BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE - ENGLISH

DIPLOMA THESIS

Attractions

Supervisor
Lect. dr. Pop Andreea Diana

Author
Pătrulescu Ronald Sandrino

2024

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ - ENGLEZĂ

LUCRARE DE LICENŢĂ

Atracții

Conducător științific
Lect. dr. Pop Andreea Diana

Absolvent
Pătrulescu Ronald Sandrino

2024

ABSTRACT

With the increasing popularity of social media and the growing interest in travel, there is a need for innovative features in travel-related applications. This bachelor thesis presents the design and implementation of a web application that provides a platform that allows visualisation of attractions and the ability to create your own attractions or lists of such attractions.

The first chapter

The second chapter

The third chapter

The fourth chapter

The fifth and final chapter

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Background . . . 2
1.2.1 The Internet . . . 2
1.2.2 The World Wide Web . . . 2
1.2.3 HTML . . . 3
1.2.4 CSS and JavaScript . . . 3
1.2.5 AJAX . . . 3
1.2.6 Emergence of Web Development Frameworks and Libraries . 4
1.2.7 Evolution of Database Technologies . . . 4
1.2.8 Advances in Testing and Security . . . 5
1.3 Objective . . . 6

2 Theoretical frame **7**
2.1 Backend . . . 7
2.1.1 lorem ipsmum . . . 7
2.2 Frontend . . . 7

3 Technical frame **8**

4 Application **9**
4.1 Analysis . . . 9
4.1.1 Functional requirements . . . 9
4.1.2 Use cases . . . 10
4.2 Architecture . . . 11
4.2.1 Class diagrams . . . 13
4.3 Implementation . . . 15
4.3.1 ORM Mapping . . . 15
4.3.2 Automapper . . . 16
4.3.3 Reading from and writing to the database . . . 17
4.3.4 URL Mapping . . . 19

4.3.5 Dependency Injection 19
4.3.6 Error handling 20

5 Conclusions and future work **22**
5.1 Conclusions 22
5.2 Future work 22
5.2.1 Internationalization (i18n) and localization (l10n) 22
5.2.2 Accessibility (A11y) 23
5.2.3 Responsive Design 23
5.2.4 Animations 23
5.2.5 Visibility for attractions 23
5.2.6 Shared collections 23
5.2.7 Personalized recommendations 24
5.2.8 Abstracting the usage of the the UI design library 24
5.2.9 Legal and informational pages 24

Bibliography **25**

Chapter 1

Introduction

This chapter serves as an opening to the thesis, providing an overview of the motivation, background and objective behind the development of a travel-focused media web application. While it builds on the abstract and preface by delving deeper into the subject, it also serves as a prelude to the technical and detailed discussions that will follow in subsequent chapters. The chapter is structured into several sections, each focusing on a particular aspect of the project.

1.1 Motivation

Travel, a cornerstone of shared experiences in the journey of life, serves as a dynamic conduit for the creation of enduring connections, precious memories, and deep personal insights. This potent medium invites us closer to each other, nature, and our inner selves, fostering a broader understanding of the world and our place within it. The exploration of diverse cultures and environments cultivates empathy and presents new perspectives, fulfilling the higher tiers of Maslow's hierarchy of needs such as self-actualization and transcendence. Living in a world driven fundamentally by relationships and the joy of experiencing the vitality of various cultures, travel becomes the nucleus of our most meaningful experiences. This basic human instinct to share individual journeys extends from the primal storytelling around a fire to the contemporary global narratives propelled by technology, underlining our collective yearning for shared experiences. Despite superficial differences, travel reveals profound commonalities between cultures, such as the instinct to survive, the pursuit of pleasure over pain, and our fundamental needs, thereby fostering empathy and reinforcing our interconnectedness.

1.2 Background

Numerous ground-breaking developments in computer science and technology have influenced the progress of web application development. In order to comprehend how the proposed travel-focused web application was developed, it is important to understand the key turning points that led to the current state of web apps. This section covers those turning points in detail.

1.2.1 The Internet

The invention of the internet fundamentally changed how people interact, obtain information, and conduct business in the modern day. The late 1960s and early 1970s saw the collaboration of numerous scholars and institutions to create the internet, often known as the ”network of networks.” The US Department of Defense’s Advanced Research Projects Agency (ARPA) first created it as a decentralized communication network called ARPANET. Establishing a strong and resilient network that could endure partial failures and continue operating in the event of a calamity or military attack was the main driving force behind its design. The internet grew over time, connecting colleges, research centers, and eventually people all over the world. [CK74]

1.2.2 The World Wide Web

The World Wide Web (WWW), also referred to as the web, has transformed how we access and interact with information and has become a crucial part of the internet. While employed by the European Organization for Nuclear Research (CERN) in the late 1980s, Sir Tim Berners-Lee created the World Wide Web. His goal was to develop a system that would let users browse and distribute materials via hypertext links. WorldWideWeb, the first web browser, and the first web server were released in 1990 by Berners-Lee. These developments created the framework for the internet as we know it today. Fundamental web technologies like HTML (Hypertext Markup Language), which is used to structure web pages, and HTTP (Hypertext Transfer Protocol), which is used to communicate between servers and clients, allowed the web to grow quickly and be widely used. It changed the internet into a networked platform for dynamic content, multimedia, e-commerce, social networking, and many other applications from a collection of static documents. [BLF99]

1.2.3 HTML

As the common markup language for developing web pages and web applications, HTML (Hypertext Markup Language) is a vital component of the World Wide Web. It gives content, including as text, photos, multimedia components, and hyperlinks, a structure for organization and layout. The structure and display of web publications are defined by tags in HTML, which gives developers the ability to specify headings, paragraphs, lists, tables, forms, and other features. HTML's extensive popularity and function as the foundation of the web are due in part to its ease of use and adaptability. [Mozb]

1.2.4 CSS and JavaScript

The layout and formatting of HTML documents are described using the style-sheet language known as CSS (Cascading Style Sheets). It gives site designers control over elements like layout, colors, fonts, and animations that affect how online pages look. The structure of the web page is separated from the design elements using CSS, which makes it simpler to change and maintain the styling over several pages. With CSS, web designers can make websites that are visually appealing and responsive to various screen sizes and devices. [Moza]

On the other hand, JavaScript is a flexible programming language that allows for dynamic behavior and interactivity on web sites. It enables programmers to handle events, create functionality, modify HTML components, and interact with servers. For producing interactive elements like form validation, sliders, carousels, and interactive maps, JavaScript is frequently utilized in web development. By providing real-time updates, client-side data processing, and seamless interaction with web apps, it improves the user experience. [Mozc]

1.2.5 AJAX

Asynchronous JavaScript and XML (AJAX) is a set of web development tools that permits data to be exchanged asynchronously between a web browser and a server without requiring a page reload. To construct dynamic and interactive online applications, it mixes JavaScript, XML (or other data formats like JSON), and asynchronous communication. Because AJAX enables seamless data retrieval and display without interfering with the user's browsing experience, web developers may utilize it to construct responsive and interactive user interfaces. Jesse James Garrett is credited with popularizing AJAX in his seminal piece "Ajax: A New Approach to Web Applications," which appeared on the website of Adaptive Path in 2005. The article explored how AJAX, which enables more dynamic and responsive web applications, has the potential to transform web development. [Gar]

1.2.6 Emergence of Web Development Frameworks and Libraries

Web development frameworks and libraries have emerged in recent years and have advanced quickly, altering how web applications are created. These frameworks and libraries give programmers a selection of instruments, pre-built frameworks, and reusable parts that speed up the programming process and increase productivity [SG14]. They make it possible for developers to concentrate more on application logic than on low-level technical details, which makes it easier to create web apps that are reliable, scalable, and packed with features.

The introduction of these frameworks and libraries has greatly streamlined web development by enabling programmers to use pre-existing code, adhere to best practices, and follow established patterns. Routing, data binding, component-based architecture, form validation, and API connectivity are just a few of the many functionalities they provide [Ste21]. These technologies help developers create web apps more quickly and easily because they abstract complicated activities and offer well-documented APIs.

The ability of web development frameworks and libraries to address issues like code organization, code reusability, performance optimization, and cross-browser compatibility is a major factor in their success. Additionally, they encourage cooperation and knowledge exchange among developers through vibrant online communities, in-depth documentation, and a robust ecosystem of plugins and extensions.

Over time, a number of significant frameworks and libraries—including Angular, React, Vue.js, Django, Ruby on Rails, and Laravel—emerged, each with unique advantages and traits. These frameworks have a big impact on the web development scene and have been widely embraced by developers. [Vue] [Dja] [Rub] [Lar]

1.2.7 Evolution of Database Technologies

Data management systems have become increasingly complex and effective as a result of tremendous evolution and innovation in the field of database technologies. This section examines the significant turning points in database technology development, highlighting noteworthy developments and their influence on the industry.

1. Hierarchical and Network Models

   Network and hierarchical modeling were popular in the early days of database systems. Data was represented using these models, which were respectively pioneered by IBM's Information Management System (IMS) and the CODASYL Data Model, in the form of trees or graphs [Dat03]. These models organized data well for the purposes of the applications they were designed for, but they lacked the adaptability to manage complicated relationships.

2. Relational Databases

   Edgar Codd changed the database industry in the 1970s by introducing the relational model [Cod70]. In relational databases, data was arranged into tables with rows and columns and relationships defined by keys. Relational database management systems (RDBMS) based on SQL, such as Oracle, MySQL, and PostgreSQL, were made possible by this model's ability to store and query structured data in a powerful and flexible manner.

3. Object-Oriented Databases

   As object-oriented programming languages became more popular, object-oriented databases (OODB) started to take the place of relational databases. By enabling the direct representation of objects in databases, OODBs sought to close the information storage and programming language gap [US90]. Complex data structures could be stored more easily as a result, and database models and application code were better integrated.

4. NoSQL Databases

   NoSQL (Not Only SQL) databases were created as a result of the development of web applications and the necessity to manage massive amounts of unstructured and semi-structured data [HELD11]. NoSQL databases introduced flexible data structures such key-value, document, columnar, and graph databases in place of the classic relational databases' inflexible schema. These databases provide high availability, scalability, and effective management of various data kinds.

5. NewSQL Databases

   The performance and scalability issues that traditional relational databases encountered in web-scale contexts led to the development of NewSQL databases [Asl11]. These databases sought to combine the scalability and flexibility of NoSQL databases with the advantages of relational databases, such as ACID compliance. Improved performance, distributed architectures, and horizontal scalability were all features of NewSQL solutions.

1.2.8 Advances in Testing and Security

The techniques of software development have been considerably changed by improvements in testing and security. Through the automation of repetitive operations, the reduction of errors, and the advancement of software quality, automated testing has completely changed the process [Ngu16]. In order to promote quicker feedback loops and effective release cycles, continuous integration and continuous deployment (CI/CD) approaches combine automated testing with seamless code integration and deployment [Hum10]. Through simulated attacks, penetration testing assists businesses in proactively identifying weaknesses [Eng13]. Security flaws and poor code quality can be found using static and dynamic code analysis techniques [Spi06]. Fuzz testing uses random or mutated inputs to stress-test software and uncover vulnerabilities [SGA07]. Threat modeling makes it possible to systematically identify and reduce potential security threats. [Sho14]. These developments aid in the creation of reliable and secure software systems, increasing resilience and dependability.

1.3 Objective

The main goal of this thesis is to create and put into practice a web application that encourages users to connect with other travelers, discuss their travel experiences, and explore new places.

Chapter 2

Theoretical frame

2.1 Backend

lorem ipsmum

2.1.1 lorem ipsmum

2.2 Frontend

Chapter 3

Technical frame

Chapter 4

Application

4.1 Analysis

4.1.1 Functional requirements

Throughout the application, the user can do the following:

- Unauthenticated user
  - Create an account
  - Login into the application
  - Reset their account's password
- Regular user
  - Log out of the application
  - Change between light and dark theme
  - Visualize, search, sort and filter attractions
  - Add a new attraction or edit a created one
  - React to an attraction with like or dislike
  - Save an attraction to a new or existing collection
  - Share an attraction to Threads, Twitter, Email or copy its link
  - View an attraction's details on its page
  - Comment on an attraction's page
  - View their own and other users' profile
  - Add or change their profile photo and description
  - Send, accept and decline friend requests and unfriend current friends

  - View their own and other users' friends
  - View their own sent or received friend request
  - View the list of their created attractions
  - View the list of their collections
  - Reorder their list of collections
  - Add a new collection or edit/delete a created one
  - View the details of a collection
  - Reorder attractions within a collection
  - Delete an attraction from the collection
  - Set the picture of an attraction as the collection's picture.

- Admin
  - Anything a regular user can do
  - View the list of attraction types and how many attractions are using them
  - Add or rename attraction types
  - Delete unused attraction types

4.1.2 Use cases

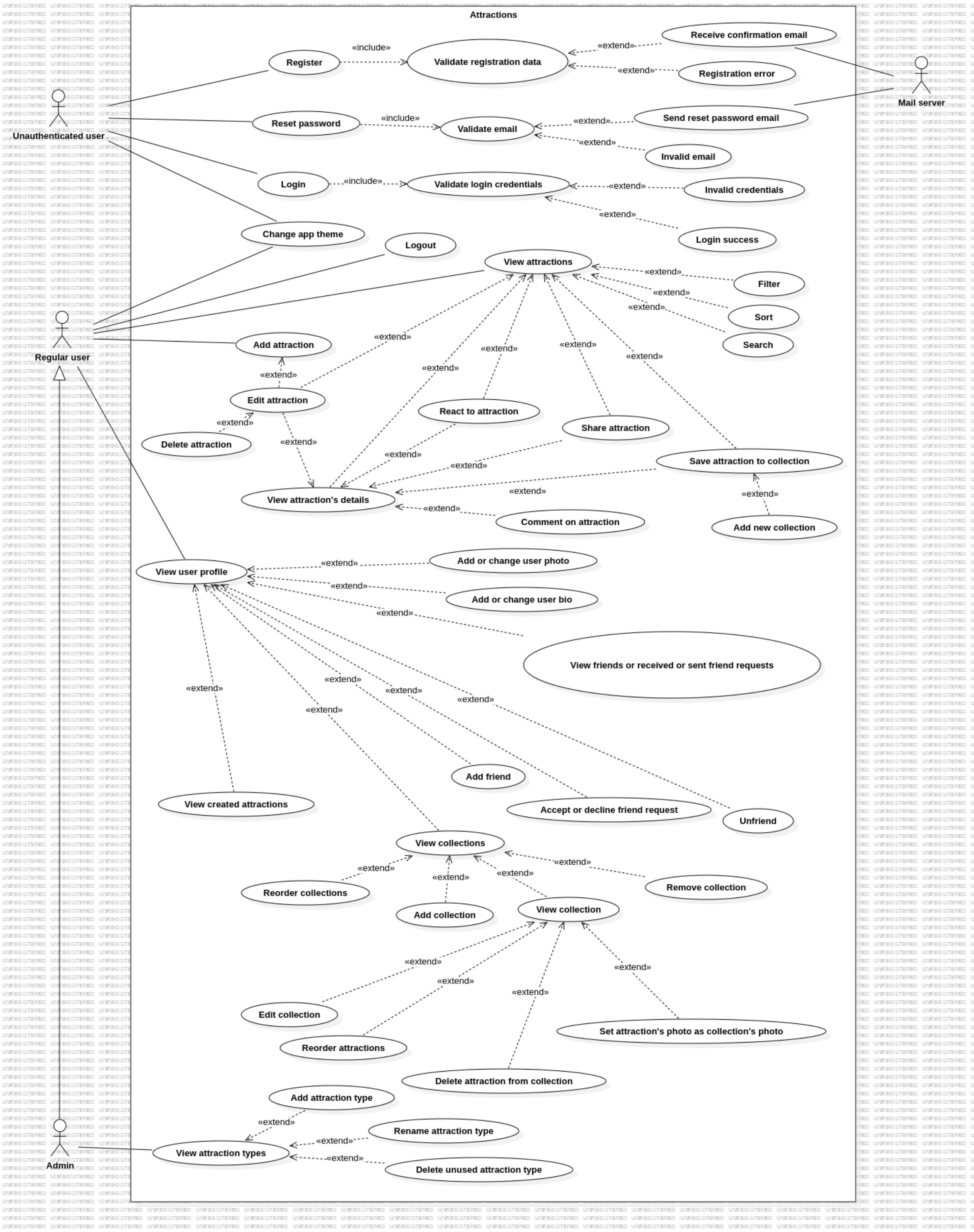

Figure 4.1: Use case diagram

4.2 Architecture

The application is structured on the client - server model. The client is a web application created using React and the server is a Web API created using the ASP.NET

Core subset of the .NET Framework. For persistent storage the database used is SQLite. External services are used for email sending and photo upload.

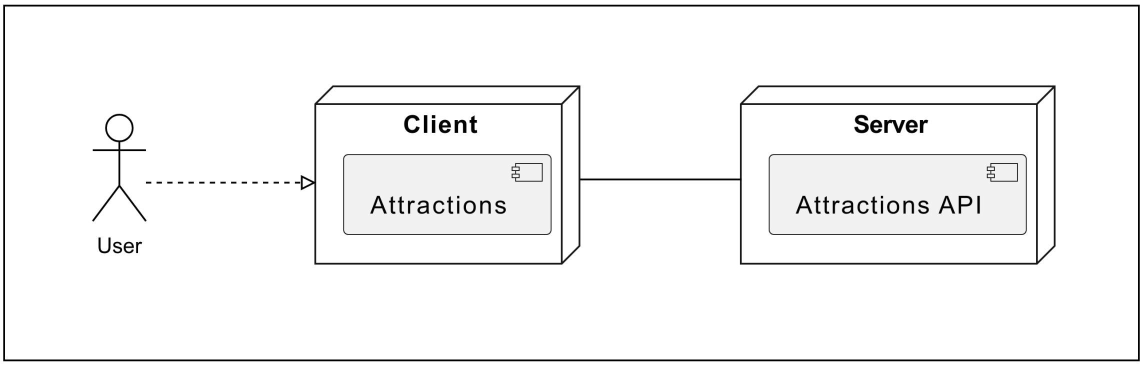

Figure 4.2: Application architecture

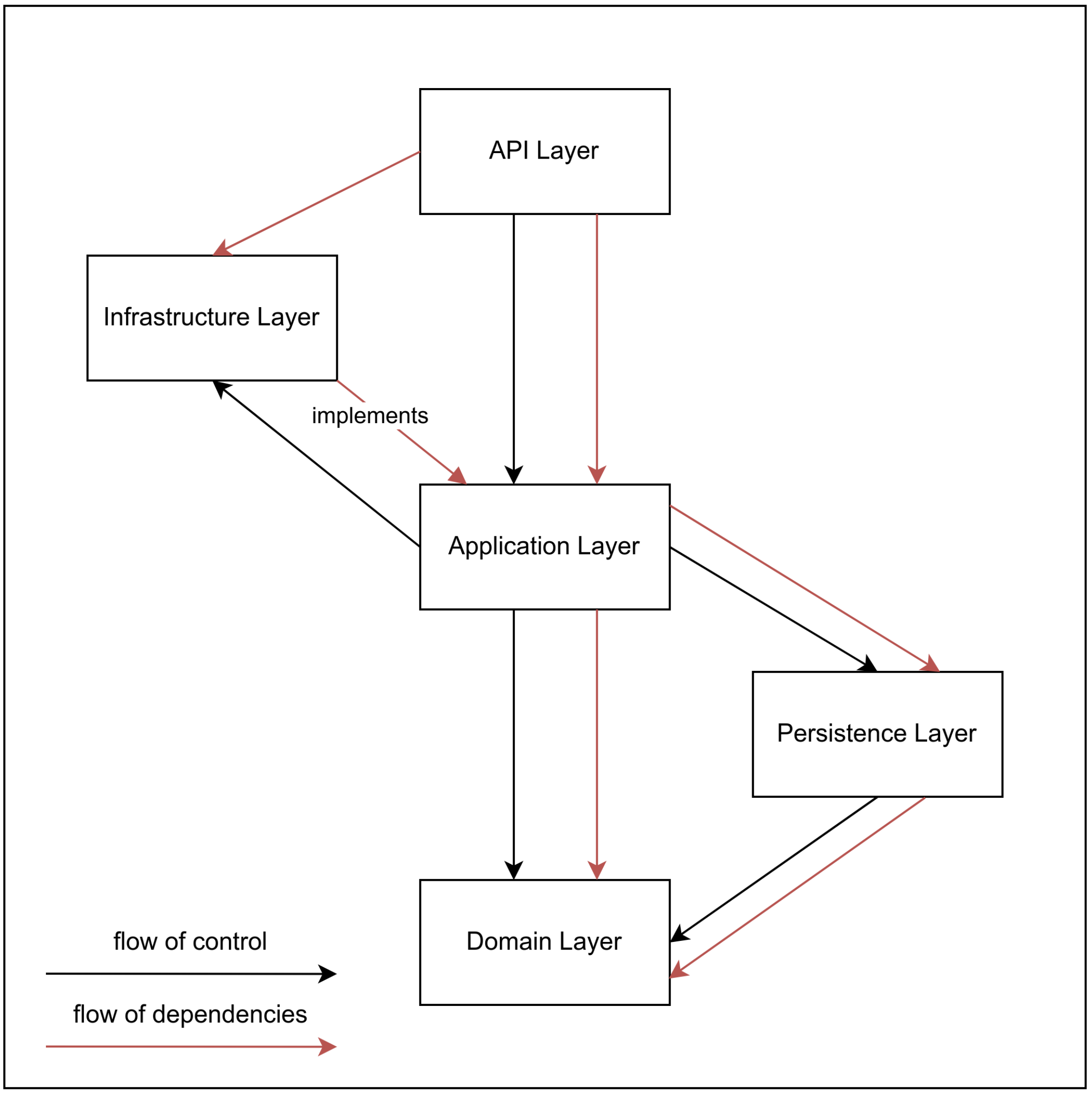

Figure 4.3: Backend layers

4.2.1 Class diagrams

Domain class diagram

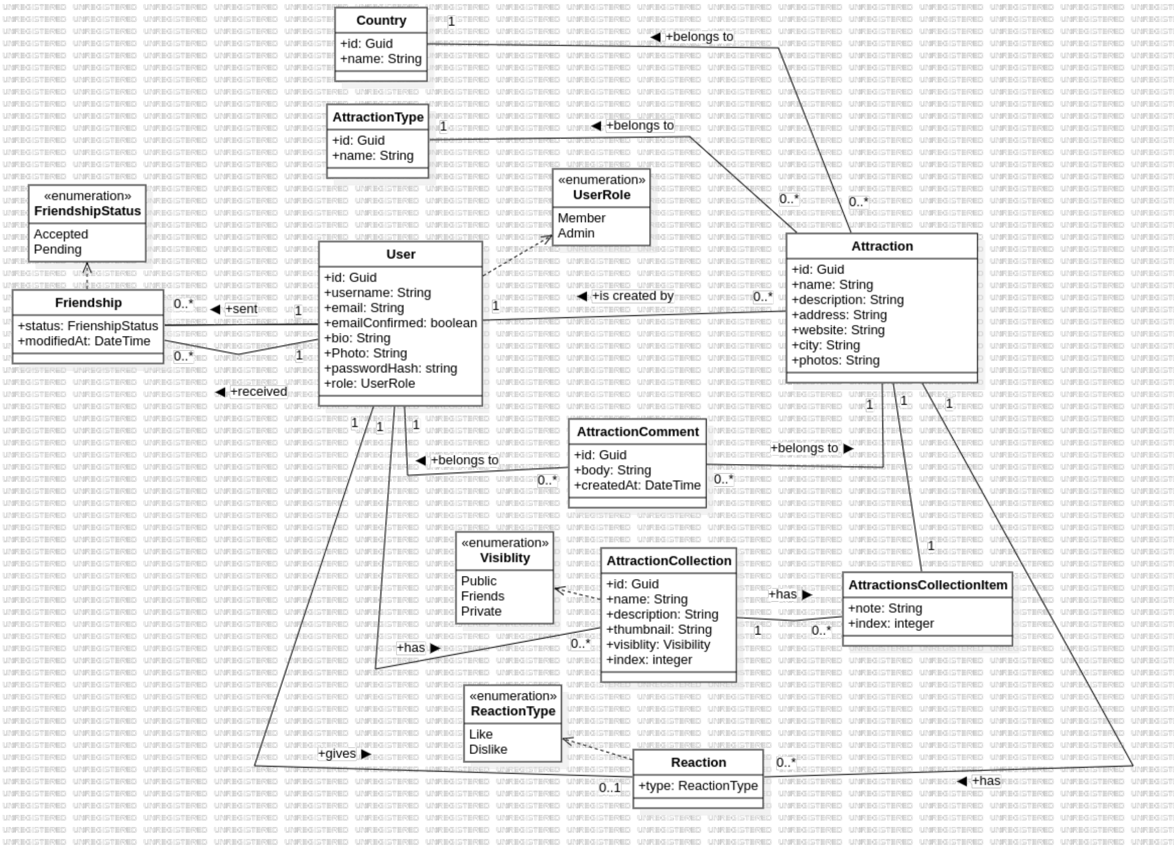

Figure 4.4: Domain class diagram (*associations should be aggregations)

Database diagram

Since the database is generated by the Entity Framework, its structure is similar to the domain

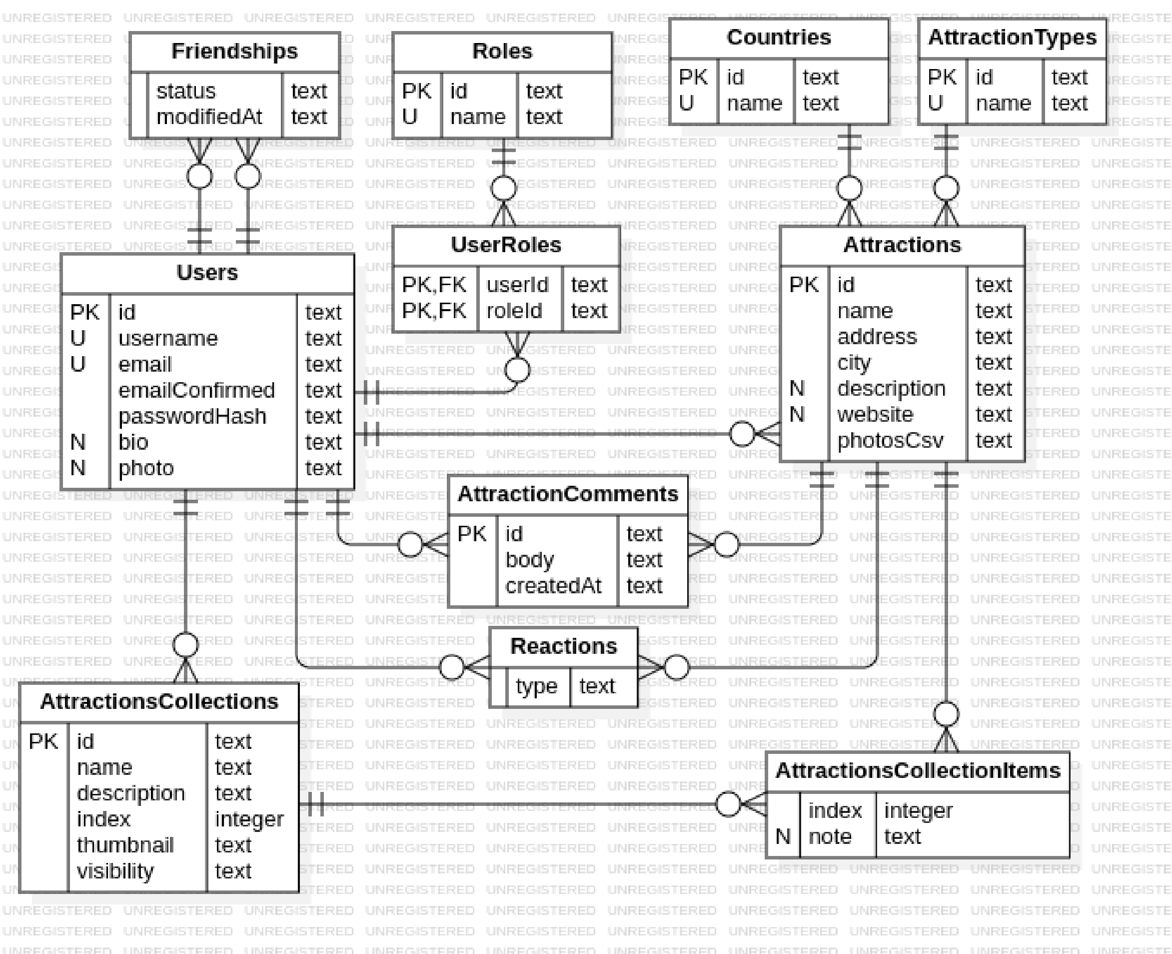

Figure 4.5: Database diagram

4.3 Implementation

4.3.1 ORM Mapping

Entity Framework was used to map the domain classes to the database tables. In order to achieve that, the domain classes have to be used as type parameters for fields of type DbSet inside a class that inherits from the framework's class DbContext. In this case the class is IdentityDbContext because it offers user management features and it is also a descendant of the DbContext class. This IdentityDbContext class takes as type parameters a class used for mapping the user, a class used for mapping the user role and the type of the primary key used for those two.

```
35 usages  Patrulescu-Ronald-Sandrino
public class DataContext(DbContextOptions options) : IdentityDbContext<User, UserRole, Guid>(options)
{
    7 usages
    public DbSet<AttractionType> AttractionTypes { get; init; }
    11 usages
    public DbSet<Attraction> Attractions { get; init; }
    5 usages
    public DbSet<Country> Countries { get; init; }
    6 usages
    public DbSet<Reaction> Reactions { get; init; }
    4 usages
    public DbSet<AttractionComment> AttractionComments { get; init; }
    9 usages
    public DbSet<AttractionsCollection> AttractionsCollections { get; init; }
    1 usage
    public DbSet<AttractionsCollectionItem> AttractionsCollectionsItems { get; init; }
    6 usages
    public DbSet<Friendship> Friendships { get; init; }

```

Figure 4.6: DataContext class' fields

The to be generated tables can be further configured by overriding the OnModelCreating method of the DataContext class. There you can define more complex primary keys, foreign keys, delete behaviors, unique indexes, check constraints, conversions, etc.

```
builder.Entity<AttractionsCollectionItem>(b :EntityTypeBuilder<AttractionsCollectionItem> =>
{
    b.HasOne(i :AttractionsCollectionItem => i.Collection) // ReferenceNavigationBuilder<AttractionsCollectionItem,...>
        .WithMany(c :AttractionsCollection => c.CollectionItems)
        .HasForeignKey(c :AttractionsCollectionItem => c.CollectionId)
        .OnDelete(DeleteBehavior.Cascade);

    b.HasOne(i :AttractionsCollectionItem => i.Attraction) // ReferenceNavigationBuilder<AttractionsCollectionItem,...>
        .WithMany()
        .HasForeignKey(i :AttractionsCollectionItem => i.AttractionId);

    b.HasKey(i :AttractionsCollectionItem => new { i.CollectionId, i.AttractionId });

    b.HasIndex(ci :AttractionsCollectionItem => new { ci.CollectionId, ci.AttractionId }).IsUnique();
    b.HasIndex(ci :AttractionsCollectionItem => new { ci.CollectionId, ci.AttractionId, ci.Index }).IsUnique();
});
}
```

Figure 4.7: Example of configuring the database tables generation by overriding the OnModelCreating method

Furthermore, the defined DataContext class has to be registered to the application services by using the AddDbContext method. This also registers it for dependency injection so it can be used inside the service classes. This is also where the database provider is set up, which requires a method for registering it (which is usually provided by the package used for the database provider), UseSqlite() in this case, and a connection string.

```
services.AddDbContext<DataContext>(options =>
{
    options.UseSqlite(configuration.GetConnectionString( name: "DefaultConnection"));
});
```

Figure 4.8: Registering the DataContext class

4.3.2 Automapper

Automapper is an object-object mapper. It is used in this app in order to do conversions between domain classes and DTO classes. Mappings can be configured using profiles, which are classes that inherit from Automapper's Profile class. The profile is then registered to the DI container of application services:

```
services.AddAutoMapper(typeof(MappingProfiles).Assembly);
```

Figure 4.9: Registering Automapper to the application's DI container

Mapping for fields with the same name happens implicitly. For the rest of them, we can define custom mappings or even ignore them, as seen in Figure 4.10, at lines 15 and 21 respectively.

```
    1 usage  Patrulescu-Ronald-Sandrino
8   public class MappingProfiles : Profile
9   {
        Patrulescu-Ronald-Sandrino
10      public MappingProfiles()
11      {
12          CreateMap<Attraction, Attraction>();
13          Guid? currentUserId = null;
14          CreateMap<Attraction, AttractionDto>()
15              .ForMember(d => d.Country, o => o.MapFrom(s => s.Country.Name))
16              .ForMember(d => d.AttractionType, o => o.MapFrom(s => s.AttractionType.Name))
17              .ForMember(d => d.Reaction,
18                  o => o.MapFrom(s =>
19                      s.Reactions.Where(r => r.UserId == currentUserId).Select(r => r.Type).FirstOrDefault()));
20          CreateMap<AttractionAddOrEditDto, Attraction>()
21              .ForMember(d => d.Photos, o => o.Ignore());
22          CreateMap<AttractionDto, Attraction>();
23          CreateMap<AttractionType, AttractionType>();
24          CreateMap<AttractionComment, CommentDto>()
25              .ForMember(d => d.AuthorUsername, o => o.MapFrom(s => s.Author.UserName))
26              .ForMember(d => d.AuthorPhoto, o => o.MapFrom(s => s.Author.Photo));
```

Figure 4.10: Example of Automapper profile configuration

```
110         var comment = new AttractionComment
111         {
112             Body = body,
113             Author = user,
114             Attraction = attraction,
115         };
116
117         attraction.Comments.Add(comment);
118
119         var success :bool = await context.SaveChangesAsync() > 0;
120         if (!success) throw new Exception("Failed to add comment");
121
122         return mapper.Map<CommentDto>(comment);
```

Figure 4.11: Example of Automapper usage (line 122)

4.3.3 Reading from and writing to the database

Reading is done by directly accesing the DbSet fields of the DataContext class. For example, in Figure 4.12 at line 58 the contents of table of attraction types are accesed by the AttractionTypes field on the variable context, which is an instance of DataContext injected into the service. The contents can be further processed on the database side by using non-terminal operations, methods that return an instance of IQueryable, like Where, Include (which does join) and ProjectTo. Then, when awaiting the call to ToListAsync the processed query is peformed on the database.

```
private async Task<List<AttractionTypeDto>> Get(Guid? id)
{
    var attractionTypes = await context.AttractionTypes.Where(at => !id.HasValue || at.Id == id)
        .Include(at => at.Attractions) // IIncludableQueryable<AttractionType,List<...>>
        .ProjectTo<AttractionTypeDto>(mapper.ConfigurationProvider).ToListAsync(); // Task<List<...>>
    return attractionTypes;
}
```

Figure 4.12: Reading from the database

Writing to the database is done in 2 steps: 1. calling Add/Remove (and derivatives like AddAsync or RemoveRange) on either the DbSet field or the DataContext instance, or by directly updating a domain class instance managed by the DataContext and which comes from a read operation and 2. by calling SaveChanges on the DataContext instances. An add example can be seen in Figure 4.11 and an update example can be seen in Figure 4.13

```
    public async Task<AttractionTypeDto> Update(AttractionTypeDto attractionTypeDto)
    {
        var attractionType = await FindInner(attractionTypeDto.Id);
        attractionType.Name = attractionTypeDto.Name;
        await context.SaveChangesAsync();
        return mapper.Map<AttractionTypeDto>(attractionType);
    }

    0+1 usages  Patrulescu-Ronald-Sandrino
    public async Task<AttractionTypeDto> Delete(Guid id){...}

    2 usages  Patrulescu-Ronald-Sandrino
    private async Task<List<AttractionTypeDto>> Get(Guid? id){...}

    2 usages  Patrulescu-Ronald-Sandrino
    private async Task<AttractionType> FindInner(Guid id)
    {
        return await context.AttractionTypes // DbSet<AttractionType>
            .Where(at => at.Id == id) // IQueryable<AttractionType>
            .Include(at => at.Attractions) // IIncludableQueryable<AttractionT
            .FirstOrDefaultAsync() ?? throw new NotFoundException();
    }
}
```

Figure 4.13: Example of database update operation

4.3.4 URL Mapping

Each URL path is prefixed by the name of the controller. This is achieved by having all the controllers inherit from BaseApiController, which defines the path prefix using the Route annotation (Figure 4.14). The rest of the URL path is defined by the argument of the annotations used for declaring the HTTP methods used (Figure 4.15). It can be observed that the paths can be parameterized.

```
namespace API.Controllers;

[ApiController]
[Route( template: $"{RoutePrefix}/[controller]")]
[Produces( contentType: MediaTypeNames.Application.Json)]
10 usages   10 inheritors   Patrulescu-Ronald-Sandrino
public class BaseApiController : ControllerBase
{
    protected const string RoutePrefix = "api";
}

```

Figure 4.14: How the name of the controller is added to the URL

```
    [HttpGet( template: "form-data/{id:guid?}")]
    2 usages   Patrulescu-Ronald-Sandrino
    public async Task<ActionResult<AttractionFormData>> GetAttractionFormData(Guid? id){...}


    [HttpPut( template: "{id:guid}/react")]
    Patrulescu-Ronald-Sandrino
    public async Task<ActionResult> React(Guid id, ReactionType reactionType){...}

    [HttpGet( template: "{username}")]
    Patrulescu-Ronald-Sandrino
    public async Task<List<AttractionDto>> GetCreatedAttractions(string username, [FromQuery] int pageNumber){...}
}

```

Figure 4.15: Defining endpoints

4.3.5 Dependency Injection

Dependency injection is achieved by specifying the needed classes as constructor parameters (4.16) and then registering those classes when the application is created (4.17). Here, they are registered using the AddScoped method, which registers their lifetime per HTTP request.

```
public class AttractionsService(
    DataContext context,
    IMapper mapper,
    AuthUtil authUtil,
    IPhotoAccessor photoAccessor)
    : IAttractionsService
{
```

Figure 4.16: Injection of dependencies using the primary constructor

```
        services.AddScoped<IAttractionTypesService, AttractionTypesService>();
        services.AddScoped<IAttractionsService, AttractionsService>();
        services.AddScoped<ICountryService, CountryService>();
        services.AddScoped<IUserService, UserService>();
        services.AddScoped<IAttractionsCollectionsService, AttractionsCollectionsService>();
        services.AddScoped<IUserAccessor, UserAccessor>();
        services.AddScoped<IFriendshipService, FriendshipService>();
        services.AddScoped<IAdminService, AdminService>();
        services.AddScoped<IPhotoAccessor, PhotoAccessor>();
        services.AddScoped<IEmailSender, EmailSender>();
        services.AddScoped<AuthUtil>();
```

Figure 4.17: Registration of dependencies

4.3.6 Error handling

When the Web API is created (in the entry point of the application - the Program.cs file) there are 2 things that are done. The first one is the registration of the services (Figure 4.18 lines 18 - 28) and the second is the configuration of the HTTP request pipeline (Figure 4.18 lines 33 - ...). The order in which the middlewares are configured is the order in which they are run. And, the reason the exception handling middleware is the first in the pipeline (Figure 4.18 line 33) is to catch any exception that occurs later in the pipeline. The purpose of the exception middleware is to have a single piece of code that handles what response is given, depending on the exception, to the initiator of the HTTP request. In this application, it changes the status code depending on the exception (Figure 4.19 lines 24 - 33), writes the validation errors to the body of the response (Figure 4.19) line 33) and adds the stacktrace to the response only if the app is running in development mode.

```csharp
var builder = WebApplication.CreateBuilder(args);
var configuration = builder.Configuration;

// Add services to the container.

builder.Services.AddControllers( configure: options =>
    {...})
    .AddJsonOptions(options => options.JsonSerialize
builder.Services.AddApplicationServices(configuratio
builder.Services.AddIdentityServices(configuration,

var app :WebApplication = builder.Build();

// Configure the HTTP request pipeline.
app.UseMiddleware<ExceptionMiddleware>();
```

Figure 4.18: Registration of the exception handling middleware

```csharp
public class ExceptionMiddleware(RequestDelegate next, ILogger<ExceptionMiddleware> logger, IHostEnvironment env)
{
    private static readonly JsonSerializerOptions JsonSerializerOptions =
        new() { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };

    Patrulescu-Ronald-Sandrino
    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await next(context);
        }
        catch (Exception e)
        {
            context.Response.ContentType = MediaTypeNames.Application.Json;
            var response = new ProblemDetails { Title = e.Message };

            switch (e)
            {
                case ForbiddenException:
                    response.Status = StatusCodes.Status403Forbidden;
                    break;
                case NotFoundException:
                    response.Status = StatusCodes.Status404NotFound;
                    break;
                case ValidationException ve:
                    response.Status = StatusCodes.Status422UnprocessableEntity;
                    if (ve.Errors.Count > 0) response.Extensions.Add("errors", ve.Errors);
                    break;
                default:
                {
```

Figure 4.19: Middleware for exception handling

Chapter 5

Conclusions and future work

5.1 Conclusions

ASP.NET Core offers a reliable and easy choice for Web API application development. To further solidify this, the minimal API that can be created is a 4-lines long code (Figure 5.1). And, by adding the Entity Framework into the mix to take care of the database side, all that remains is the business logic.

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

Figure 5.1: Minimal API with ASP.NET Core [Mic]

Since React is an unopinionated library, it leaves room for a high degree of customization. But that is not without cost, since it could lead to having to reinvent the wheel and/or ending up with tons of dependencies. It's like the space-time trade-off, the best choice depends on the individuality of each project.

5.2 Future work

5.2.1 Internationalization (i18n) and localization (l10n)

Internationalization (i18n) is the provisioning of the application with different languages. Internationalizing the application can expand its the reach to a global audience. Besides, since attractions could be all over the world, having the ability to switch between languages creates an easy way for comparison, which is handy for language learners.

Localization is the adaptation of the application to specific locales. Specifically, referring to date and time formats; number formats; symbols, icons, and colors; Right-to-Left (RTL) Language Support. This too increases the regional reach of the application.

5.2.2 Accessibility (A11y)

Accessibility means making the web application usable by as many people as possible, including those with disabilities. This increases the potential user base of the application.

5.2.3 Responsive Design

This means making the app usable on various devices, like phones, tablets and even smart TVs. Currently the app is the design for PC/laptop screen sizes only.

5.2.4 Animations

Adding animations to the application will improve the smoothness of the application and the user experience.

5.2.5 Visibility for attractions

Currently, only attractions collections allow setting the visibility. By extending it to attractions, it can improve the user experience by increasing the user's control of what parts of their information is publicly available. And, you can even extend the app's scope beyond the standard meaning of attractions. For example you could add your secret favorite place, even if it's not what you would normally call an attraction.

5.2.6 Shared collections

Are you planning a multi-destination trip with your friends/family and want to put all the suggestions or even the final itinerary in one shared place? That's where shared collections would come in.

5.2.7 Personalized recommendations

All the user interactions produce data: what attractions the user interacts with and how much time they spend on those attractions? what about their friends? is there a pattern to all the collections they made or to the attractions they created? etc. With collected data on such questions, an algorithm could be made that would give personalized recommendations to the users.

5.2.8 Abstracting the usage of the the UI design library

Unlike the other ideas, which have in mind the user experience (UX), this is about the developer experience. The UI design library in this application (Material UI) is used as is, by directly using the components provided. But those components could be abstracted into other components, put together in one place, that only expose functionality and no implementation details of the design library. This way, should a need for the change of the UI design library appear, the required work to achieve that would be way more tedious.

5.2.9 Legal and informational pages

Pages like Terms and Conditions, Privacy Policy, and Contact are crucial for regulatory compliance and for for establishing trust and transparency with users.

Bibliography

[Asl11] M. Aslett. The newsql movement. *451 Research*, 1(14), 2011.

[BLF99] T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its inventor*. Harper, USA, 1st edition, 1999.

[CK74] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 20(3), 1974.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 1970.

[Dat03] C. J. Date. *An Introduction to Database Systems*. Pearson, USA, 8th edition, 2003.

[Dja] Django. Django documentation. `https://docs.djangoproject.com/en/4.2/`. Online; accessed 15 March 2023.

[Eng13] P. Engebretson. *The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy*. Syngress, USA, 2nd edition, 2013.

[Gar] Garrett, J.J. Ajax: A New Approach to Web Applications. Adaptive Path. `https://web.archive.org/web/20180823223757/https://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/`. Online; accessed 15 March 2023.

[HELD11] Jing Han, Haihong E, Guan Le, and Jian Du. Survey on nosql database. In *2011 6th International Conference on Pervasive Computing and Applications*, pages 363–366, 2011.

[Hum10] D. Humble, J. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, USA, 1st edition, 2010.

[Lar] Laravel. Laravel Documentation. `https://laravel.com/docs/10.x`. Online; accessed 15 March 2023.

[Mic] Microsoft. Create a minimal API with ASP.NET Core. `https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api#examine-the-code`. Online; accessed 24 aug 2024.

[Moza] Mozilla Developer Network. CSS: Cascading Style Sheets. `https://developer.mozilla.org/en-US/docs/Web/CSS`. Online; accessed 15 March 2023.

[Mozb] Mozilla Developer Network. HTML: HyperText Markup Language. `https://developer.mozilla.org/en-US/docs/Web/HTML`. Online; accessed 15 March 2023.

[Mozc] Mozilla Developer Network. JavaScript. `https://developer.mozilla.org/en-US/docs/Web/JavaScript`. Online; accessed 15 March 2023.

[Ngu16] T. A. Nguyen. *Automated software testing*. USA, 1st edition, 2016.

[Rub] Ruby on Rails. Ruby on Rails Guides. `https://guides.rubyonrails.org/`. Online; accessed 15 March 2023.

[SG14] Shyam Seshadri and Brad Green. *AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps*. O'Reilly Media, USA, 1st edition, 2014.

[SGA07] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional, paperback edition, 2007.

[Sho14] Adam Shostack. *Threat Modeling: Designing for Security*. Wiley, kindle edition, 2014.

[Spi06] D. Spinellis. *Code Quality: The Open Source Perspective*. Addison-Wesley Professional, USA, 1st edition, 2006.

[Ste21] Stoyan Stefanov. *React: Up Running*. O'Reilly Media, USA, 2nd edition, 2021.

[US90] Rainer Unland and Gunter Schlageter. Object-oriented database systems: Concepts and perspectives. In *IBM Symposium: Database Systems of the 90s*, Advances in Database Systems, pages 154–197. Springer Berlin Heidelberg, 1990.

[Vue] Vue.js. The Progressive JavaScript Framework. https://vuejs.org/. Online; accessed 15 March 2023.