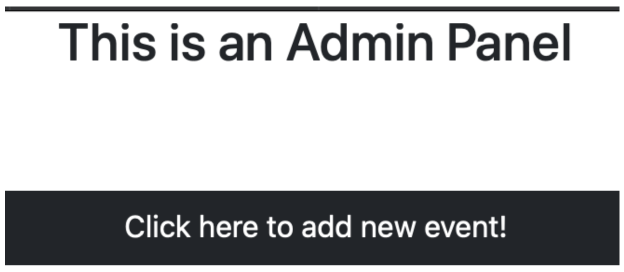
Description: Musical event for all people

Location: Sarajevo

Date: 26.06.2022.

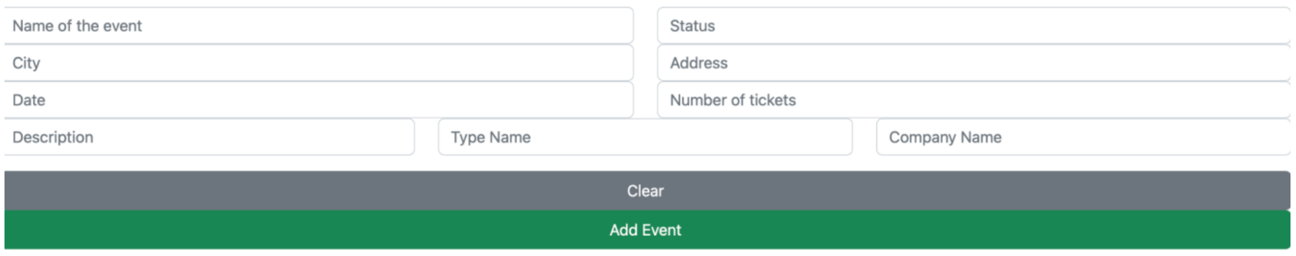
View more option provides detailed info about the event.

1.2.7 Admin panel



Admin panel is made for admins, where new event can be added.

1.2.8 Add new event form



Add new event form is made for making a new event that is later displayed on landing page. All data will be added to the database.

2. Project Structure

2.1. Technologies

When it comes to storing our data, we have decided to create a database and maintain it in **MySQL Workbench**. The database was, however, deployed to Heroku in order to be able to extract data from it and feed data to it.

The programming language used for communicating with the database (backend) was **PHP**. PHP (recursive acronym for `PHP: Hypertext Preprocessor`) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

Frontend languages used within this project are: **HTML**, **JS** and **CSS**. We also decided to use jQuery which is a fast, small, and feature-rich JavaScript library to help us with developing this app. Besides that we also used some Bootstrap templates to make the whole process easier.

The coding standards used within the project for the backend were PSR-1&2 for PHP. We used a fixer at the end to make the code look neater and to follow the already mentioned guidelines. For the frontend part we have downloaded an Atom package and used its beautify option (in HTML, JS and CSS files) to gain the same goal as for the backend.

2.2. Database Entities

Provide a list of *tables or entities* you have in your database/schema. If it is not obvious from the name of the table/entity what it is used for, also provide a brief explanation next to it. For example:

- user
- company ✇stores companies which host the events
- event ✇stores all events which are shown on the website
- eventtype ✇stores all the types of events
- reservationdetails ✇stores all the reservation details for each event and user

2.3. Architectural Pattern

For our project we have decided to use the **layered architecture pattern** because we are already familiar with it and find that it to be one of the better and most clear approaches to organizing the system. The **layered architecture pattern** (also called the **n-tier pattern**) organizes the system into *layers* with related functionality associated with each layer. A layer provides services *to the layer above it* so the lowest-level layers represent core services that are likely to be used throughout the system.
In general, the layered architecture pattern recognizes 4 main layers:

- **presentation layer** (also known as *UI layer*)

- **application layer** (also known as *service layer*)
- **business logic layer** (also known as *domain layer*)
- **data access layer** (also known as *persistence layer*)

2.4. Design Patterns

- bridge pattern: used in the backend, in the file *rest/dao/ALL_FILES*
- bridge pattern: used in the backend, in the file *rest/services/ ALL_FILES*

Since bridge is a structural design pattern that lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other, we decided to use it in these two folders. Namely the dao and service folders each contain a Base class (abstraction), where that class contains all methods which can be used throughout the other classes (implementation) located in these folders (EventDao.class.php, EventService.class.php, UserDao.class.php, UserService.class.php, etc.). The Base classes within these folders aren't supposed to do any real work (other than connecting to the database in BaseDao) on its own. They should delegate the work to the other classes within these files. What does that mean? The Base classes are going to define all the main methods that are going to be used within the other classes and modified as per need of each class. If we didn't do this, we would have been creating a lot more code, which in some parts would be the same or similar meaning that we are wasting time and resources.

- singleton pattern: used in the backend, in the file *rest/Database.class.php*

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance. The perfect opportunity to use this design pattern presented itself for accessing our database since the connection is supposed to be established only once. The database connection class acts as a **singleton**. This class doesn't have a public constructor, so the only way to get its object is to call the getInstance method (used in BaseDao.class.php). This method caches the first created object and returns it in all subsequent calls.

3. Conclusion

We think that we did okay, however we also think that we lack some knowledge when it comes to the frontend considering all the technologies we used. A lot more effort was put in to the backend part of our app – which is logical since all of the functionalities are there. This presented itself as a challenge, but also helped us learn and adapt throughout the whole process. Something that was very annoying was the shared GitHub repository because of all the pulls and merges we had to do which would make our app non-functional until we went over the whole code and found the bug that was causing it. Another issue we faced is the photos which should be displayed for each event. We didn’t find a way to properly fetch and display them. In the end, there is always room for improvement, including this project as well and we will make sure to continue working on it to make it more presentable.