

Web-aplikacija za izgubljene i pronađene stvari

Macukić, Marcel

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:168:968801>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2025-03-14**



Repository / Repozitorij:

[FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repository](#)



SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 10

WEB-APLIKACIJA ZA IZGUBLJENE I PRONAĐENE STVARI

Marcel Macukić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 10

WEB-APLIKACIJA ZA IZGUBLJENE I PRONAĐENE STVARI

Marcel Macukić

Zagreb, lipanj 2024.

ZAVRŠNI ZADATAK br. 10

Pristupnik: **Marcel Macukić (0135252557)**

Studij: Computing

Modul: Computing

Mentor: prof. dr. sc. Boris Milašinović

Zadatak: **Web-aplikacija za izgubljene i pronađene stvari**

Opis zadatka:

Izraditi web-aplikaciju koja bi omogućila evidenciju izgubljenih i pronađenih stvari. Korisnik bi mogao prijaviti nestanak, prijaviti pronalazak te pretraživati objave. Prilikom prijave nestanka omogućiti korisniku da na karti označi područje na kojem smatra da je predmet nestao, a korisniku koji je pronašao stvar omogućiti točan unos lokacije označavanjem na karti ili s pomoću GPS-a. Korisnici mogu unijeti fotografije pronađene ili izgubljene stvari, podatak gdje se izgubljena stvar treba dostaviti, odnosno može pokupiti, pri čemu kontakt podatak ne mora biti javno vidljiv, pa je unutar aplikacije potrebno podržati razgovore između korisnika.

Rok za predaju rada: 14. lipnja 2024.

Contents

1	Introduction	3
2	Software Specification	4
2.1	Functional Requirements	4
2.1.1	Actors	4
2.1.2	Use Cases	4
2.2	Conceptual Model	9
3	Technologies Used	10
3.1	Frontend	10
3.1.1	React	10
3.2	Backend	11
3.2.1	.NET	11
3.3	Database	13
3.3.1	MSSQL	13
4	Implementation	14
4.1	Backend	15
4.2	Frontend	20
4.3	Authentication	21
4.4	Directory Layout	23
4.5	Web Service	24
5	Application Showcase	27
6	Installation Manual	31

7 Conclusion	32
8 Bibliography	33
Abstract	34
Sažetak	35

1 Introduction

In today's fast-paced world, losing personal belongings has become an everyday problem. Once you have realized you have lost something, the hardest part is to remember where it could be. To address this issue, I have developed a web-based application designed to simplify the process of reporting and recovering lost and found items. The idea is that you have a place where you can have a quick look and see if the item you have lost has been found by someone. If not, you can still make a post specifying what you lost and where you think you lost it.

The web-based nature of this application is intentional. Unlike apps that require frequent use and constant presence on a user's phone, this application addresses a specific need that does not occur daily. Users can conveniently access the platform from any web browser without the need for a permanent app installation on their devices. You could make a post that you lost something on your phone, go home and check up on it on your laptop/PC.

2 Software Specification

In this section the focus will be on the actions that the web application needs to perform and its intended use. The application is implemented as a single page application, meaning that once it is compiled and running, there is one page that is loaded and after that all of the modifications are done by the frontend. This allows the users to navigate the application without constantly loading pages from the backend which can be really slow. Single page applications however require more effort to maintain state. Running the application is configured to be launched at once, as a SPA proxy. This means that it is enough to launch the backend and the frontend will be launched automatically.

2.1 Functional Requirements

2.1.1 Actors

- Unregistered user
- Registered user
- Listing author

2.1.2 Use Cases

- **View homepage**
 - Actors: Any website visitor
 - Goal: Reading about the website

- Basic course:
 - * User types in the website URL.
 - * Users browser loads the website homepage.

- **Register**

- Actors: Unregistered user
- Goal: Registering an account
- Basic course:
 - * User types in the website URL
 - * Users browser loads the website homepage
 - * User navigates to the registration page
 - * User inputs the required data
 - * User registers an account
- Possible deviations:
 - * User inputs the wrong data
 - * Username already chosen
 - * Password too weak

- **Log in**

- Actors: Registered user
- Goal: Authenticating the user
- Basic course:
 - * User types in the website URL
 - * Users browser loads the website homepage

- * User navigates to the Profile page

- * User inputs the required data

- * User Logs in to his profile

- Possible deviations:

- * User inputs the wrong data

- **Report lost/found item**

- Actors: Registered user

- Goal: Create a listing with the information about the item

- Basic course:

- * User navigates to the listings page

- * User clicks on the "Report lost/found item" button

- * User puts the required information into the form

- * If the user allows, the application finds users location

- * The user can upload a photo of the item

- * User posts the listing

- Possible deviations:

- * User inputs the wrong data

- * User not logged in

- * User does not input all of the required data

- **Message user**

- Actors: Registered user

- Goal: Send a message to the user who posted the listing

- Basic course:
 - * User navigates to the listings page
 - * User clicks on the message user button
 - * User types the message and sends it to the owner of the listing
- Possible deviations:
 - * User not logged in

- **Delete listing**

- Actors: Listing author
- Goal: Remove a listing from the database
- Basic course:
 - * User navigates to the profile page
 - * User checks his listings
 - * User chooses which listing to delete
- Possible deviations:
 - * User not logged in

- **Search lost/found items**

- Actors: Registered user
- Goal: Searching through the posted listings
- Basic course:
 - * User navigates to the listings page
 - * Browser loads the listings
 - * User searches through the listings
- Possible deviations:
 - * User not logged in

2.2 Conceptual Model

The conceptual model describes the relationships between entities present in the application. The users can post many listings, and a listing can only be posted by one user. User can send many messages and a message can only be sent by one user, and received by one user. There can be many messages in connection to one listing, but each message can only be connected to one listing.

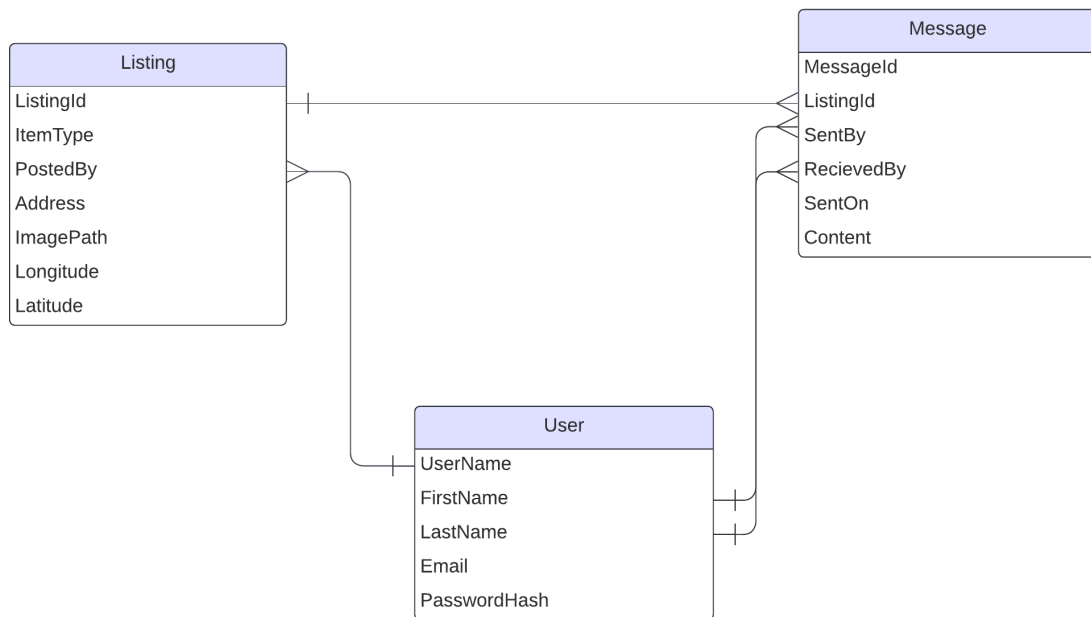


Image 2.1: Conceptual Model

3 Technologies Used

In this section, I will explain the technologies used during the development of the application. The main technologies include React for the frontend, .NET for the backend, and MSSQL for the database. The web application is implemented as a Single Page Application (SPA), ensuring a smooth and responsive user experience.

3.1 Frontend

The frontend of the application is developed using React, a popular JavaScript library for building user interfaces, particularly single-page applications where efficiency and a seamless user experience are critical.

3.1.1 React

React is a powerful library that allows developers to create large web applications that can update and render efficiently in response to data changes. Some of the key features of React include:

- **Component-Based Architecture:** React enables the development of reusable UI components, which help in maintaining a clean and manageable codebase. Each component can be reused in any other component which makes the code very readable and easy to maintain.
- **Virtual DOM:** React uses a virtual DOM to optimize rendering. Instead of updating the entire page when data changes, React updates only the components that have changed, resulting in faster and more efficient performance.

- **JSX Syntax:** JSX, a syntax extension for JavaScript, allows developers to write HTML directly within JavaScript, making the code more readable and easier to debug. An example of JSX syntax can be seen in the ??.
- **State Management:** React's state and props system helps manage the application's state effectively, ensuring that the UI reflects the current state of the data. In React you can share the state of two components to always change together, and to achieve this you remove the state from those components, move it to their closest shared "parent" and pass it to them via props.

3.2 Backend

The backend of the application is built using .NET, a free, cross-platform, open-source developer platform for building many different types of applications. The .NET framework provides a robust and scalable solution for backend services.

3.2.1 .NET

.NET is a free and open-source application platform. With .NET you can build mobile applications, desktop applications, microservices, games, web applications and much more. For developing web application I used the ASP.NET Core, which is an open-source framework that is known for its security and robustness. The language used to write code in ASP.NET Core applications is C#. A general-purpose, high-level, object-oriented programming language that supports multiple paradigms. C# is one of the most used languages today, with some of the applications being AutoCAD, GitHub Desktop, Stack Overflow and many more.

- **ASP.NET Core:** ASP.NET Core is an open-source web development framework that was designed for building modern cloud-based applications. Compared to the original ASP.NET framework, it has several benefits, such as enhanced performance, cross-platform compatibility, and easier development. In contrast to earlier iterations that were limited to Windows, ASP.NET Core is now cross-platform and compatible with macOS and Linux.

- **Entity Framework:** The benefit of using Entity Framework is that you can work with data using objects of domain without worrying about the database tables where this data is actually stored. It allows the developer a higher level of abstraction when dealing with data. This all means that there is much less room for error since most of the work is done by the framework. Entity Framework can connect to an already existing database that has data in it, or it can create a database and its tables based on the Context class.
- **Context Class:** Context class is a core part of the Entity Framework. It allows CRUD operations on the database using a session. Context is used to represent the data from the database in a way that is easiest for the developer to use.
- **Models:** Models are classes that represent data used in our application. The model is used when creating a Context as a blueprint of how the data in the database should be stored. This makes for easier programming and is much less prone to errors. A model I made for listings can be seen in the ??.
- **Controllers:** A controller is a part of ASP.NET Core that is used for handling the incoming Http requests, deciding which view to load and work with the model. It is the part of the backend application that is in the center of all traffic.

3.3 Database

For the database, I used Microsoft SQL Server (MSSQL), a relational database management system developed by Microsoft. The server was running on my personal laptop and then using the Entity Framework a database was created based on the specified models and DbContext.

3.3.1 MSSQL

MSSQL provides a reliable and scalable solution for data storage and management. Some of the benefits of using MSSQL include:

- **Robust Performance:** MSSQL is designed to handle large volumes of data and complex queries efficiently, ensuring high performance and reliability.
- **Advanced Security Features:** MSSQL includes advanced security measures such as encryption, access controls, and auditing to protect data.
- **Integration with .NET:** MSSQL seamlessly integrates with .NET, providing a cohesive development experience and simplifying data access and manipulation.
- **Comprehensive Tools:** MSSQL offers a variety of tools for database management, monitoring, and optimization, making it easier to maintain and scale the database.

4 Implementation

In this section the focus will be on how I used the before specified technologies to implement the application and all of its required functionalities. The application is made to be very responsive (fast response time) because it is implemented as a single page application. The structure of the application can be seen in the image 4.1

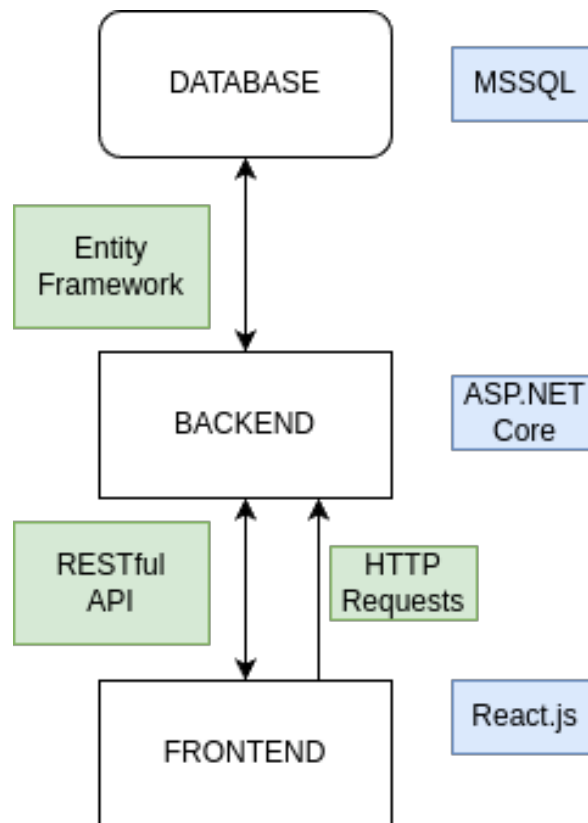


Image 4.1: Diagram of The Application Structure

Database

The database is configured on a MSSQL server running locally on Ubuntu 23.04. The database itself was generated using the Entity Framework and the database context that uses the same models that the backend uses for manipulating the data. This allows for seamless updates and corrections while developing the application.

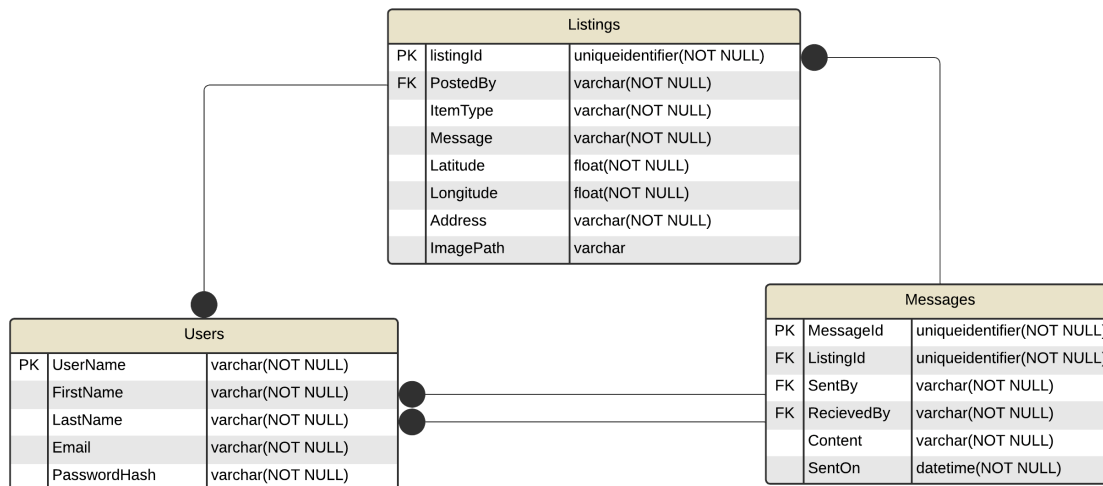


Image 4.2: Database ER Diagram

4.1 Backend

The backend of the application is developed using the ASP.NET Core with addition of Entity Framework. The application was setup in the Rider IDE by using the feature of generating an application that is already setup to be run. The benefit to this is that the developer can dive straight into the functionality of the application and not worry about making everything work together. The application that was generated uses .NET C# for the backend and React for the frontend. The backend and frontend are connected automatically by the IDE when generated and work perfectly. The backend connection to the database was made using the context class and Entity Framework.

Models

Implemented for this application we have the *Listing* model, *UserModel* model, *UserDto* model (DTO means data transfer object), *UserDtoRegister* model and *ListingDto* model. Models containing Dto in their name are used as data transfer objects, they are files used

by the controllers to transfer data from the frontend to the backend, or from the backend. Once the data from the Dto models has been processed it is stored in its respective model (one without Dto).

UserModel

The *UserModel* is a C# file that describes the user of the application. Each user has a first name, last name, email, username and password. The *UserModel* is used by the *AuthController* to save user data when registering and to verify the user data when logging in. The model is also used by the *LostAndFoundContext* which uses it to describe the user table in the database.

```
namespace LostAndFound.Models;

public class UserModel
{
    [Key]
    public string UserName { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string PasswordHash { get; set; }
}
```

Code snippet 1: UserModel

Controllers

The controllers used for this application are the *AuthController*, used for registration, login and getting information about the user and the *ListingController*, used to create or delete listings about lost or found items.

AuthController

The *AuthController* is used while the user registers, logs in or to get user information based on the JWT token that the user has stored in his browser. The *AuthController* is implemented as an ApiController which means that it extends the ControllerBase class instead of the Controller class. The main difference is that the ApiController does not render views but instead returns data in the xml or json format. In the code snippet 2 we can see that the *AuthController* is indeed an ApiController since it inherits the ControllerBase class. We can also see the `_context` variable which is used to manipulate the database.

```
namespace LostAndFound.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly LostAndFoundContext _context;

        private static UserModel _user = new UserModel();
        private readonly IConfiguration _configuration;
        public AuthController(IConfiguration configuration,
            LostAndFoundContext context)
        {
            _configuration = configuration;
            _context = context;
        }
        ...
    }
}
```

Code snippet 2: Part of the AuthController

ListingController

The *ListingController* is also implemented as an *ApiController*. The function it serves is to save the listings posted by the user or delete it. It also has a *HttpGet* function which returns all of the listings that are posted in the database. In the code snippet 3 we can see that the function to get all of the data is under the *Authorize* attribute, this means that only an authorized user can use this function. The authentication is done by the JWT token *Bearer* that is sent to the backend in the header of the request.

```
...
[HttpGet("getall")]
[Authorize]
public async Task<ActionResult<List<Listing>>> GetAllListings()
{
    var listings = await _context.Listings.ToListAsync();
    return Ok(listings);
}
...
```

Code snippet 3: HttpGet function from ListingController

Program.cs

Program.cs is a file written in C# that is used to add services to our application, build it and run it. This file sets up the dependencies, registers the components and sets up the configuration. In the file we can find a variable builder which is a variable used to define said functionalities, add services and ultimately run the application. In the code snippet 4 we can see a part of the *Program.cs* file where use the builder variable to add authentication using the JWT token. This is setup for development purposes to use a token set in the appsettings, in production it would use some third party tool for secret safekeeping.

```

...
builder.Services.AddAuthentication(options =>
{
    options.DefaultChallengeScheme = JwtBearerDefaults
        .AuthenticationScheme;
    options.DefaultAuthenticateScheme = JwtBearerDefaults
        .AuthenticationScheme;

}).AddJwtBearer(options =>
{
    options.RequireHttpsMetadata = false;
    options.SaveToken = true;
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8
            .GetBytes(builder.Configuration
                .GetSection("AppSettings:token").Value!)),
        ValidateIssuer = true,
        ValidIssuer = jwtSettings["ValidIssuer"],
        ValidateAudience = false,
        ValidAudience = jwtSettings["ValidAudiences"],
    };
});
...

```

Code snippet 4: Part of The Program.cs File

4.2 Frontend

The frontend application is developed using React.js, which is a popular JavaScript library that makes developing applications easier and more intuitive. The application is implemented as a SPA (Single Page Application) which means all of the routing is done on the frontend. This means that the initial page is loaded, and the content is dynamically changed based on user interaction with the application.

Components

The frontend is divided into components which are the building blocks of React. The components are exported and used by other components in a way you would use any Html element. This allows us to reuse the component as many times as needed, saving a lot of time and drastically lowering the chance of making an error. Components also make updating the application much easier. A good example is the *Registration* component which is used to display the registration form to the user, using which the user can register an account. In the code snippet 5 we can see an asynchronous function from the *Registration* controller which sends a http post request to the backend server and waits for the response. If the username is not already in the database, the backend will create a new user, store the information in the database and respond with the code 200 (OK).


```

async handleRegister(event) {
  event.preventDefault();
  const { userName, password , firstName, lastName, email} = this.state;

  const response = await fetch('https://localhost:7186/api/auth/register',
  {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body:JSON.stringify({userName, password, firstName, lastName, email})
  });

  const data = await response.json();
  if (response.ok) {
    this.setState({ message: 'Registration successful!' });
  } else{
    this.setState({ message: data || 'Registration failed' });
  }
}

```

Code snippet 5: Asynchronous function handleRegister from Register component

React Leaflet

Leaflet is an open-source JavaScript library for interactive maps. React Leaflet provides us with bindings between React and Leaflet. This means that it leverages Leaflet to abstract layers as React components. React does not render Leaflet layers to the DOM, this is done by Leaflet itself. React Leaflet library can be imported and used just as any other React library.

4.3 Authentication

It is always better to use third party solutions when it comes to authenticating users and encrypting and storing sensitive data. Developing these parts of the application on your

own lead to potential weaknesses and security problems. This is why this application uses the BCrypt library for hashing and encryption and JWT tokens for authentication.

JSON Web Token

JSON Web Token or JWT token for short, is an open standard that defines a way of securely transmitting information between parties as a JSON object. This information is trusted because it is digitally signed. JWTs can be signed using a secret or with a public/private key pair. In this application it is implemented using a secret with the HMAC algorithm. JWT consists of the header, payload and signature. In image 4.3 we can see what the JWT token structure looks like

Algorithm: HS256

Encoded: PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG91IiwiaWF0IjoxNTE2MzY1MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded: EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "sub": "1234567890",  "name": "John Doe",  "iat": 1516239022}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret  )  secret base64 encoded
```

Signature Verified

SHARE JWT

Image 4.3: jwt.io Website Debugger Screenshot

BCrypt

BCrypt in .NET is a cryptographic hashing library that provides secure hashing and password management capabilities. It is commonly used for hashing passwords before storing them in a database, ensuring that the passwords are stored in a secure, non-reversible manner. It tries to prevent off-line password cracking using a computationally-intensive hashing algorithm. The algorithm generates a random salt every time you use it so you

can has the same password multiple times and get different results.

code snippet 6 shows how we can use BCrypt to hash a password and code snippet 7 shows how we can use BCrypt to check if the password the user has provided to us, matches the previously hashed password.

```
string passwordHash = BCrypt.Net.BCrypt.HashPassword(request.Password);
```

Code snippet 6: Example of Using BCrypt to Hash a Password

```
if (!BCrypt.Net.BCrypt.Verify(request.Password, user.PasswordHash))  
{  
    return BadRequest("Log in failed!");  
}
```

Code snippet 7: Example of Using BCrypt to Check The Password

4.4 Directory Layout

The directories are arranged as in the image 4.4. The *ClientApp* directory is where, as the name suggests, all of the client application files can be found. In table 4.1 we can see what each of the folders, and some important files within them represent.

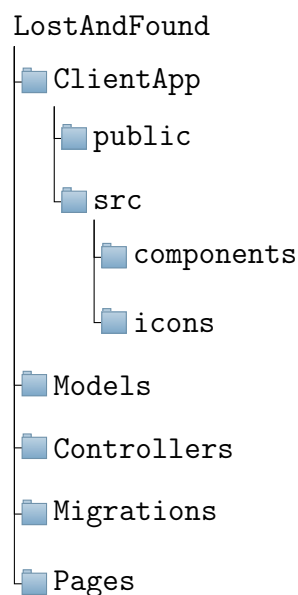


Image 4.4: This is a directory tree

Folder/File	Description
LostAndFound/	The root directory of the project
ClientApp/	The directory containing the client application
src/	Folder containing client application source files
components/	Folder containing files describing React components
Models/	Folder containing C# files describing models
Controllers/	Folder containing C# files describing controllers
Migrations/	Folder containing C# files describing migrations
Pages/	Folder containing CSHTML files describing Views
appsettings.json	File containing custom application configuration information
Program.cs	C# file that is the entrypoint of the application

Table 4.1: Folder Structure and Descriptions

4.5 Web Service

The application can be used from the browser through the frontend application, or it can be used as a web service (HTTP API). The backend is implemented as a web service meaning that we can use it by sending HTTP requests.

Register

```
POST /api/auth/register HTTP/1.1
Host: https://localhost:7186
Accept: application/json

{
  "username": "somename",
  "firstname": "some",
  "lastname": "name",
  "password": "password",
  "email": "somename@gmail.com"
}
```

Log in

```
POST /api/auth/login HTTP/1.1
Host: https://localhost:7186
Authorization: Bearer YOUR_ACCESS_TOKEN
Accept: application/json

{
  "username": "example",
  "password": "example"
}
```

Get user information

```
GET /api/auth/getuser HTTP/1.1
Host: https://localhost:7186
Authorization: Bearer YOUR_ACCESS_TOKEN
Accept: application/json
```

Get all listings

```
POST /api/listing/getall HTTP/1.1
Host: https://localhost:7186
Authorization: Bearer YOUR_ACCESS_TOKEN
Accept: application/json
```

Delete a listing

```
DELETE /api/listing/deleteListing/listingId HTTP/1.1
Host: https://localhost:7186
Authorization: Bearer YOUR_ACCESS_TOKEN
```

```
Accept: application/json
```

Make a listing

```
POST /api/listing/makepost HTTP/1.1
Host: https://localhost:7186
Authorization: Bearer YOUR_ACCESS_TOKEN
Accept: application/json

{
  "itemtype" : "keys",
  "message" : "I found these keys on the road",
  "latitude" : 45.5465,
  "longitude" : 75.2342,
  "address" : "Ilica 122"
}
```

Send a message

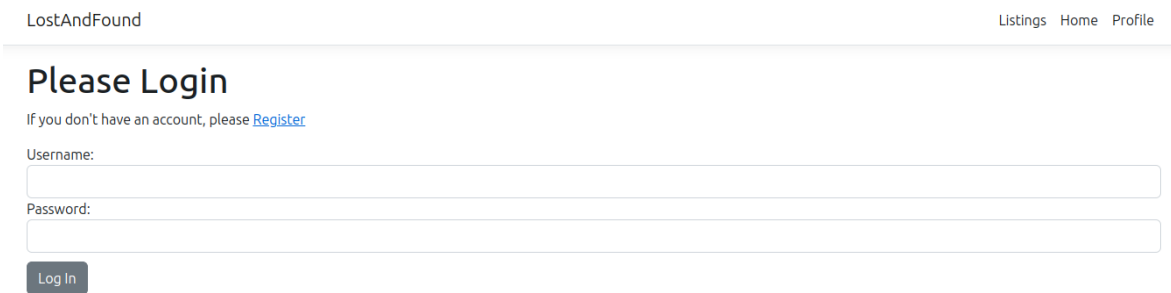
```
POST /api/messagess/sendMessage/listingId HTTP/1.1
Host: https://localhost:7186
Authorization: Bearer YOUR_ACCESS_TOKEN
Accept: application/json

{
  "message" : "some message"
}
```

5 Application Showcase

Log In / Register

If the current user is not logged in, when the user clicks on the *Profile* link in the top right corner, the website takes him to the log in screen(image 5.1). There the user can log in, or if the user does not have an account, they can click on the *Register* link which will take them to the registration screen(image 5.2).



The screenshot shows the login page for 'LostAndFound'. At the top left is the site name 'LostAndFound' and at the top right are navigation links 'Listings', 'Home', and 'Profile'. The main heading is 'Please Login'. Below it is a link: 'If you don't have an account, please [Register](#)'. There are two input fields: 'Username:' and 'Password:'. A 'Log In' button is located below the password field.

Image 5.1: The log in screen



The screenshot shows the registration page for 'LostAndFound'. At the top left is the site name 'LostAndFound' and at the top right are navigation links 'Listings', 'Home', and 'Profile'. The main heading is 'Please Register'. Below it are several input fields: 'Username:', 'First Name:', 'Last Name:', 'Email:', and 'Password:'. A 'Register' button is located below the password field.

Image 5.2: The registration screen

Search / Create Listings

Once logged in, the user can access the full functionality of the web application. By going to the Listings screen (image 5.3), the frontend loads all of the listings posted on the website and displays them for the user. The user can message other users with questions about their listings and search the listings by the type of item in the listing.

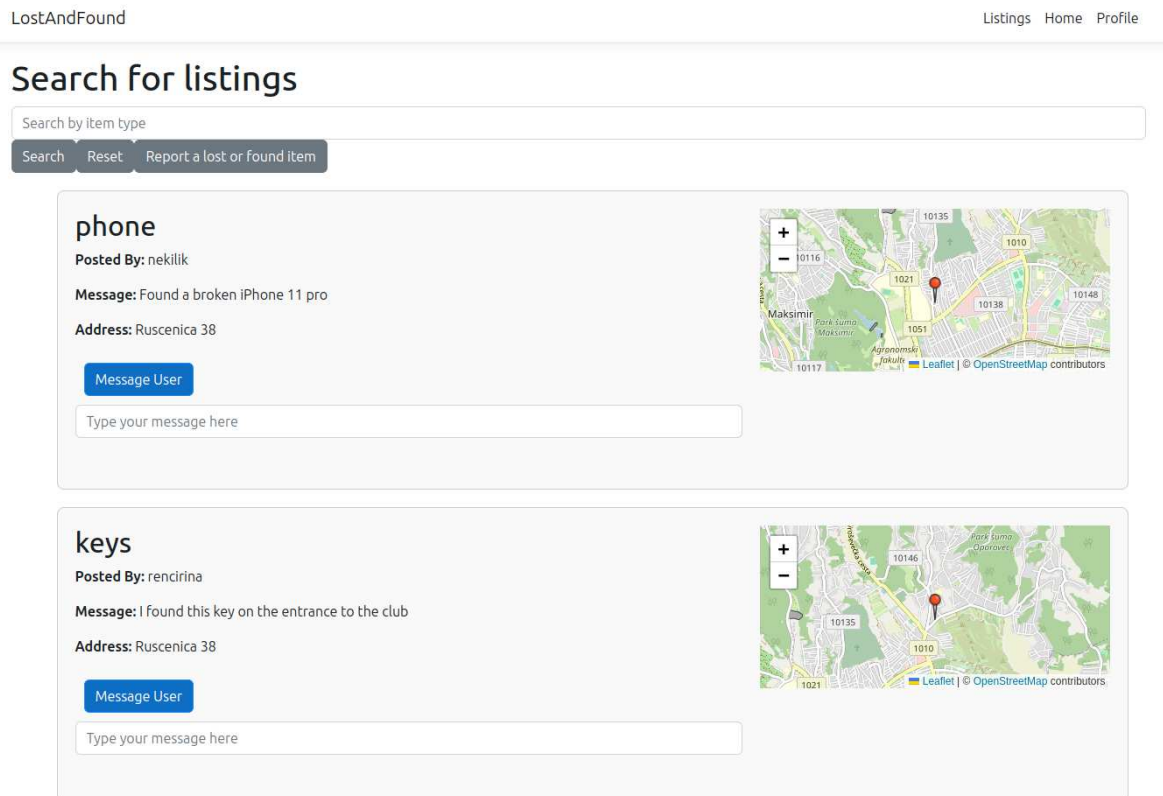


Image 5.3: The listings screen

To make a listing, the logged in user navigates to the listings screen (image 5.3) and clicks on the Report a found item button. This will open a form to fill in the information about the lost/found item. There will be a popup asking the user to allow the browser to use his current location, and the mini map will set the pin to the received location. The user can then correct the pin by dragging it on the map.

To search for listings using the item type the user simply writes the type of item he would like to filter by and press the search button. If there are no listings with the specified item type, there will be no change in the listings displayed. To reset the search criteria, the user presses the reset button.

LostAndFound Listings Home Profile

Search for listings

Search by item type

Search Reset

Create a Post

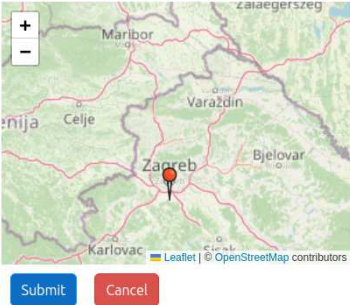
Message:

Item Type:

Address:

Upload photo

Choose File No file chosen



Submit Cancel

Image 5.4: Making a listing

Profile

The profile screen, once the user has logged in will display the user information and if there are any, his listings. The user can press the log out button to log out of the web application and this will send him back to the log in screen. User can delete the listings he has posted if he has posted any (image 5.5).

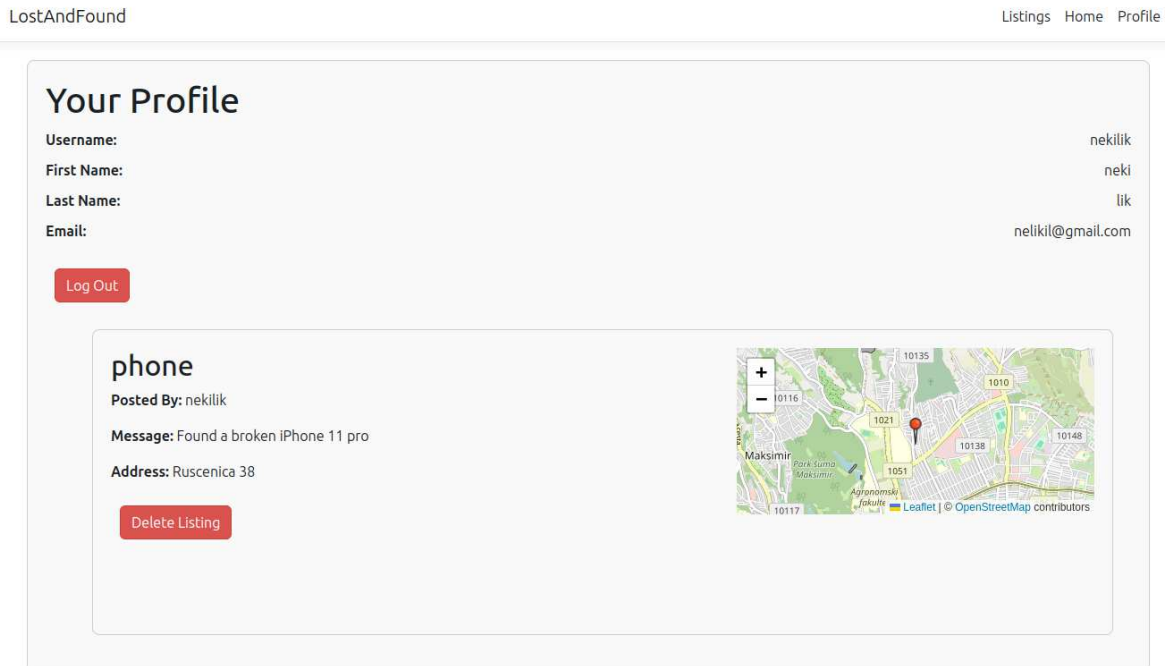


Image 5.5: The profile screen

6 Installation Manual

The application is located on <https://github.com/Mmacukic/LostAndFound/>. You need to download it locally and open it in an IDE of your choice that supports *ASP.NET Core* and *JavaScript* or open the directory in some sort of terminal or command prompt. The first thing you need to do is install dotnet on your personal computer. After installing dotnet you need to setup the application to work on your computer. First you need to configure a database management server either locally or via some third party service. The connection string is located in the *Program.cs* under the file The server is accessed by the application using a connection string, which is a set of information that you need to connect to the database server. After connecting to the server you need to add a migration using Entity Framework, either from the IDE or the terminal. In the terminal you need to navigate to the directory in which the solution is located and run the command "dotnet ef migrations add <name the migration> -project LostAndFound". This command will create a migration that prepares the data to be added to a database. The next command you need to run is "dotnet ef database update -project LostAndFound". This command will create the database using the previously added migration. The next step is to provide the JWT bearer function with a secret token. The token is used in the *Program.cs* file and in the *AuthController*. If you want to deploy the application you need is to have a secure third party application (Azure Vault or something similar) to keep your connection string and your secret token safe. To run the application you need to run the command "dotnet run".

7 Conclusion

In this thesis, we have successfully developed and documented a web application for lost and found items. The application is implemented using react.js on the client side, and ASP.NET Core that is implemented as a web service. The application provides the user with functionalities such as searching for listings, creating listings and messaging other users about listings.

The application has many options for possible upgrades. One of them being implementing a real-time chat that uses the SignalR package from .Net. This would provide the user with a way to communicate in real time while, for example, trying to find each other to retrieve or return the lost item. One other usefull upgrade would be to add a better form of authenticating users on registration, such as email and phone number confirmation, and providing a form of identification to prove that the given information is in fact true. The application presents a useful solution to an everlasting problem that is present in our day to day life.

8 Bibliography

- [1] *Umbraco*, <https://umbraco.com/knowledge-base/asp-dot-net-core/>, 10.06.2024.
- [2] *Entity Framework Tutorial*, <https://www.entityframeworktutorial.net/entityframework6/what-is-entityframework.aspx>, 11.06.2024.
- [3] *Dot Net Tutorials*, <https://dotnettutorials.net/lesson/controllers-asp-net-core-mvc/>, 22.05.2024.
- [4] *MDN Web Docs*, <https://developer.mozilla.org/en-US/docs/Glossary/SPA>, 19.05.2024.
- [5] *Microsoft Learn* <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/>, 05.06.2024.
- [6] *JWT Introduction* <https://jwt.io/introduction>, 01.06.2024.
- [7] *React Leaflet Introduction*, <https://react-leaflet.js.org/docs/start-introduction/>, 11.05.2024.

Abstract

Web application for Lost and Found Items

Marcel Macukić

The application developed in the scope of this thesis is designed for people who have found or have lost an item. The application implements making a post about items using the users current location. Users can message each other only through a post. Users that have posted can see their posts on their profile page and delete them. Users can also search through the posts using the search bar on top of the listings page. The search bar only allows for searching by item type and can be reset using the reset button.

Keywords: dotnet; .NET; React; C#; leaflet; MSSQL; JavaScript

Sažetak

Web-aplikacija za izgubljene i pronađene stvari

Marcel Macukić

Aplikacija razvijena u okviru ovog rada je dizajnirana za ljude koji su pronašli ili izgubili neku stvar. Aplikacija implementira izradu objave o izgubljenim ili nađenim stvarima koristeći trenutnu lokaciju korisnika. Korisnici mogu slati poruke jedni drugima samo preko objave. Korisnici koji su objavili svoje objave mogu vidjeti na svojoj profilnoj stranici i izbrisati ih. Korisnici također mogu pretraživati objave pomoću trake za pretraživanje na vrhu stranice listings. Traka za pretraživanje omogućuje samo pretraživanje prema vrsti stavke i može se poništiti pomoću gumba reset.

Ključne riječi: dotnet; .NET; React; C#; leaflet; MSSQL; JavaScript