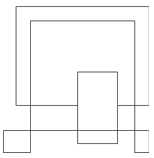


Travel advisor

Semester project 3

Lucija Domljan 325591

Zsolt Nóvé 326345

Joan Tammo 325753

Ameya Mahankal 326157

Supervisors:

Troels Mortensen

Jørn Martin Hajek

[Software Technology Engineering]

[Semester III.]

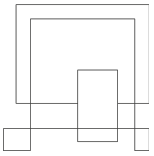
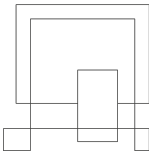
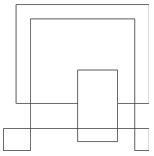


Table of content

Abstract 4
1 Introduction 1
Analysis 2
1.1 Requirements 3
1.2 Functional Requirements 3
1.3 Non-Functional Requirements 4
1.4 Use Case Diagram 5
Use Case Description 6
1.6 System Sequence Diagram 10
1.7 Domain Model 11
Security 12
2 Design 17
Architecture Pattern 17
Class Diagram 21
Technologies 22
SOLID 22
Interaction Diagrams 24
Data Models 25
Global Relation Diagram 27
Relation schema 28
UI design choices 30
Security 33
3 Implementation 34
4 Test 45
BloomRPC 45
Junit 46
Nunit 47
4.1 Test Specifications 48
5 Results and Discussion 50
6 Conclusions 52
7 Project future 54



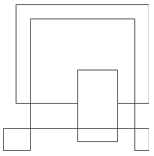
8 Sources of information 55
9 Appendices 1
List of figures and tables **3**



Abstract

The task at hand was to create a thorough travel advisory system that would give consumers a platform to explore the many activities offered in various cities. The system also had a comments option that let users express their thoughts and experiences. The project used a variety of computer languages and technologies, including Java, SQLite, C# (.NET), Blazor, and gRPC, to accomplish this.

Unified Process (UP) concepts and specific Scrum methodologies were blended into the project management approach to create a special process that was adapted to the project's requirements.



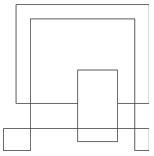
1 Introduction

After COVID-19 settled, people started to travel again, which is expected to result in a 30% increase in international tourist arrivals in 2023, up from a gain of 60% in 2022. These figures are anticipated to stay below pre-pandemic levels, though. [Tourism Outlook 2023 report]

According to a survey on travel trends, consumers are choosing their discretionary spending more wisely as 46% of travelers aim to take the same amount of vacations in 2023 as they did in 2022, while 41% plan to travel more next year. [Skyscanner's 2023] Travel agencies have restarted operations at pre-pandemic levels to fulfill the growing demand. Travelers could encounter problems including biased or erroneous information, malicious or fraudulent reviews by unidentified users and sponsored locations that don't go through rigorous background checks. This is because profit margins are the fundamental driving force behind travel agents.[Tegan S. report, 2015].

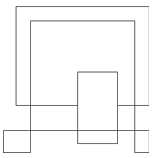
Travel advisor helps clients to find wanted destinations, provide expert advice and recommendations. It is a useful tool for tourists seeking professional guidance when traveling. The experts give clients individualized advice and services to make sure they have a pleasant and safe trip.

The website's major objective is to give visitors useful information to aid in trip planning so they may have a positive experience.The travel advisor gives clients comprehensive information about the places they are visiting, including top attractions, regional traditions, and safety precautions.



Analysis

In order to control the complexity of the system development and provide a clear view, several artifacts were created, including requirements, Use Case diagrams with descriptions, Activity diagrams, System Sequence diagrams, and a Domain model. The first step involved compiling a list of functional requirements, carefully considering needs and requirements of the target audience.



1.1 Requirements

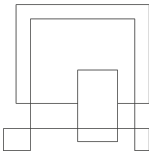
The project description's problem statement served as the foundation for determining the requirements. Group prioritized the requirements, which is reflected in the order that is shown below. Functional requirements define what the system must be able to achieve. Respectively, functional requirements were listed first and afterwards non-functional requirements.

1.2 Functional Requirements

Utilizing techniques like use cases, use case descriptions, and actor descriptions, functional requirements are specified. These approaches offer a methodical way to define the functionality and behavior that the system is supposed to have.

Critical priority:

As a user I want to be able to log in to the system.
As a user I want to be registered.
As an admin I want to be registered.
As a Guest I want to be able to browse the website.
As a user I want to be able to see a (display of information about) a city I searched for.
As an admin I want to be able to upload pictures to the posts.



High priority:

As a user I want to be able to add one comment.

As a user I want to be able to delete my comments.

As a user I want to be able to have a wishlist/save visited city.

Low priority:

As an admin I want to remove comments that are problematic.

As an admin I want to edit text(in a description).

As a user I want to report comments.

As a user I want to send messages to an Admin.

As an admin I can choose to respond to messages from users.

As an admin I want to close comments.

As an admin I want to be able to approve/remove reported comments(The admin will be able to remove the comment from the reported comments database).

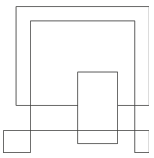
As a user I want to be able to add star reviews with my comments.

1.3 Non-Functional Requirements

System must be heterogeneous (C# and Java).

System must be a distributed system.

System must use two different network technologies(REST and gRPC).



1.4 Use Case Diagram

The diagram below was made based on our above-mentioned requirements. The diagram outlines the several actors, their roles, and how they relate to the system.
The user can access various pages on which they have permission to do so while also writing reviews and reporting reviews.

The Guest cannot post a comment; they can just research various pages. The admin controls and moderates the content that is present on the various pages. Every use case has its own description.

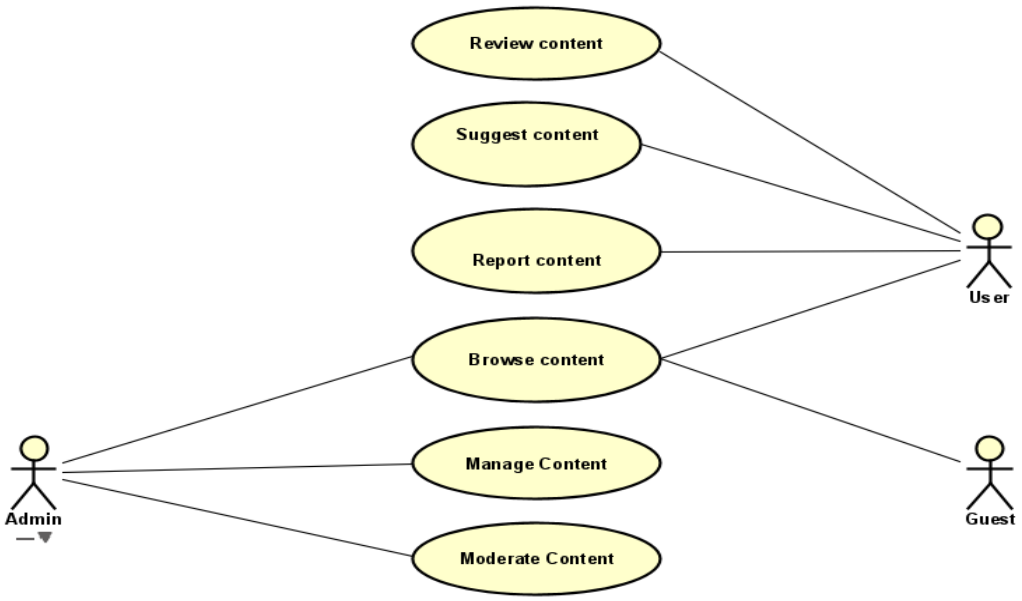
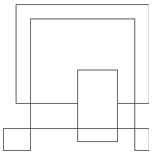


Figure 1. Use case diagram

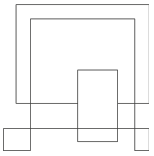


Use Case Description

Use case description	
Use case name	**Review content**
Summary	User is able to review the content of the website
Actor	User
Scope	Website
Preconditions	User is logged in. Users successfully found the post they want to review.
Success Guarantee	The review is displayed below the reviewed post.
Main Success Scenario	1.User writes a comment on a comment section. 2.User can leave star reviews on a post. 3.User submits their review.
Extensions	3.b.If the comment by the user already exists on the post, the next comment is denied.
Frequency of Occurrence	Once per post (restaurant,hotel,city)

Figure 2. Use case description table

The interaction between a developed system and its users is accurately captured in the use case description. It includes possibilities, outcomes, and thorough step-by-step instructions for both the primary course of action and unusual circumstances. Here is one example of a use case description that includes the base scenario for most users.



Every use case also comes with an activity diagram that shows how the system's activities flow.

1.5 Activity Diagram

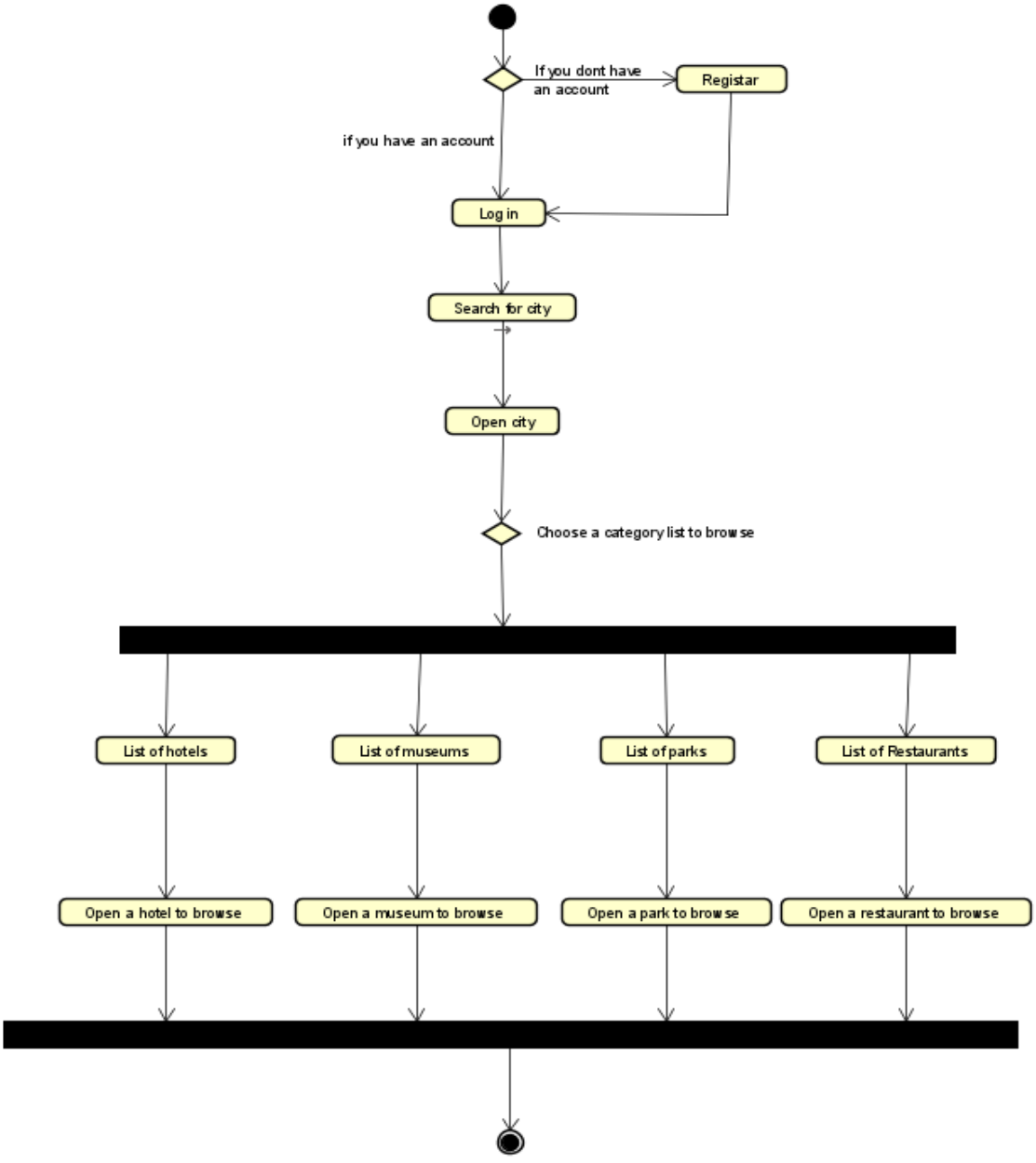
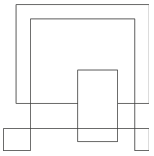
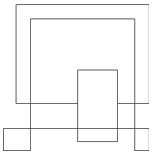


Figure 3. Activity diagram



This browsing activity diagram shows how a system or process's actions flow. It is a visualization of a system's decision-making processes and activities. A user can search for a desired city after signing in or registering. The user opens the city's page for more information once they have located the chosen city in the search results. The user chooses a category of interest from the city's page, such as the restaurants, hotels, museums, or parks they want to visit.



1.6 System Sequence Diagram

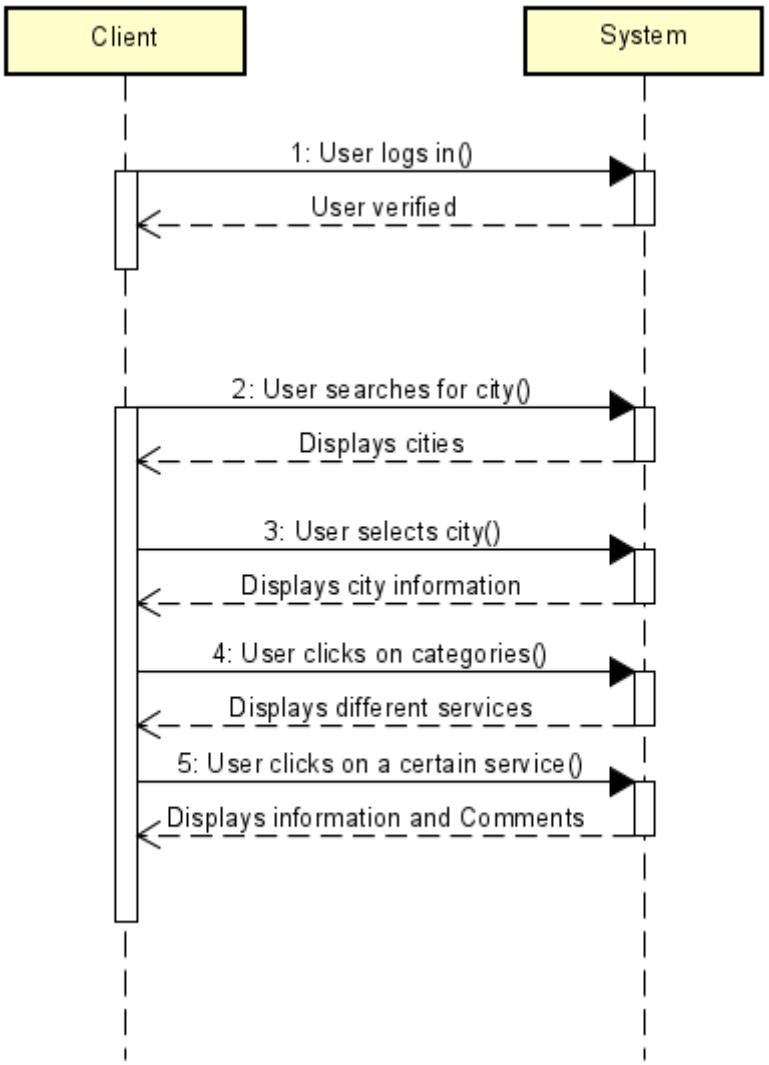


Figure 4. System Sequence diagram

This diagram was made based on a thorough use case description that outlines the steps a user takes after checking in, looking for a preferred city, and seeing information along with users comments.

1.7 Domain Model

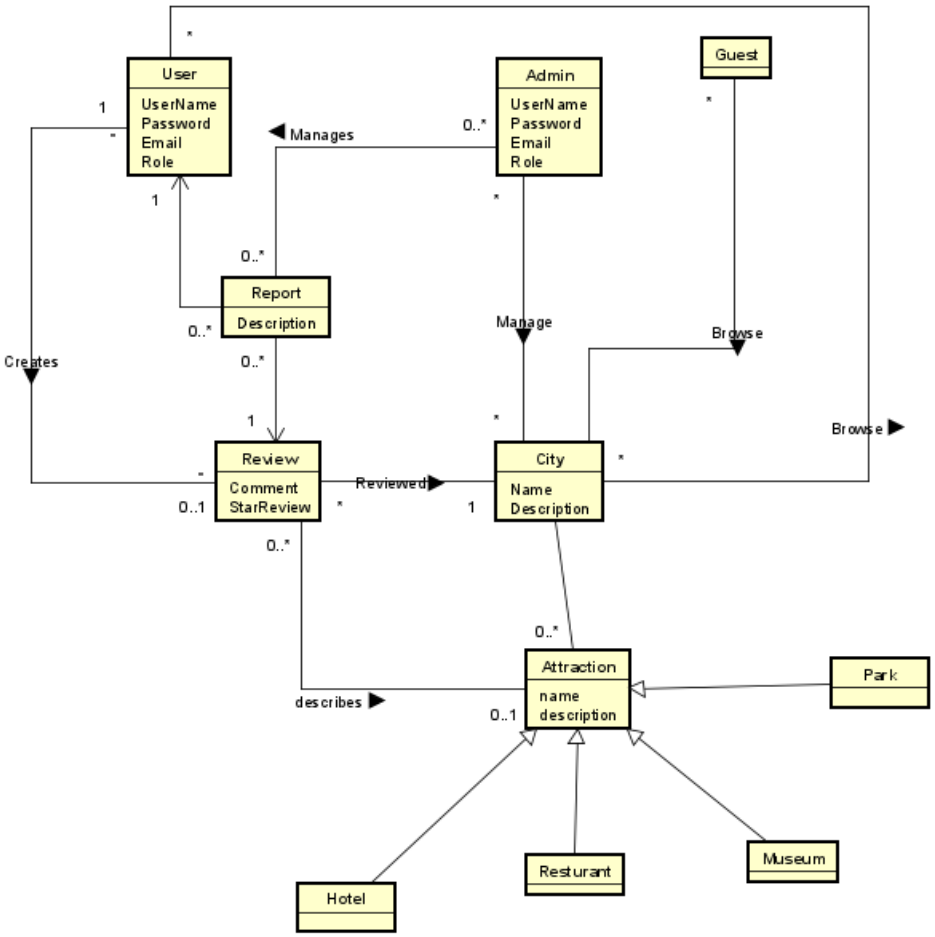
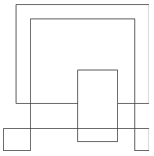


Figure 5. Domain model

After careful research, the domain model was developed, displaying the various entities. While an admin moderates reviews and manages reports, users can report and submit reviews.



Security

The project will use JWT tokens for authentication purposes to distinguish the different functions a user or admin have access to.
In this section , there will be an overview of potential threats that could arise by looking at the container diagram.The main reason is to identify all possible weaknesses and showcase possible solutions in the Design section of the report.

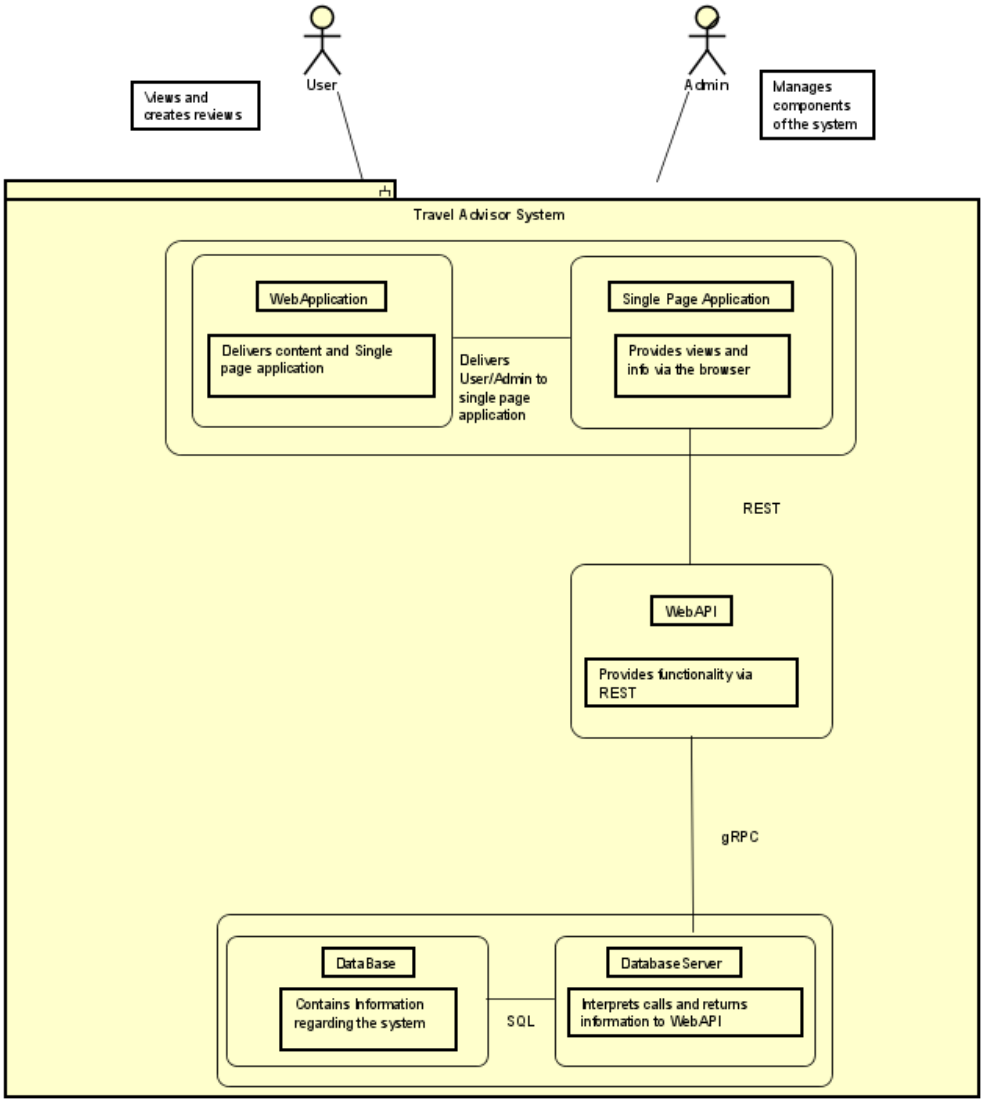
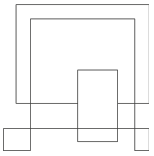
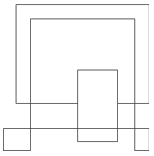


Figure 6. Container diagram

In the system there are 2 trust boundaries between the client and client and the Web Api through REST and the other between the WebAPI and the database server through gRPC where there is no authentication being done on both sides and there is a degree of trust that each side is communicating with the right entity respectively.



Risks	Frequency Scale: 1-5	Severity Scale: 1-5	Effects	Preventive Measures
Spoofing	1	3	Users abusing admin functions or creating reviews under another user's account	User authentication,two factor authentication
Tampering	3	5	Data that are displayed can be defaced with false or unpleasant information	Encryption,Hash keys
Repudiation	1	4	Anonymous reviews	Use of accounts in order to track which review has been created by who, Digital signatures
Information Disclosure	2	3	Leaks of password or emails	Strict authorization on methods that could retrieve sensitive data (eg,passwords) generic errors to prevent easy understanding of logic
Denial of Service	3	5	Services will be down allowing no access to pages	Addition of one user account can be created per email(which is not currently present) in the register.Distributing of traffic,firewall.



Elevation of privilege	1	5	Massive nuisance as unauthorized personnel could destroy user curated reviews and information	Strict access checks,removal of unnecessary rights,Black box testing

Figure 7. Risk table

Means of the attacker:

Passive:

Nefarious individuals may be able to eavesdrop on communication or undergo traffic analysis which could lead to a leak in sensitive information.This type of attack can not be easily detected but could be avoided with the help of encryption methods.

Eavesdropping:Potential breach of data when being transferred through trust boundaries.

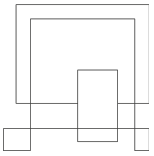
Traffic Analysis:Hack may attempt to analyze traffic in a network in order to get data.

Active:

Hackers may brute force through the system or modify vital transmissions that pose massive damage to the system and its users.While it can be easily detected,numerous checks would be required over each component to verify there are no breaches through which hackers could exploit.

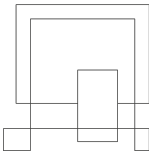
Masquerade: A hacker may attempt to deceive a component as legitimate and receive critical information.

Replay: An attacker may capture a message and replay it to the receiver and receive a response from the server



Modification: An attacker may intercept one or all messages and modify them as there are no checks verifying the integrity of the message from beginning to end

Denial of Service: An attacker may completely block all communication due to no firewall or strict user creation procedures.



2 Design

Architecture Pattern

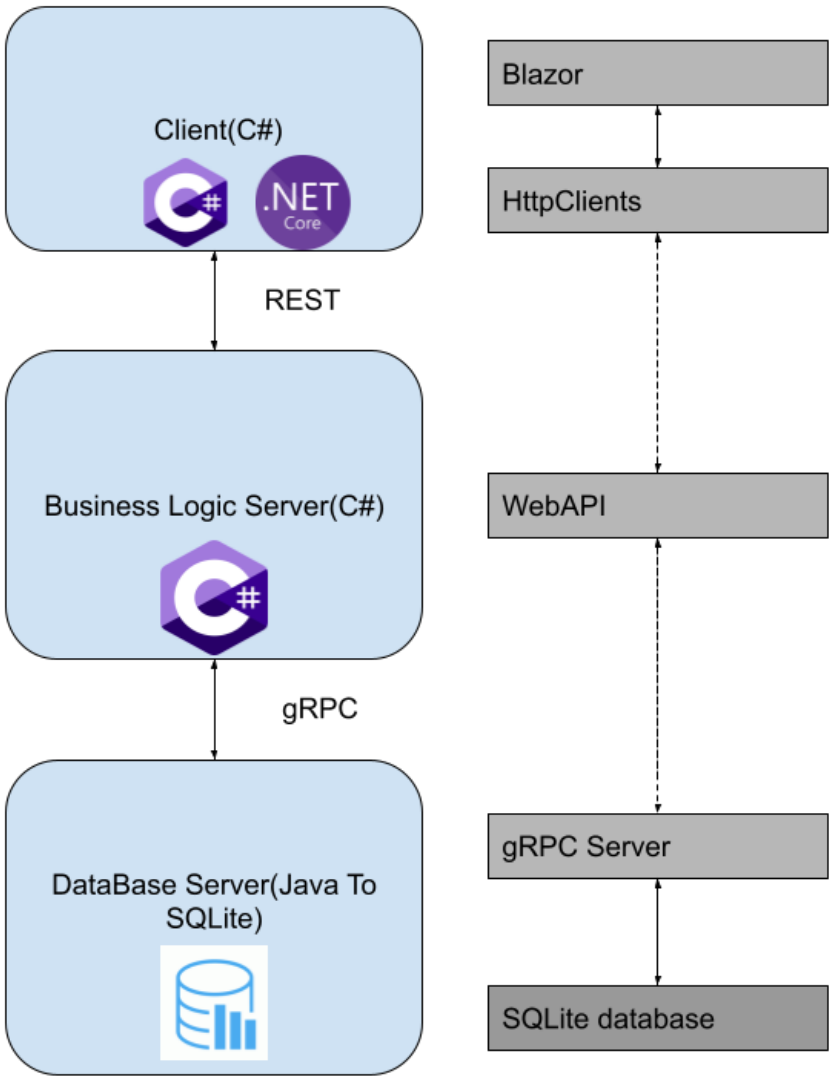
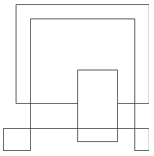


Figure 8. Architecture diagram

The project is based on three layer architecture consisting of the client , business logic server and the database server.The different layers are separate



and loosely coupled for easier maintenance and testing while keeping SOLID principles into consideration.

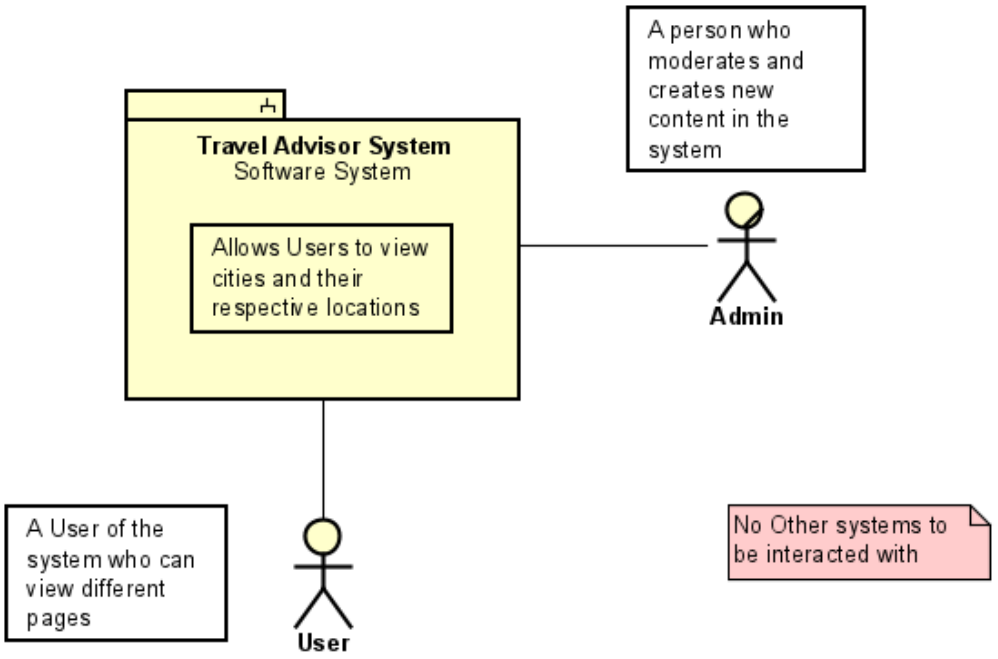


Figure 9. System context diagram

Above is the system context diagram, showing the boundaries of the program and does not interact with any exterior system.

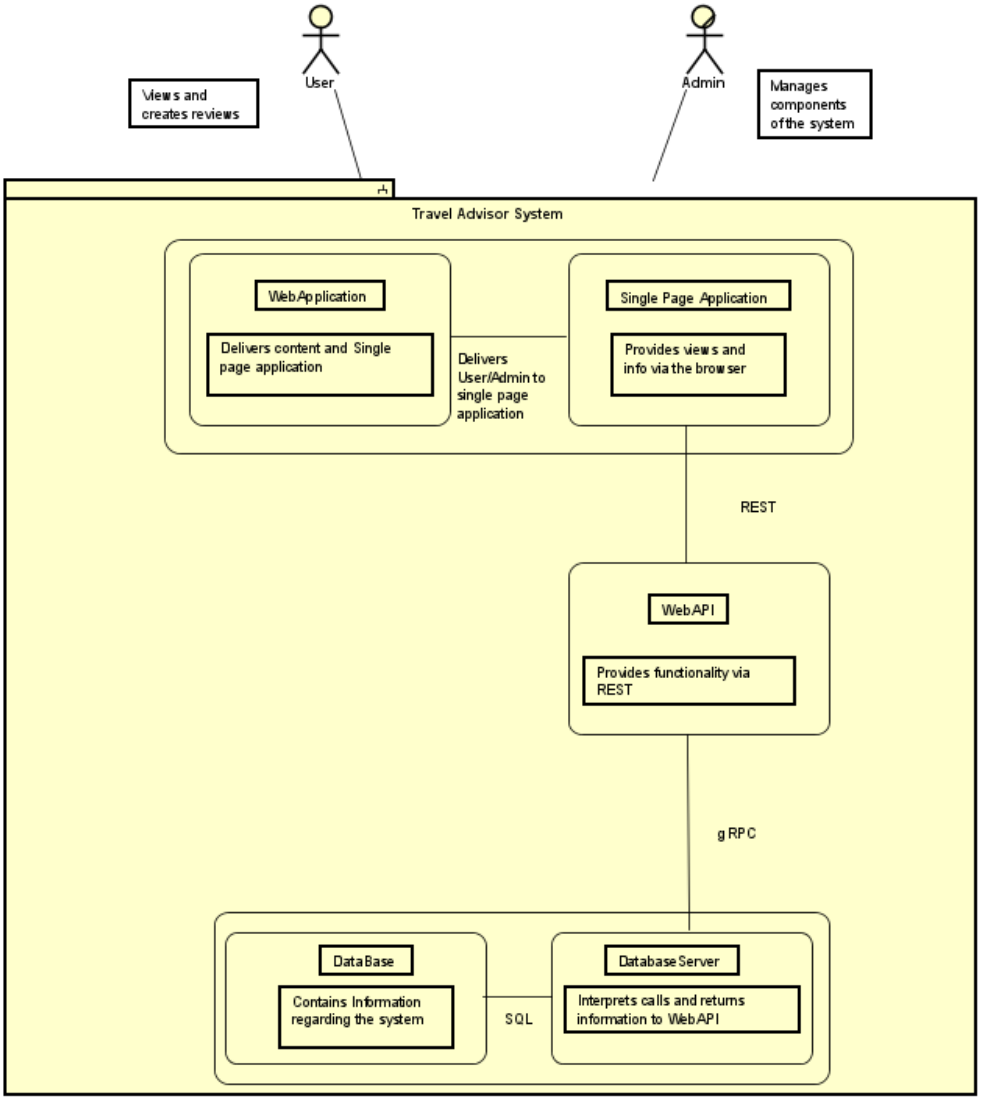


Figure 10. Container diagram (detailed)

Here is a more detailed Container diagram showing the different responsibilities of the system. Users can browse and add reviews while the administrator manages system components. Database Server, which has a SQL database, is connected to WebAPI by gRPC, and WebAPI is connected to Web applications and Single page applications by REST.

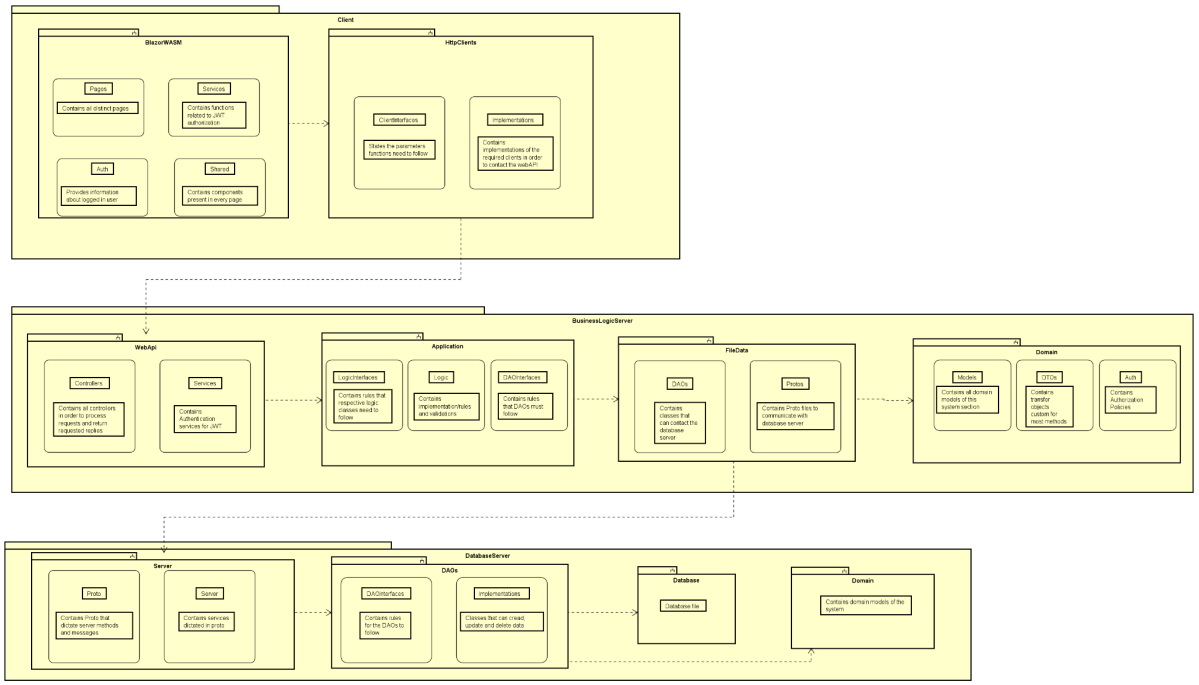


Figure 11. Component diagram

The component diagram is further detailed and explains the functions of the different packages present in the project.

Class Diagram

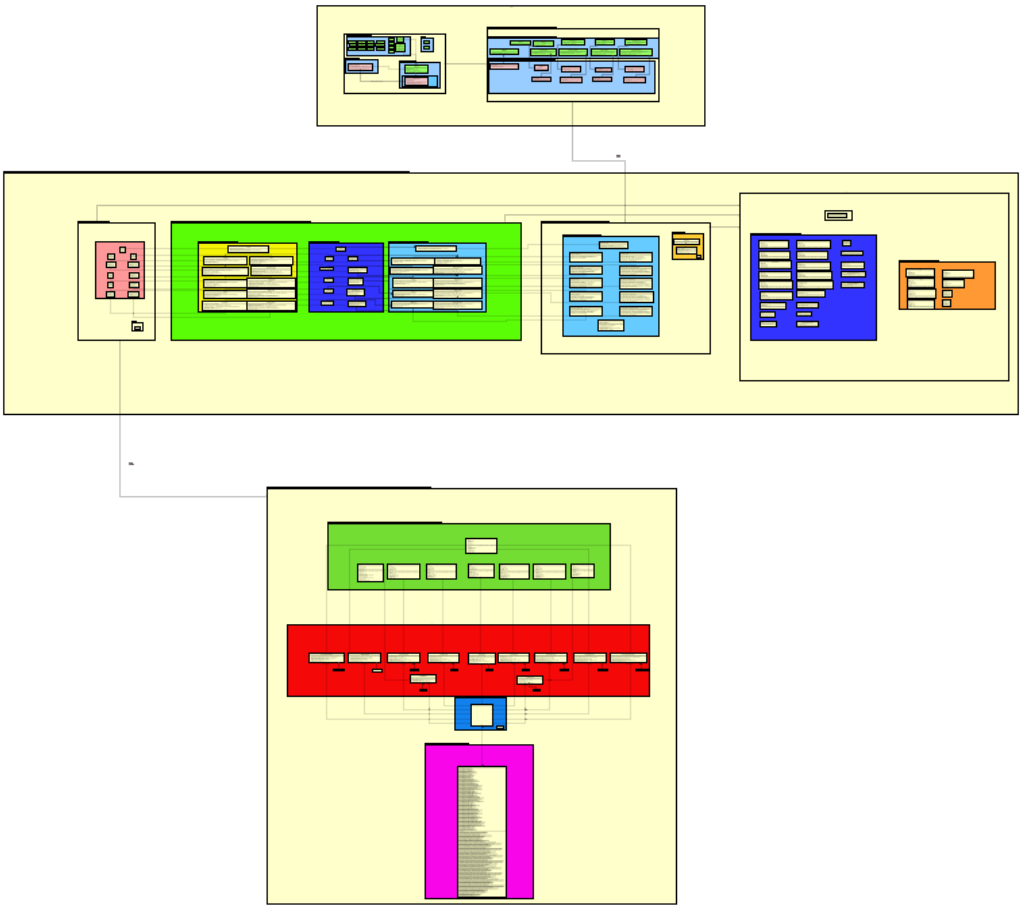
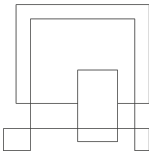


Figure 12. Class diagram(Appendices/Appendice-A-ClassDiagram)

The class diagram consists of three packages consisting of the client, business logic server and the database server.Their relationships and methods are detailed along with the interfaces each class inherits.The client package consists of the Blazor views and its respective httpclients required to contact the business logic server.The business logic has the controllers,logic and its data access classes which can contact the database server.Finally the database server has access to the database and the grpc services available to call.



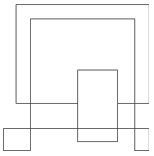
Technologies

Online communication: Discord, Messenger.
Java: Intellij was used for writing the Java section of the project.
Database: SQLite was used as the database.
JavaDataBaseConnectivity: sqlite-jdbc was used to access the database
SQL: Datagrip was used to construct the database and queries.
C#: Rider was used for writing the C# section of the project.
Diagrams: Astah,Google Drawings was used to create relevant design and analysis diagrams.
GUI: Blazor was used to create our GUI for the client due to prior experience with HTML and CSS with knowledge easily transferred
Network Technology-1(Between client and Server): REST for the client to make requests that will be handled by the server.
Network Technology-2(Between Server and Database Server):gRPC for communication between database server and business logic server.
GitHub: Github was used as required to track and help collaboration.
BloomRPC: BloomRPC was used to easily test the response from functions in the Database Server
Authentication:JWT
HTML(in addition to CSS):HTML was used to create a prototype view to be later incorporated into Blazor
Maven: To add dependencies relevant to gRPC and JDBC

SOLID

SOLID principles were taken into account while designing the classes and to pay attention to avoid violations

Single Responsibility Principle:



Each class was designed such that it has a single responsibility and does not handle multiple operations in order to avoid conflicts with merges and version control.

Open-Closed Principle:

A number of interfaces were employed in order to be ready for extension of more methods while keeping already created methods safe from modification.

Interface Segregation Principle:

All classes implement interfaces that are only required of them and have no unnecessary implementations.

Interaction Diagrams

Sequence diagram

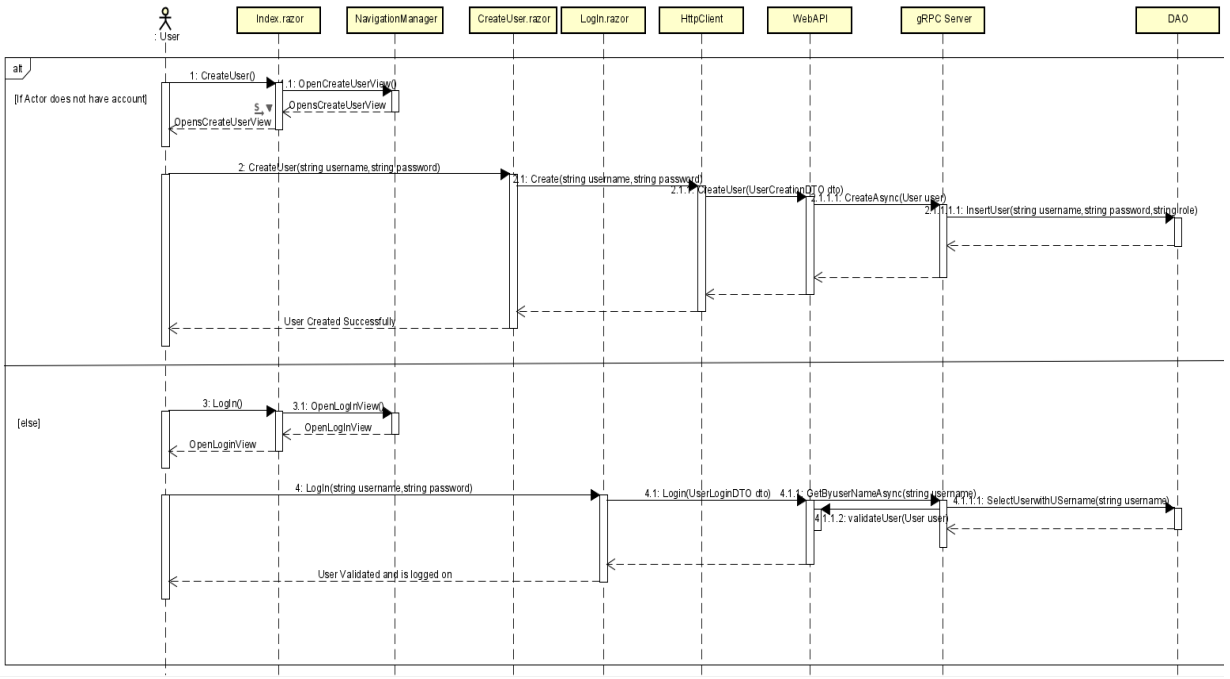
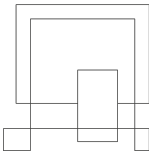


Figure 13. Sequence diagram

An overview of the procedure for users generating accounts may be found in the diagram above. It is made up of a number of elements that are represented by lifelines, and it shows how these elements communicate with one another along each lifeline as the process progresses. If there is no existing account, Actor initiates creating the account by CreateUser method using index razor, the navigation manager opens createUserView. Actor afterwards starts creating



user, CreateUser razor by inserting username and password , using HttpClient contact the WebAPI, WebAPI contacts the gRPC Server and inserts into DAO.

The DAO, in the same way, gives back the successfully created user. If there is an existing account, the user initiates the login method using index razor, navigation manager opens the login view. User logins with username and password using login razor, directly contacting WebAPI and gRPC and inserting into DAO.

Data Models

The EER diagram represents the essential concepts and relationships inside the database design that was generated and conceptualized from the domain model. It is a foundation for the eventual database structure by visualizing the entities, characteristics, and their relationships.

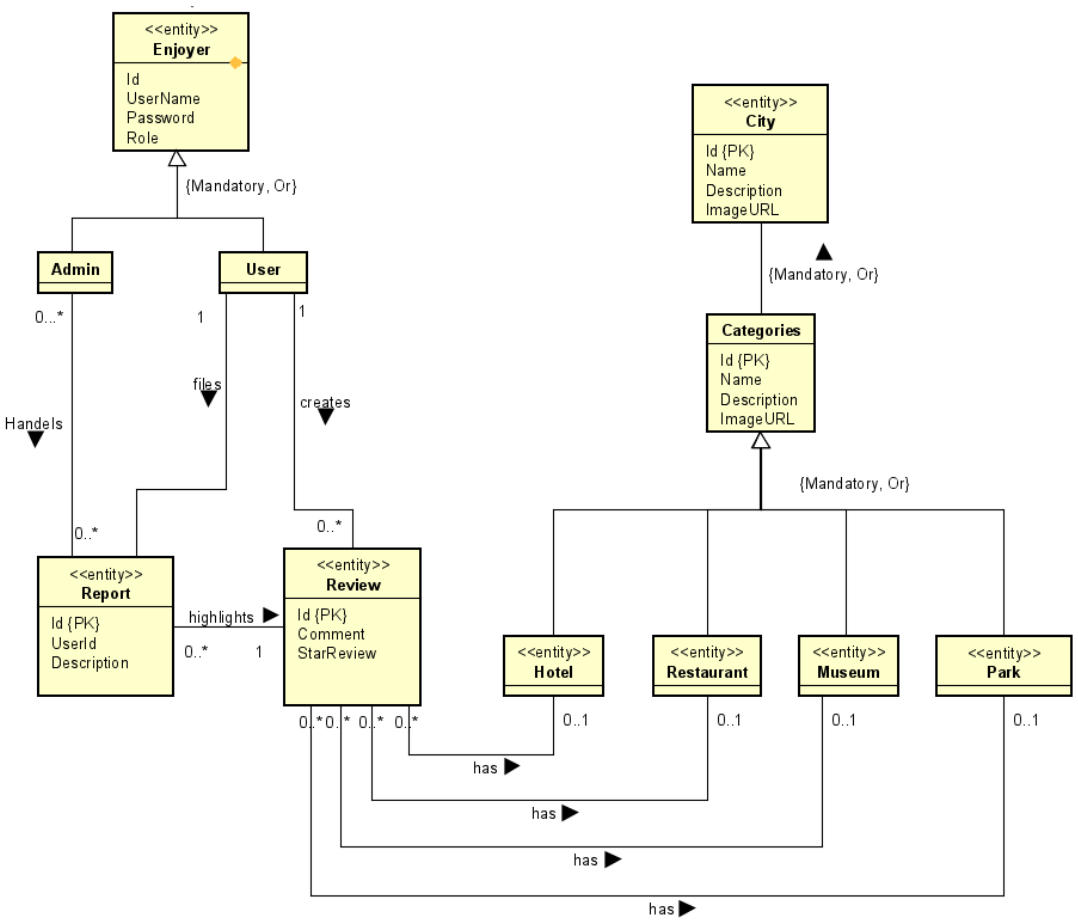


Figure 14. EER diagram

Global Relation Diagram

Global Relation Diagram is a visual representation of the relation schema that presents a database structure.
The diagram was normalized from the beginning and was constructed in the database. Early in the project, the group gave normalization priority and constructed the global relations diagram.

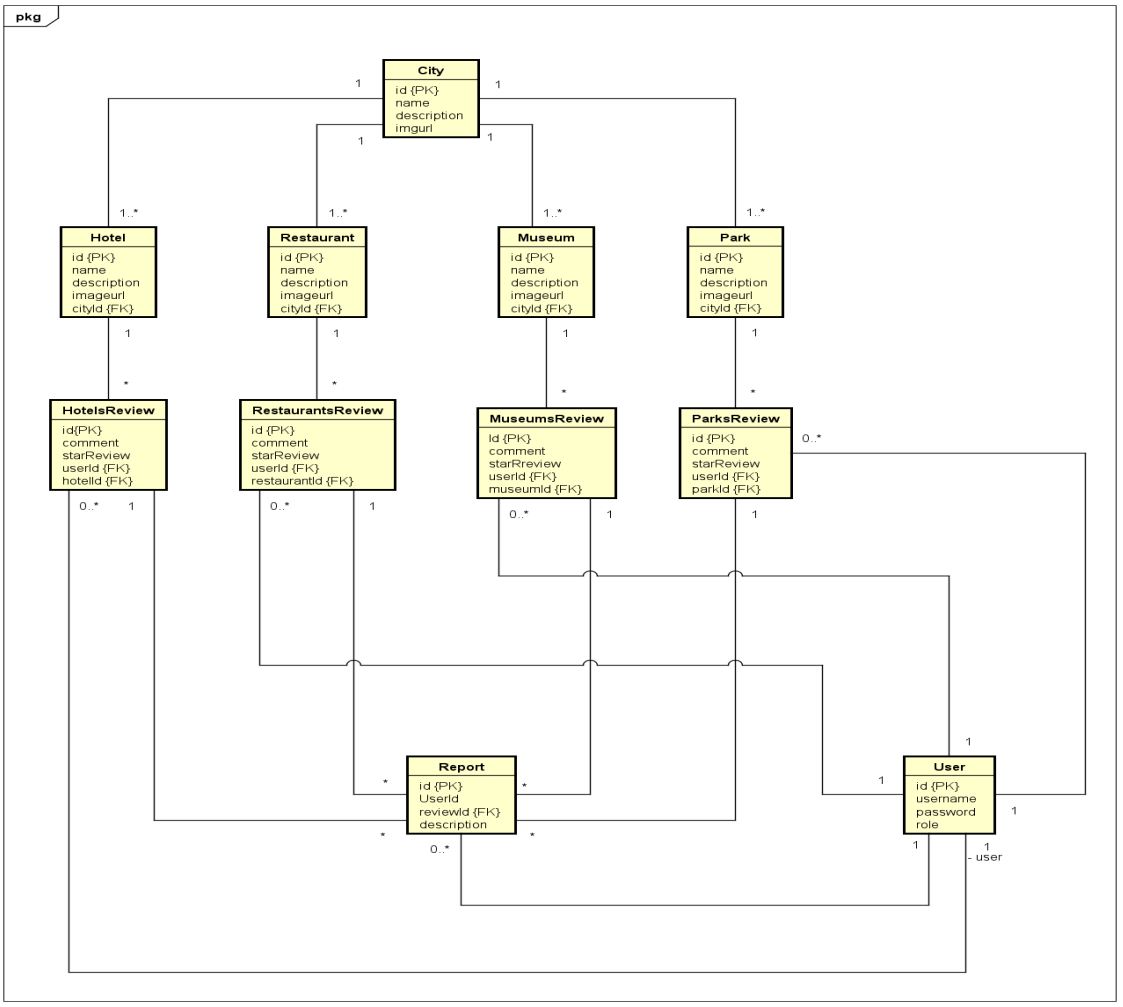
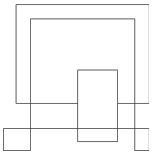


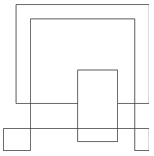
Figure 15. Global Relation Diagram



Relation schema

Relation schema describes the organization and structure of a relational database with tables and their relationships between them. The names of the tables are specified also as primary and foreign keys.

User (id, username, password, role) PK: id
Report(id, EnjoyerId, description) PK: id FK: reviewId , references to Review FK: reviewId , references to Review FK: reviewId , references to Review FK: reviewId , references to category Review FK: USERID
City(id,name, description, imageurl) PK: Id
Hotel (id,name, description, imageurl) PK: Id FK: CityId references to City
Restaurant(id,name, description, imageurl) PK: Id FK: CityId references to City
Park (id,name, description, imageurl) PK: Id FK: CityId references to City



Museum (id,name, description, imageurl) PK: Id FK: CityId references to City
MuseumsReview(id, comment, starReview, userId, museumId) PK: id FK: userId references to User FK:museumId references to Museum
ParksReview(id, comment, starReview, userId, parkId) PK: id FK: userId references to User FK:parkId references to Park
RestaurantsReview(id, comment, starReview, userId, restaurantId) PK: id FK: userId references to User FK: restaurantId references to Restaurant
HotelsReview(id, comment, starReview, userId, hotelId) PK: id FK: userId references to User FK: hotelId references to Hotel

Figure 16. Relation Schema

UI design choices

In the beginning a prototype view of how the final system would look was created in HTML and CSS which would later be incorporated into Blazor with some difficulty.
Some pages for log-in or creating a user were not made.
Prototype(HTML,CSS,JS):

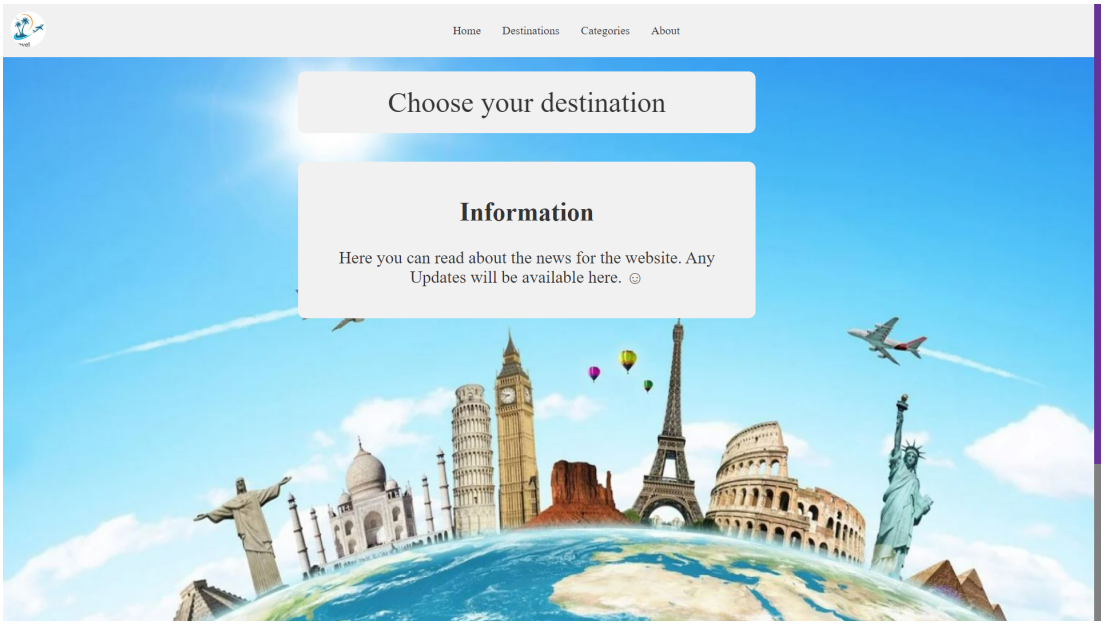


Figure 17.1. Picture 1.

There was some difficulty in integrating the CSS as a more modern design would be chosen for the pages making it look clean and sleek.

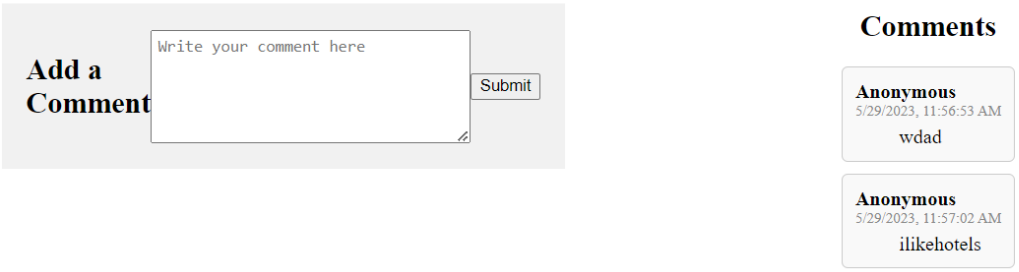


Figure 17.2. Picture 2.

There was also use of javascript for the comment section which was completely ignored for integration in blazor and the view for comments was revamped as adding star reviews was added.

Blazor:

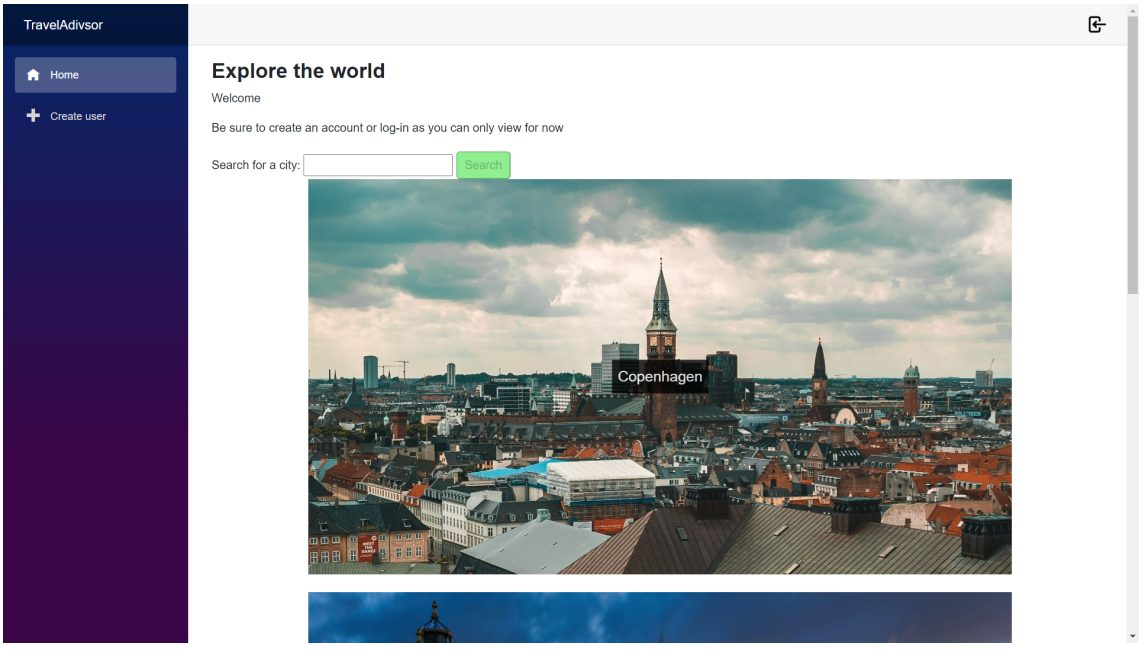


Figure 17.3. Picture 3.

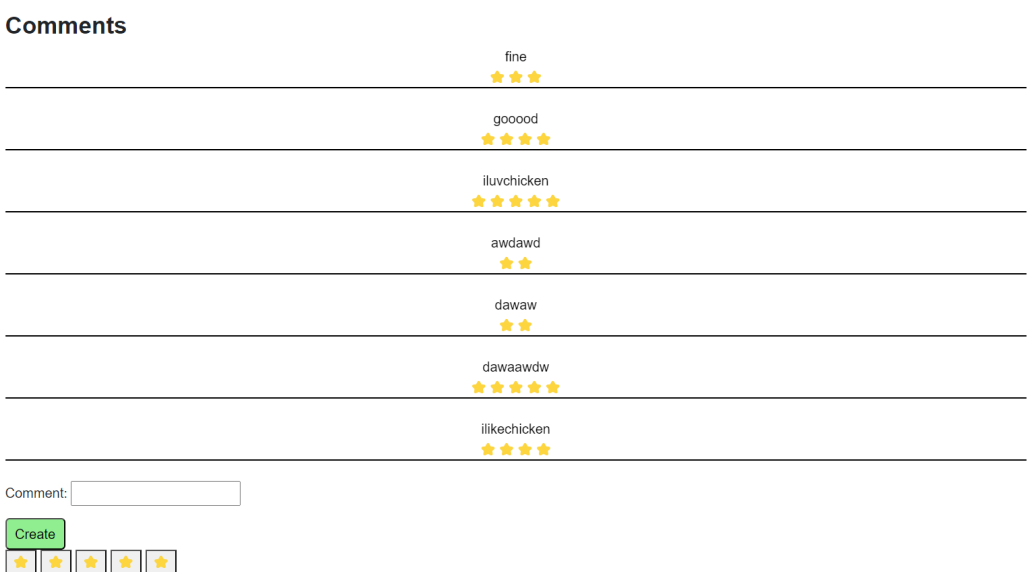
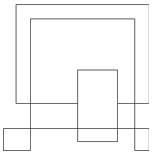


Figure 17.4 Picture 4.

The images present on the website are all taken from the internet and depend on it still being available as URLs.
There was an underestimation of the difficulties in integration but in the end a viable UI was made which could help users navigate the system and further explained in the user manual.



Security

Server to Server authentication

Currently, there is no between the business logic server and the database server and there is only a certain assumed trust as they both communicate on the same port.
So possibly, a check can be done where both the servers authenticate each other by having certificates in order to resume communication.

Asymmetric Encryption

RSA algorithm could be implemented for when users create accounts consisting of username and password when being stored or called for login as hackers may intercept the password which could be catastrophic for users who reuse passwords for other essential online accounts.

Message authentication

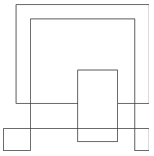
Regarding message integrity, Elgamal encryption could be used for at least the authentication of users to avoid tampering of data or corruption .

SSL certificate

A SSL certificate would be obtained for the website due to handling of sensitive user data (and in the future email addresses) and to verify that legitimacy of the website from possible duplicate websites designed to dupe users into entering data for nefarious purposes

PGP

PGP could be used to securely store and retrieve data in the database.



3 Implementation

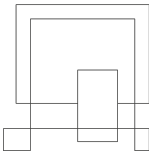
In the beginning, a concept of how the GUI would look was created in HTML with CSS. A WebAPI Project was created for the server with json used as rudimentary storage which eventually was swapped out for the Java database connected through gRPC.Eventually a Blazor project was created with clients to contact the server through REST.The prototype HTML pages were salvaged and CSS taken for the design of elements on each page.

The project's earliest phases were devoted to constructing the graphical user interface (GUI), with the help of an HTML and CSS concept. It created a visual representation, giving an intuitive design, by utilizing HTML and CSS. A WebAPI Project was simultaneously created as the system's server-side component. JSON was initially utilized as a simple storage option for storing and retrieving data and afterwards with a more reliable database solution. The team gradually switched out the JSON storage for a Java database connected via gRPC.

This change gave the system a more dependable and expandable data storage mechanism, guaranteeing that it could successfully handle growing data quantities and user interactions. Additionally, a Blazor project was established to develop the system's client-side components. The team was able to create clients using Blazor that could connect to the server via RESTful APIs, enabling smooth data transfer and interaction between the front-end and back-end components.

As a result of choosing the three tier architecture, our program has three tiers as follows:

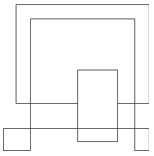
An overview of the project’s most essential feature which is creating a review for a specific category will be traced from the view to the database.



The Client:

The application begins with the main view consisting of a list of all cities after which the user may log-in or create an account if not done so then continue exploring a specific city and then when the user finds a hotel,restaurant etc, are able to view all the comments left by other users if present.

Figure 18.1. User Comment picture



The following would be the alternate view if the user has logged-in.

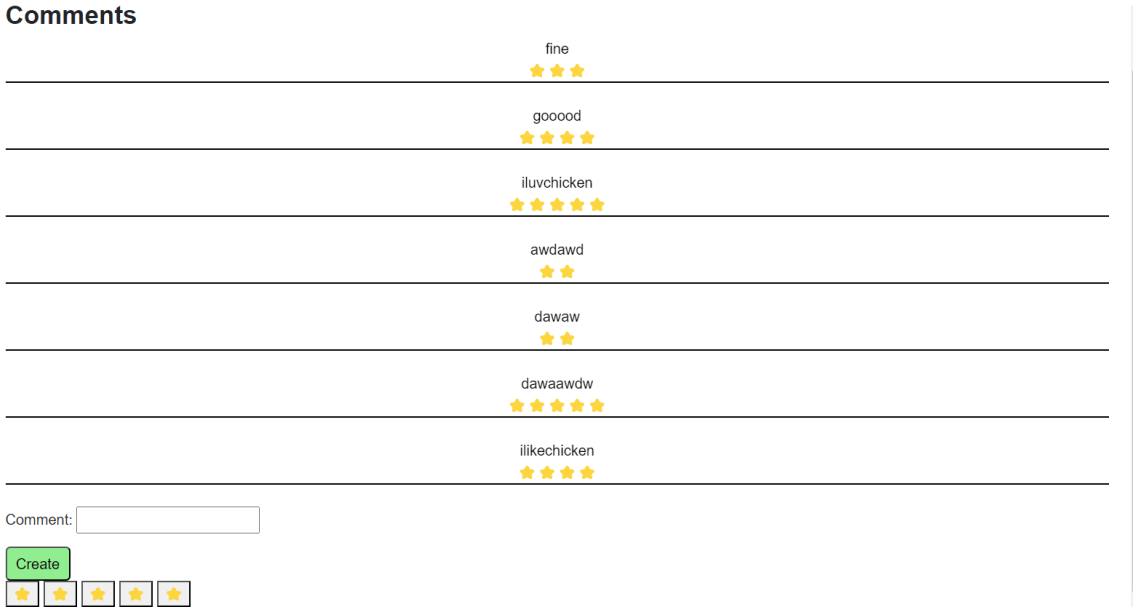
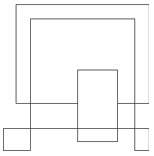


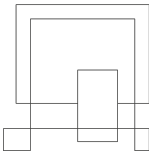
Figure 18.2. Admin comment picture



```
<AuthorizeView>

    <Authorized>

        <div>

                <label>Comment:</label>

                <input type="text" @bind="comment" @bind:event="oninput"/>

            </div>

            <div class="button-row">

                <button @onclick="Create" class="acceptbtn">Create</button>

                <br/>

                <button @onclick="SelectStar1">&#11088;</button>

                <button @onclick="SelectStar2">&#11088;</button>

                <button @onclick="SelectStar3">&#11088;</button>

                <button @onclick="SelectStar4">&#11088;</button>

                <button @onclick="SelectStar5">&#11088;</button>

            </div>

    </Authorized>

    <NotAuthorized>

        <div>

            <p>Log in or create an account in order to leave a review :D</p>

        </div>

    </NotAuthorized>

</AuthorizeView>
```

Figure 19.1. Code snippet of user comment, it displays the logic of the stars review

So, a guest is unable to view the create a review section of the page it is a feature that is authorized by logged in accounts.



The create button on click calls the Create method which after injecting the http client interface for restaurants reviews passes along four parameters in a DTO being

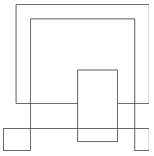
the string comment,

the integer starreview (which is selected but the 5 corresponding star buttons) ,

the user id which is obtained from the user's claims and,

```
AuthenticationState authState = await AuthState;

ClaimsPrincipal user = authState.User;

isLoggedIn = user.Identity != null;


if (!isLoggedIn) return;


userClaims = user.Claims;


stringselectedUserId = user.Claims.First(claim => claim.Type.Equals("Id")).ToString();


string stringid="";
foreach (char VARIABLE in stringselectedUserId)
{
   if (Char.IsDigit(VARIABLE))
   {
        stringid = stringid + VARIABLE;
   }
}


selectedUserId = Convert.ToInt32(stringid);
userId = (int)selectedUserId;
```

Figure 19.2. Code snippet of how user id is extracted from user claims



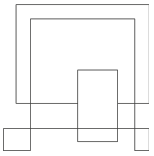
the restaurant id which is obtained from the two parameters in the page's parameters(the second parameter int id represents the restaurant id)

```
@page "/OpenRestaurant/{cityid:int}/{id:int}"
```

```
private async Task Create()

{

   try

   {

       await RestaurantsReviewService.CreateReview(new ReviewCreationDto(comment,
starreview, userId, id));

       reviews = await RestaurantsReviewService.getRestaurantsReviews(id);

   }

   catch (Exception e)

   {

       Console.WriteLine(e);



       throw;

   }

}
```

Figure 19.3. Code snippet detailing the method creating a review

Now in the implementation of the http client, where the client sends POST request, awaits a response and returns the review to show task is complete.

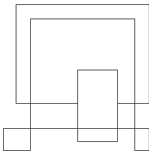


```
public async Task<Review> CreateReview(ReviewCreationDto dto)

{

   HttpResponseMessage response = await
client.PostAsJsonAsync("/RestaurantsReview",dto);



   string result = await response.Content.ReadAsStringAsync();

   if (!response.IsSuccessStatusCode)

   {

       throw new Exception(result);

   }



   Review review = JsonSerializer.Deserialize<Review>(result)!;

   return review;



}
```

Figure 19.4. Code snippet showing how the client is connecting to the web api

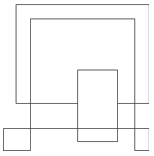
The Business Logic Server:

The controller dispenses the task to the corresponding logic class



```
[HttpPost]
public async Task<ActionResult<Review>> CreateAsync(ReviewCreationDto dto)
{
    try
    {
        Review review = await restaurantsReviewLogic.CreateAsync(dto);
        return Created($"/reviews/{review.Id}", review);
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return StatusCode(500, e.Message);
    }
}
public async Task<Review> CreateAsync(ReviewCreationDto dto)
{
    Review review = new Review()
    {
       Comment = dto.CreateComment,
      StarReview = dto.CreateStarReview,
      UserId = dto.UserId,
      CategoryId = dto.CategoryId
    };
    Review created = await restaurantsReviewDao.CreateAsync(review);
    return created;
}
```

Figure 19.5. Code snippet of the review creation in the web api controller

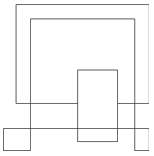


Here the logic class converts the DTO into the corresponding domain class Review and then makes a request to the DAO class to contact the database .

There could be more checks added here such as swear filters or some data validation which could be added in the future.

```
private Access.AccessClient client;


public RestaurantsReviewDao()
{
    var channel = GrpcChannel.ForAddress("http://localhost:9090");
    client = new Access.AccessClient(channel);
}


public Task<Review> CreateAsync(Review review)
{
    ReviewToCreate request = new ReviewToCreate()
    {
        Categoryid = review.CategoryId,
        Comment = review.Comment,
        Starreview = review.StarReview,
        Userid = review.UserId
    };


    var send = client.CreateRestaurantsReviewAsync(request);


    return Task.FromResult(review);
}
```

Figure 19.6. Code snippet of the review creation from the gRPC client DAO

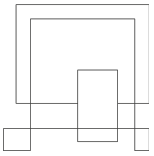


Here the Create method converts the domain class into the corresponding message class which could have been done in the logic class which was an oversight.Then the request is sent on the port the server listens on and returns Task completed with specified review.

The database Server:

```
@Override

public void createRestaurantsReview(DataAccess.ReviewToCreate request,
StreamObserver<DataAccess.ReviewCreated> responseObserver) {

   String comment = request.getComment();

   int starreveiw = request.getStarreview();

   int userid = request.getUserid();

   int categoryid = request.getCategoryid();




   restaurantReviewDAO.insertRestaurantReview(comment, starreveiw, userid,
categoryid);



   System.out.println("Received request ==> " + request);

   DataAccess.ReviewCreated response = DataAccess.ReviewCreated.newBuilder()

           .setCode(200)

           .setComment(request.getComment())

           .setStarreview(request.getStarreview()).build();



   responseObserver.onNext(response);

   responseObserver.onCompleted();

}
```

Figure 19.6. Code snippet of the review creation request from in the database server



Our server overrides methods generated from the proto file and implements the ImplBase with the corresponding one simply using the DAO to insert the attributes from the request into the database and sending back the appropriate message response.

```
public class DataAccessService extends AccessGrpc.AccessImplBase
```

Figure 19.7. Code snippet of the extension of the protocol buffer in the database server

4 Test

BloomRPC

GRPC services were tested and interacted with using bloomRPC. It offers a direct way to send gRPC requests, examine answers, and investigate gRPC API features. Utilizing BloomRPC as a gRPC client allowed us to communicate with the services, check replies, and find any problems or discrepancies during development and testing. This allowed us to test and validate the gRPC services quickly.

For instance, we may rapidly check how each method in the proto file responds.We successfully receive the proper data back in the response in the section below.

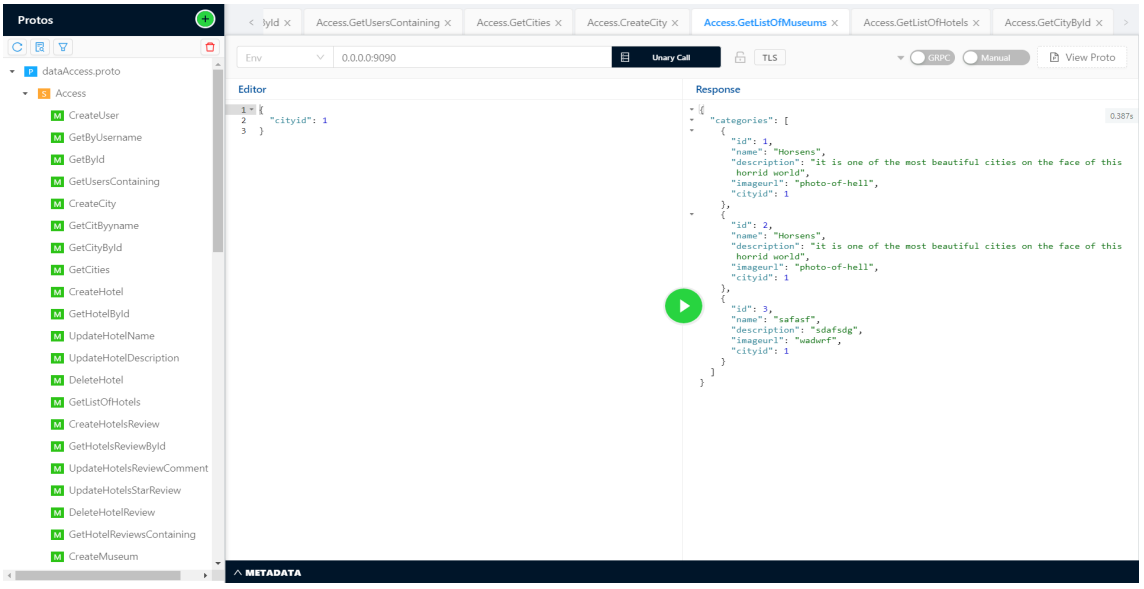
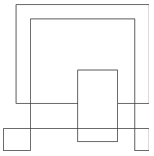


Figure 20. Snippet of BloomRPC



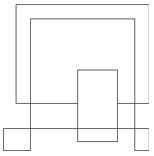
Junit

Junit testing was used to quickly and easily test DAO methods present in the Java section of the project if changes were made to verify the integrity of the methods as well as testing connectivity to the database.

```
@Test
void createCity() //Test that all attributes of a user are
sent and can be called
{
    cityData.insertCity("Mumbai","coolplace","niceimage");
    City findCIty=cityData.getCityByName("Mumbai");
    assertEquals("Mumbai",findCIty.getName());
    assertEquals("coolplace",findCIty.getDescription());
    assertEquals("niceimage",findCIty.getImageURL());
    cityData.deleteCity(findCIty.getId());
}
```

Figure 21. Code snippet of createCity() method tested

For example,the above test method tests whether an object can be inserted into the database file and if it can show all data columns (except for primary key Id) and is then promptly deleted from the table to prevent needless data.



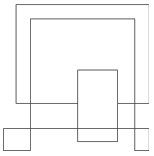
Nunit

Nunit testing was used in the C# section of the project to test the Data access methods to quickly verify the integrity of methods

```
[Test]
public void GetAdmin() //Test that user Role can be
confirmed as Admin
{
    IUserDao userDao = new UserDAO();

    Task<User?> user = userDao.GetByUsernameAsync("bobby");

    Assert.That(user.Result.Role,Is.EqualTo("Admin"));

}
```

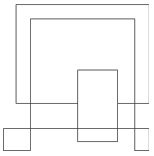
Figure 21.1 Code snippet of GetAdmin() method tested

The above unit test was used to verify that an account that is an admin can be searched in the database as it is an essential method in order to separate the functions that are only authorized by admin personnel.



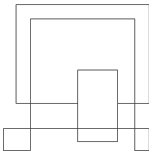
4.1 Test Specifications

Test Scenario	Action	Result	Status
User can browse through the system	Select city	View changes to selected city	Success
	Select Restaurants(or another list of category)	View changes to a list of restaurants of the corresponding city	Success
	Select desired restaurant(or another category) from list	View shows the details of the category along with user reviews	Success
User can create a review	Enter comment and select star review	System acknowledges requests and awaits to send	Success
	Enter create review button	System saves review and refreshes list with newly added review	Success
User can report a comment	Click report button	System opens new page where user can write a comment about the review in question	Failure
	Comment entered and click on send	System adds report which can be judged by an admin	Failure
Admin can delete a review	Click delete button(next to a comment)	System deletes review and refreshes the list of reviews	Success
Admin can create a category	Click on city	System opens City page	Success
	Click on category	System opens list of corresponding category	Success



	Click on Create “Category”	System opens view where a new category can be created for the city	Success
	Enter information and click create	New category is created	Success

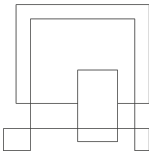
Figure 22. Test table



5 Results and Discussion

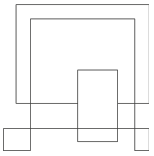
The outcomes and discussion section's goal is to present the project's outcome and results that were attained.The project’s main goal was to create a travel advisor system which allows users to leave reviews and to read unadulterated reviews from other users.As a result critical requirements were reserved the utmost priority before lower priority requirements could be completed.

User story	**Result**
As a user I want to be able to log in to the system.	Successfully implemented
As a user I want to be registered.	Successfully implemented
As an admin I want to be registered.	Successfully implemented
As a Guest I want to be able to browse the website.	Successfully implemented
As a user I want to be able to see a (display of information about) a city I searched for.	Successfully implemented
As an admin I want to be able to upload pictures to the posts.	Not implemented
As a user I want to be able to add one comment.	Successfully implemented
As a user I want to be able to delete my comments.	Not implemented



As a user I want to be able to have a wishlist/save visited city.	Not implemented
As an admin I want to remove comments that are problematic.	Successfully implemented
As a user I want to report comments.	Not implemented
As a user I want to send messages to an Admin.	Not implemented
As an admin I can choose to respond to messages from users.	Not implemented
As an admin I want to close comments.	Successfully implemented
As an admin I want to be able to approve/remove reported comments(The admin will be able to remove the comment from the reported comments database).	Not implemented
As a user I want to be able to add star reviews with my comments.	Successfully implemented
As an admin I want to edit text(in a description).	Not implemented
As a user I want to get verified as a businessman.	Not implemented

Figure 23. Result table



6 Conclusions

The goal of this system was to develop a web application that enables users to find and evaluate different amenities in towns all over the world, offering unbiased information gathered from both locals and tourists. The system was designed to give users a place to research and rate various services, including dining establishments, lodging options, tourism attractions, and more.

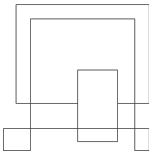
The team carefully investigated the issue at hand throughout the analysis phase and identified various crucial system needs.

The team developed use cases that illustrated the precise interactions and features that the web application had to provide based on these important requirements. These use cases aided in defining the system's scope and providing direction for the following design and implementation phases.

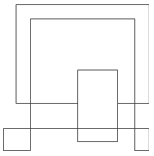
The project used a three-tier architecture, which is a popular design pattern for websites. The architecture most likely included a front-end presentation layer for user interface and interactions, a back-end business logic layer for processing user requests and handling key operations, and a database for storing and retrieving pertinent data.

Both gRPC and REST were used by the system in terms of network technologies. gRPC was probably used to facilitate effective communication between various system components, enabling high-performance and scalability interactions. REST, on the other hand, may have been used to integrate with third-party services or expose APIs to other clients.

User stories and functional requirements that were established during the design phase of the project have to be converted into actual code during the implementation phase. During this stage, unit testing was vital since it allowed



the team to confirm the accuracy and functionality of individual methods and components. This strategy ensures the overall quality and dependability of the system by assisting in the early detection and correction of problems.

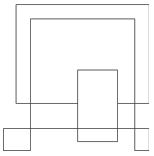


7 Project future

The current architecture of the system allows for an easy extension of features with the main focus being the various user stories that could not be implemented in time in order to improve user satisfaction with the program.

Goals:

- Wishlist(in regards to cities where users can save destinations to a tab)
- A new user type:business owner(to reply to comments on their respective business)
- Report system(where users may report other comments which can then be assessed by the admin)
- Automatic swear detection
- Implement more of SOLID principles everywhere and fix possible discrepancies
- Add more information for each of the categories(Eg, boolean isVegetarian for restaurants)



8 Sources of information

The Economist Intelligence Unit Limited.

Tourism [Outlook 2023]
Available at:

https://www.eiu.com/n/campaigns/tourism-in-2023/ [Accessed May 2023].

Skyscanner report

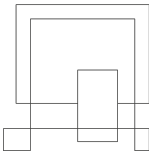
Skyscanner Ltd 2002-2023

Available at: https://www.skyscanner.net/news/travel-trends [Accessed May 2023].

Tegan, S., 2015. Why you should never, ever trust a travel agent-[online] Available at:

https://thenewdaily.com.au/life/travel/2015/04/28/never-ever-use-travel-agent/

[Accessed May 2023].



9 Appendices

Appendix-A-Class Diagram

Appendix-B-ProjectDescription

Appendix-C-UserManual(User)

Appendix-D-UserManual(Admin)

Appendice-E-Logbook

Appendix-F-UseCaseDescription

Appendice-G-Architecture,Context,Component,Container

Appendix-H-CATWOE,Richpicture

Appendix-IN-ActivityDiagrams

Appendix-J-DataModels

Appendix-K-DomainModel

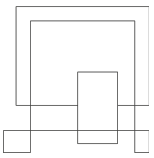
Appendix-L-SequenceDiagram

Appendix-M-SystemSequenceDiagram

Appendix-N-UseCaseDiagram

Appendice-O-ProcessReport

Appendix-P-ProjectReport



List of figures and tables

Figure 1. Use case diagram

Figure 2. Use case description table

Figure 3. Activity diagram

Figure 4. System Sequence diagram

Figure 5. Domain model

Figure 6. Container diagram

Figure 7. Risk table

Figure 8. Architecture diagram

Figure 9. System context diagram

Figure 10. Container diagram (detailed)

Figure 11. Component diagram

Figure 12. Class diagram

Figure 13. Sequence diagram

Figure 14. EER diagram

Figure 15. Global Relation Diagram

Figure 16. Relation Schema

Figure 17.1. Picture 1.

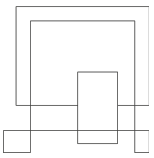


Figure 17.2. Picture 2.

Figure 17.3. Picture 3.

Figure 17.4 Picture 4.

Figure 18.1. User Comment picture

Figure 18.2. Admin comment picture

Figure 19.1. Code snippet of user comment, it displays the logic of the stars review

Figure 19.2. Code snippet of how user id is extracted from user claims

Figure 19.3. Code snippet detailing the method creating a review

Figure 19.4. Code snippet showing how the client is connecting to the web api

Figure 19.5. Code snippet of the review creation in the web api controller

Figure 19.6. Code snippet of the review creation request from in the database server

Figure 19.7. Code snippet of the extension of the protocol buffer in the database server

Figure 20. Snippet of BloomRPC

Figure 21. Code snippet of createCity() method tested

Figure 21.1 Code snippet of GetAdmin() method tested

Figure 22. Test table

Figure 23. Result table