



Large System Development: Hotel System

Group 1:

Pravien Thaveenrasingam
Arkadiusz Maksymilian Paryz
Hassan Mohdi
Hallur Við Neyst
Gordon Mikkelsen
Kristoffer Rath Hansen
Justas Sa
Sean Altoft
István Farkas
Vlad Mircea Burac
Muhammad Umar Ulhaq
Mohammad Saad
Yakubo Adeyemi Oseni
Lovro Biljeskovic

Introduction	2
Toolbox Artifacts	3
Use Cases	4
What is a use case and why use one?	4
Logical domain Model	5
System operations	7
SLA (part 1)	9
Introduction	9
SLA Agreement	9
Uptime/availability	9
Mean response time	9
Mean time to recover	10
Failure frequency	10
Description of tools and experiences (part 2)	11
Continuous integration(azure pipelines)	11
What is Continuous Integration	11
What is Continuous Deployment	11
How we implemented CI and CD in our project and why	12
Artifactory(azure artifacts)	13
Overview	13
Using the artifacts through NuGet	14
Consume NuGet packages can be done in Visual Studio	14
Consume packages	14
The better way	15
Hosting(azure)	16
Logging set up	16
Problems with CD and Azure	19
Tools used for testing	20
Overview	20
Unit Testing	21
Integration Testing	21
System Testing	22
Risk Matrix	23
Reflection & Conclusion	25

Introduction

Overview

Our team has developed a shared booking system designed for multiple partnered hotel chains and their respective travel agencies who will be working with them. It has been decided that the system will comprise of both a 3rd party web-service portal as well as a web application. Through the use of our system these hotel-chains and their business partners will be able to track the availability of the rooms and guests in their different hotel locations, as well as the ability to book and cancel said booking of a specified hotel room.

For a greater understanding of how our system operates, we have included descriptions and analysis of the artifacts involved in our system, as well as the SLA agreement, logging and the test tools and coverage of our system. Models or visualisations such as the risk matrix and the logical domain model have also been included to help supplement understanding of the system.

A description of how the continuous deployment and continuous integration has been implemented during development of the system will also hopefully shed more light the intricacies of the system, the development cycle involved and how further development can be done in the future.

Context

Many of the tools and artifacts of our system will also contain brief introductions to the topics, so as to remain an educational report, but will otherwise contain higher level analysis aimed at students and educators. Some synonyms will be included and prior knowledge of system development tools and processes is expected.

Goals and non goals

Many of the goals for this work align with the course objectives, such as becoming aware of and overcoming some of the issues that arise in the workflow of a large team of developers. CI chains, CD, modelling processes and modern version control techniques will be just some of the ways that we have done this and will be elaborated on below in the report.

While we have not set up specified metrics or measurements of success for this project. The main measurement for success that we will be using will be whether we can meet deadlines

that were assigned to us by our stakeholders, our code fully meets the functional and non-functional requirements, that the code is deployable, that the code is scalable and of good quality of quality and that the documentation of the code that will be included in the following report is of good quality.

Toolbox Artifacts

An Artifact is any physical piece of information used or produced by a system. In Enterprise Architect these are represented by the Artifact element, which can have one of a number of stereotypes to tailor it to a specific purpose, including internal operations and structures within the model as indicated in the examples. Artifacts can have associated properties or operations, and can be instantiated or associated with other Artifacts according to the object they represent.

In our .NET Core Solution, our Artifacts are represented by the .cs files that make up the project, and they are split into multiple types.

In our backend project, you can find the building blocks of our application, starting from:

1. The Service Interface:

The interface for our choice of implementation can be found in a separate repository than our backend. We ended up using this pattern because it allows our backend to be coded in many different ways, as long as the interface is implemented as described.

2. Service Implementation:

Our implementation is where you can find how we use all the other types of Artifacts that we have in our application.

3. Behaviour:

We have an attribute artefact that extends the functionality of our service class. In it we do things like bind parameters that relate to our application (like ServiceDescription, ServiceHostBase, a collection of endpoints, and other binding parameters), the message dispatching behaviour (with a ChannelDispatcher), and, if needed, validation.

4. The Context:

In HotelContext.cs you can find a wrapper for getting and setting our database models.

5. The Models:

These serve as the objects that map onto our SQL Database structure, and help us with our CRUD operations.

6. The DTOs:

These Artifacts contain the structure of the data that is consumed and is produced by our service, and they are called Data Transfer Objects, a fitting name for their purpose. They can be found in the interface project.

7. The Mappers:

These are the artifacts that help convert our database models into the DTOs, for us to be able to implement the proper response in our Service Implementation's methods, as defined in the interface.

8. The Error Handler:

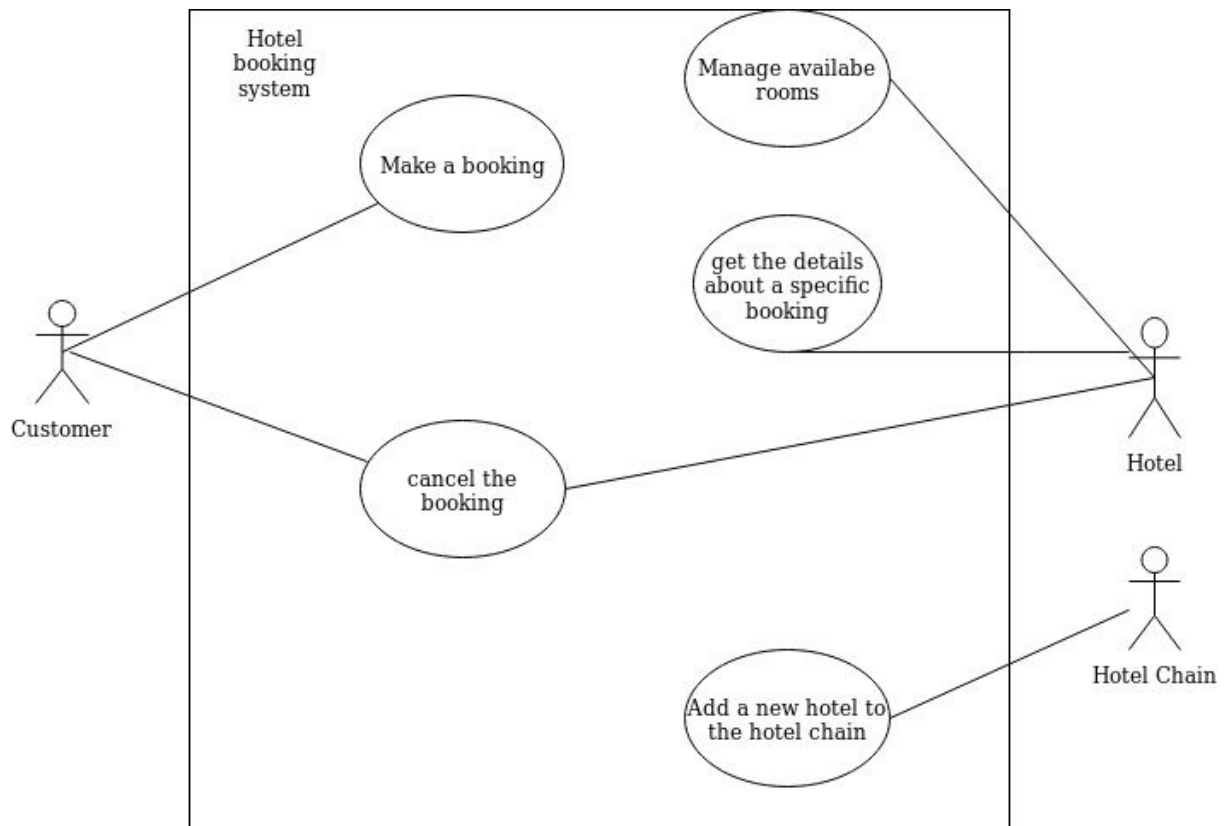
This is an artifact that is used throughout the application if an event needs to be logged.

9. Exceptions:

In the interface project we have custom exceptions defined, that are thrown in specific conditions, to help us during debugging or when interacting with the application and unexpected behaviour occurs.

Some of these artifacts were in our design from the start, but as we were implementing them, some others have organically become part of our design as well, as we felt there was a need to add new types of artifacts to satisfy some parts of our applications functionality and usability, as well as help ease our way of development (i.e. the error handler).

Use Cases



What is a use case and why use one?

Use cases are used to represent high-level functionalities and how the user will handle the system

As a group we felt that what would be invoking the features needed to be visualized, so that we could capture the dynamic behaviour of our system.

The process of our use case and what it represents

After a good discussion (and with some inspiration from our logical domain model), we finally agreed upon what the high level functions in our system should be. We decided on being able to: make a booking, cancel the booking, manage the hotel rooms, see details on the booking and finally add a hotel to the chain.

Then next we had the roles. At first we only had the hotel and customer in mind. However, as time went by we decided that the hotel chain also needed a role. Furthermore, there were some disagreements on who uses which functions (as for example: who cancels bookings?). After having discussed together with the group, and also with the teacher, we had settled for a final use case diagram. The result was the following: the customer can only create or cancel bookings. The hotel can also cancel bookings but has aswell access to managing rooms and getting details on a specific booking. The only job for the hotel chain, is to add new hotels to the chain.

Logical domain Model

What is a logical domain model and how did we use it

A logical domain model is used for showing the modelling of the objects that are required for a business purpose. By obtaining a high-level and abstract representation of our data, we are able to present important business processes and show how they would relate to each other. The creation of a logical domain model allows for the further creation of physical models and even database design.

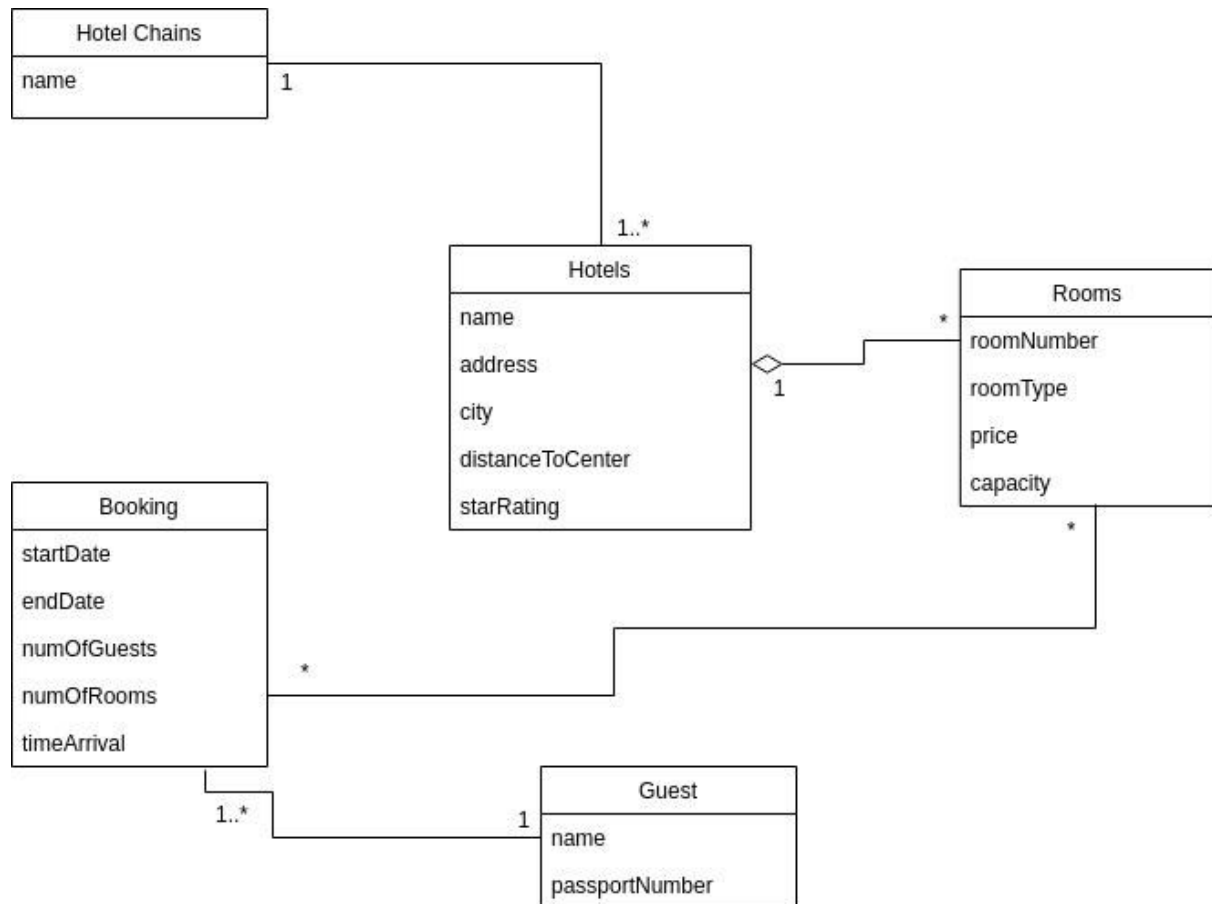


Figure 2 - Logical Domain Model representing a high level overview of the hotel booking system

The objects of our hotel booking system (described in detail below) are modelled in Figure X as follows:

A hotel chain can be the owner of one or more hotels. A hotel consists of one to many rooms. A booking can book one or more rooms (slight mistake in the model here) and a guest can make one or more bookings.

- **Hotel Chains:** relevant hotel chains that exist in the area. A hotel chain needs a name such as : Hotel Hilton. The definition of a hotel chain is a hotel business that has expanded from one to more locations with their hotels.
- **Hotels:** the hotels of the chain need names (one could be called ex: Hotel Hilton Kastrup). Also a location is needed for the hotel, more specifically the address and city. The hotel also needs to know how far away the center is (center is downtown in the same city as the hotel is). This is because it is a frequently asked question by the guests. Finally, the hotel needs to have a rating system, so that the chain knows what the guests think of the hotel.
- **Rooms:** There can be multiple of these “room” elements within a hotel. Rather than having a name such as that of hotel-chains or hotels, a room-number is instead used

as an identifier for a room instead. Rooms also contain a room-type, the room-type is a way for hotels to further differentiate rooms and to cater to the different needs of the customer and the hotel itself. The hotel-type will also affect the price of the hotel-room. The price is represented on a room basis rather than hotel basis as the rooms are not all uniform. Finally the rooms will also contain the maximum capacity of adults that the room is recommended to hold, this is dictated by figures such as room size and the number of beds.

- **Booking:** A booking is what a customer does when he wishes to reserve a room. It contains the start and end date of when the customer will be making use of the rooms. Other information will also be obtained, such as the number of guests and the number of rooms required by the customer. This figure also contains the time of arrival which is to be recorded when the guest(s) arrive at check-in.
- **Guest:** A guest is a customer that has reserved a room. It is not possible to book a room without being registered as a guest. This means having ones full name written down and the passport details.

System operations

System operational contracts are an advanced way of showing system behaviour, they help to indicate the changes in state of different domain objects within the system.

The following are some of the more relevant system operational contracts to operate our system.

Contract CO1: Create Booking

Operation: CreateBooking(DateTime startDate, DateTime endDate, int numberGuest, List<RoomIdentifier> listOfRooms, int passportNumber, string guestNumber)

Preconditions:

Postconditions:

- A booking instance is created
- Attributes of user was initialized
- User is associated with booking

Contract CO2: Cancel Booking

Operation: CancelBooking(BookingIdentifier bookingIdentifier)

Preconditions:

- A booking instance has been made

Postconditions:

- Booking instance removed

Contract CO3: Add New Hotel

Operation: AddNewHotel(string name, string city, string address, HotelChainIdentifier hotelChainIdentifier)

Preconditions:

- Relevant Hotel-chain instance already exists

Postconditions:

- Hotel instance is initialized
- Hotel associated with hotel-chain

Contract CO4: Find Available Hotels

Operation: FindAvailableHotels(DateTime startDate, DateTime endDate, int numRooms, string city)

Preconditions:

- A booking agent is logged in at the web-client

Postconditions:

SLA (part 1)

Introduction

A service level agreement is a contract made between a customer and a service provider. The SLA agreement is used to set the level of service the customer is expecting from the service provider. We did not exchange an SLA agreement with another group therefore we have made our own SLA. Which can be used by service providers. The SLA agreement consist of different types of metrics. These metrics are used to see if the system is upholding the standards we as a customer and service provider agreed upon.

SLA Agreement

Uptime/availability

This metric is used by the customer to decide how much uptime the system should have. This metric is often a percentage value. Eg. the customer could demand that the uptime of the system should be 95 % for the year, this value is also considered based on the fact that a customer, should also take into consideration where and when maintenance is needed for system/end-product. This consideration should also comply with another SLA metric; failure frequency.

Taking our own product and service into consideration, we could create a service contract agreement between us as the service provider and a potential customer, using the above mentioned uptime value of 95% of our Hotel Booking System. If the case is that we do not comply and results show that our service did not cover those 95% both parties agreed on, we could for example settle on a discount of our service for our customer, to counter those periods of downtime.

Mean response time

This metric is used to set a value for the mean response time of the system. So when you make a request to the system has to respond within this time period.

Now taking this SLA metric parameter and comparing it to our Logging setup through Application Insights in Azure, that provides us with this data, we can see that based on recent datalog our mean response time is 48.81ms for the server response, with 26 requests being pushed through.

Mean time to recover

This metric is used to specify the meantime a system can take to recover, when it has been out of service. This metric is very important because if the mean recovery time is greater than the value agreed upon, it might have major consequences for the customer.

This was mainly handled by other group's projects, where they had to ensure that these specific SLA guidelines were being followed.

Having said that, there's no specific implementation that carries out restarting or handling errors in the system if it gets manipulated into causing downtime via. a crash or error.

Failure frequency

This metric is a percentage between 0 and 100 and is used to specify how often a failure is allowed to occur in the system. Taking relevant information from our Risk Matrix we can see that there are still some difficulties keeping the user-experience fault-proof. Most notably regarding setting up accounts, changing passport information and some missing front-end testing might cause the system to abruptly cause a bug or in worst case crash. Seeing as we're aware of this, and have taken these issues into consideration for future revisions of our service which can resolve these issues, the failure frequency should drop by a large amount. In that case a new service contract is likely to take place, between us as the service providers and our customers.

Description of tools and experiences (part 2)

Continuous integration(azure pipelines)

As a part of the project we were supposed to use Continuous Integration.

For the project we have been assigned to use CI so we will start with telling about what CI in general is, and later on we'll explain how we have implemented into our project and for what reason.

What is Continuous Integration

Atlassian defines Continuous Integration as follows:

“Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project¹”

So what we can understand from the definition is that every time a developer commits his/her code, the buildserver takes the code that has been made and integrates all the parts into the master branch. This is done every time continuously.

This is a common practice in large projects for the very reason that it reduces integration errors which in the end saves time and money in finding the errors and fixing them.

Although there is much to gain with CI there are also some disadvantages. One of them is that it is not easy having to set CI up.

With this we mean that if for example you are working for a larger company it can be quite difficult and therefore take much time and cost a lot having CI integrated in a project which is not favorable for most firms and therefore choose not to use CI.

There are many tools to choose from in regards the use of CI. One of the more popular ones is Jenkins , Travis CI , Azure Devops and so forth.

What is Continuous Deployment

Atlassian defines Continuous Deployment as follows:

“Continuous Deployment (CD) is a software release process that uses automated testing to validate if changes to a codebase are correct and stable for immediate autonomous deployment to a production environment.²”

When using Continuous Deployment we are always sure about the fact that our latest release doesn't contain any faulty code. Hence, if the code is not executable, the code wouldn't be released until the problems are solved.

¹ <https://www.atlassian.com/continuous-delivery/continuous-integration>

² <https://www.atlassian.com/continuous-delivery/continuous-deployment>

How we implemented CI and CD in our project and why

For continuous integration / deployment we have used Azure DevOps in our project. We have setup a buildserver and have configured it to always deploy when we push our commits to Github which will inturn have it pushed to Azure and have it build up there. The reason why we used Azure DevOps is because our project is written in C# which goes hand in hand with Azure. This saves us a lot of time having to deploy and we can be more productive. As our C# projects are more comparable with Azure. It's a lot easier having to configurer settings so that we will in theory always have the newest release ready , as opposed to using other services.

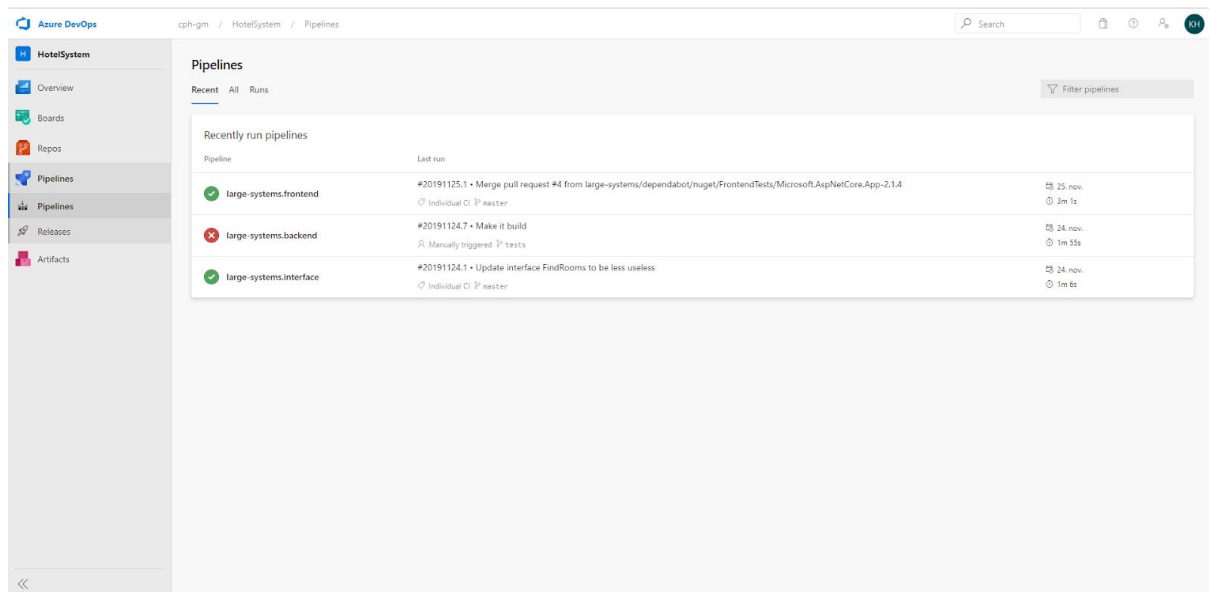


Figure 1

As we can see in Figure 1 that we have set up for pipelining our different projects which is for our three projects namely the frontend, backend and interface/contract where tests have been added in our two projects except the interface project.

Artifactory(azure artifacts)

Overview

We will need to consider and take a closer look at software repository and NuGet before we can continue to talk about Package Management with Azure Artifact

A **software repository**, or “repo” for short, is a storage location where software packages are stored and retrieved. A software repository is typically managed by source control or repository managers. Package Managers allow for installing and updating the repositories (sometimes called “packages”) versus having to manually do this.

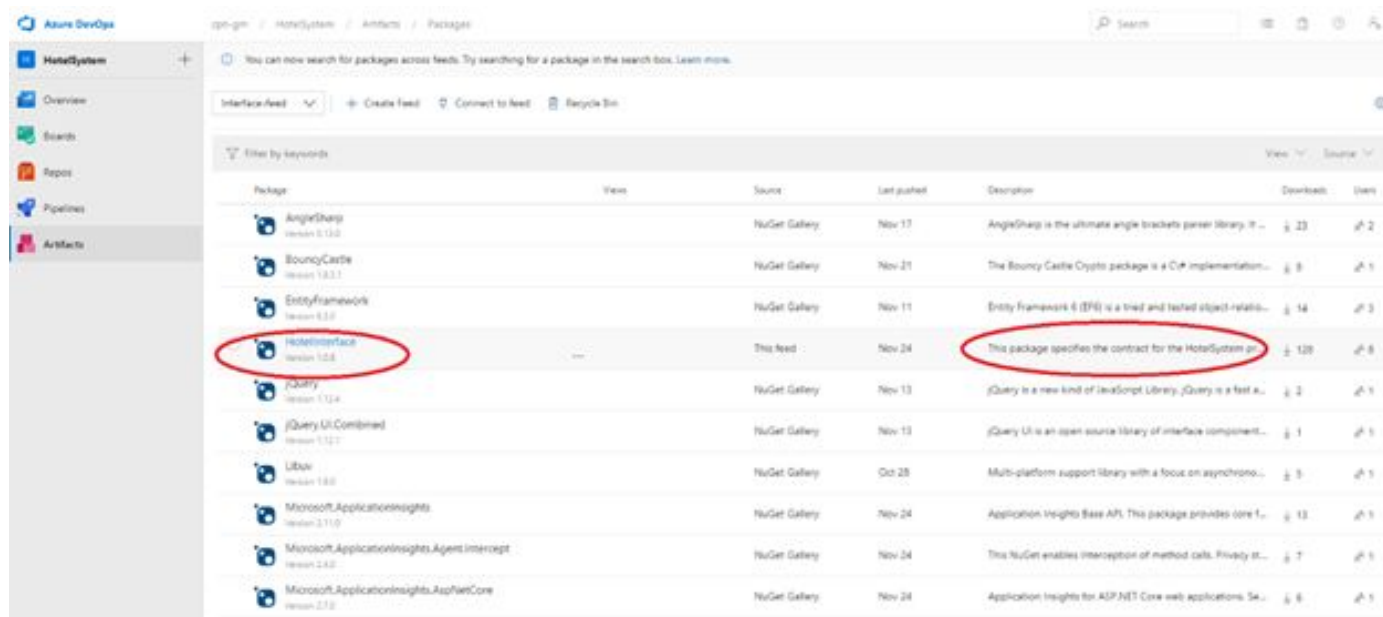
NuGet in the other hand is distributed as a [Visual Studio](#) extension. Starting with Visual Studio, NuGet comes pre-installed by default. NuGet is also integrated with [SharpDevelop](#). NuGet can also be used from the command line and automated with scripts.

Azure Artifacts is an extension to Azure DevOps Services and Azure DevOps Server.

With Azure Artifacts, you can create and share Maven, npm, and NuGet package feeds from public and private sources with teams of any size. You can add fully integrated package management to your continuous integration/continuous delivery (CI/CD) pipelines with a single click. We have used Azure Artifacts to manage our NuGet packages (feed) in our HotelSystem.

There are numerous task tasks that could be done with Artifacts, such as Creating and connecting to a feed, Creating and publishing a NuGet package, importing a NuGet package or Updating a NuGet get package.

The figure below shows some of our created NuGet packages.



The screenshot shows the Azure DevOps Artifacts Packages page. The left sidebar contains navigation links: Overview, Boards, Repos, Pipelines, and Artifacts. The main content area displays a list of packages under the 'Interface-feed'. The 'HotelsInterface' package, version 1.0.0, is highlighted with a red circle. Its description, 'This package specifies the contract for the HotelsSystem project', is also circled in red.

Package	Version	Source	Last pushed	Description	Downloads	Users
AngleSharp	Version 0.13.0	NuGet Gallery	Nov 17	AngleSharp is the ultimate angle brackets parser library. It ...	23	2
BouncyCastle	Version 1.6.1	NuGet Gallery	Nov 21	The Bouncy Castle Crypto package is a C# implementation...	8	1
EntityFramework	Version 6.2.0	NuGet Gallery	Nov 11	Entity Framework 6 (EF6) is a tried and tested object-relatio...	14	3
HotelsInterface	Version 1.0.0	This feed	Nov 24	This package specifies the contract for the HotelsSystem pr...	129	8
jQuery	Version 1.12.4	NuGet Gallery	Nov 13	jQuery is a new kind of JavaScript library. jQuery is a fast a...	2	1
jQuery.UI.Combined	Version 1.12.1	NuGet Gallery	Nov 13	jQuery UI is an open source library of interface component...	1	1
Libuv	Version 1.0.0	NuGet Gallery	Oct 28	Multi-platform support library with a focus on asynchrono...	5	1
Microsoft.ApplicationInsights	Version 2.11.0	NuGet Gallery	Nov 24	Application Insights Base API. This package provides core f...	13	1
Microsoft.ApplicationInsights.Agent.Intercept	Version 2.0.0	NuGet Gallery	Nov 24	This NuGet enables interception of method calls. Privacy st...	7	1
Microsoft.ApplicationInsights.AspNetCore	Version 2.7.0	NuGet Gallery	Nov 24	Application Insights for ASP.NET Core web applications. Se...	6	1

Using the artifacts through NuGet

Consume NuGet packages can be done in Visual Studio

The easiest way is to follow [this](#) guide. The problem with it is that then it will only be added for **you**. If you were to create a new project like this and then push it to Github, everyone using it would have to follow the same guide. There is an alternative.

Consume packages

You can now discover and use packages in this feed. To add a package reference to a project:

1. Find your project in Solution Explorer.
2. Right-click **References**.
3. Select **Manage NuGet Packages**.
4. In the **Package source** drop-down list, select your feed.
5. Look for your package in the list.

If you're using [upstream sources](#), package versions in the upstream source that haven't yet been saved into your feed (by using them at least once) won't appear in the NuGet Package Manager search. To install these packages:

1. On the upstream source (for example, nuget.org), copy the Install-Package command.
2. In Visual Studio, open the Package Manager Console from **Tools > NuGet Package Manager**.
3. Paste the Install-Package command into the Package Manager Console and run it.

The better way

Create a file called NuGet.Config in the projects folder. The file should contain the following code, the value(it says **CHANGE ME** now) must be changed to the **package source URL** found on Azure DevOps artefacts section under the feed called **Interface feed**.

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>

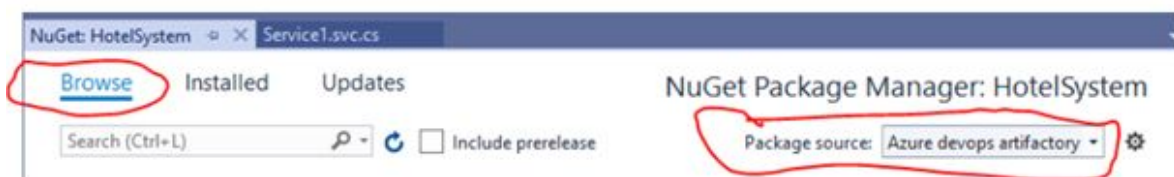
  <packageSources>

    <add key="Azure devops artifactory" value="CHANGE ME" protocolVersion="3" />

  </packageSources>

</configuration>
```

After creating this file, close the solution and re-open it(if you had it open). After this you should be able to open the project and selected **Manage NuGet packages...** as you normally would. Make sure you select the correct **Package source** in the NuGet interface, see picture below.



Hosting(azure)

Azure Devops is a cloud service that hosts services on a global network, also called the cloud. More people are starting to use the cloud to host applications and databases since it provides a better overview to manage all parts of the services. Additionally, services can run continually without the downtime that can occur if services are hosted on a computer.

Since our group are mostly coding in c#, we feel like Azure Devops would be the perfect cloud to host our application because c# and Azure Devops are both created by Microsoft, so it is the best cloud service to host most of our stuff. Some of us have experience with hosting applications on Azure Devops from previous projects and leisure projects, so we anticipate no hosting problems. We have chosen to host both our back-end and front-end applications in an app service on Azure. We have chosen an sql database on Azure to host our database.

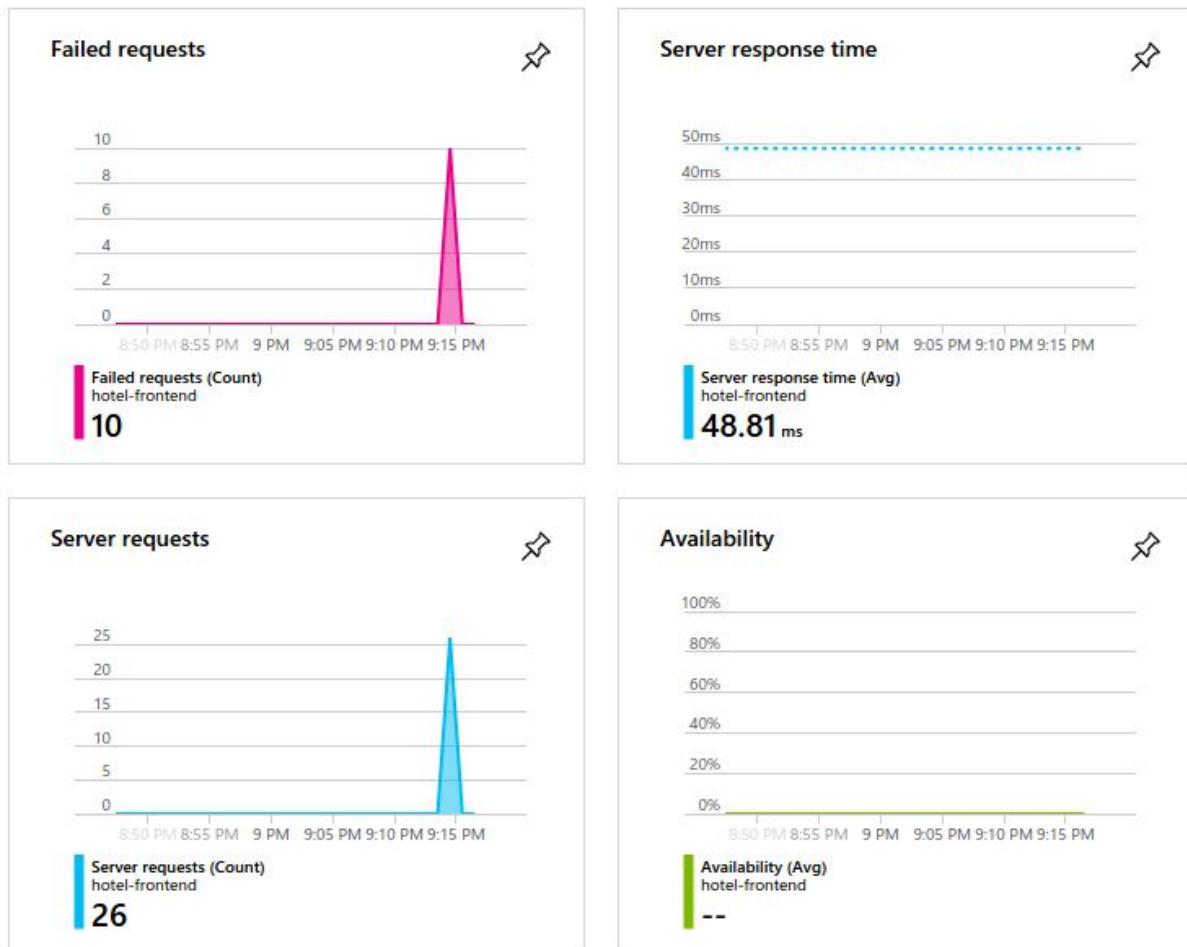
We also thought it advantageous to use Azure Devops' other services, for example, having continuous integration for our applications as one of our development practices since it enables us to code into a shared repository frequently, where each commit can then be verified by an automated build and automated tests. This makes it easier to detect and locate errors more quickly in the application.

Logging set up

Microsoft Azure collects application logs and provides services to analyze and monitor them. The means to gather these logs differs depending on the application itself, but the presentation is the same. Azure is centered around resources, a resource can be anything: a hosted application, a security group or log data. The specific resource to analyze logs is called "Application Insights". While in our case this resource is only used to monitor application error rates live and browse log data it offers many more features such as Performance information, Smart Detection and others.

Out of the box, it all already provides some valuable metrics for web application: number of failed requests, response time, the overall number of requests and availability. All this data is already there with the default web application hosting through Azure. There is also an option to build a custom dashboard, with loads of metrics to choose from, including custom ones that can be added by integrating Application Insights into the applications manually.

Show data for last: **30 minutes** 1 hour 6 hours 12 hours 1 day 3 days 7 days 30 days



For historical log data analysis, Azures Application Insights provide a view that supports a special query language to query, filter, sort and transform log entries. For example, the query “exceptions | [order by](#) timestamp desc nulls first” will return all exceptions, sorted in descending order of their timestamp, with null values coming first.

The more difficult part is configuring an application to send exception, and other log information to Azure to display. The exact process differs from application to application, depending on its type and targeted framework but at the simplest form it is just installing a few packages using intuitive Visual Studio interface.

```

app.UseExceptionHandler(builder =>
{
    builder.Run(async context =>
    {
        var contextFeature = context.Features.Get<IExceptionHandlerFeature>();
        if (contextFeature != null)
        {
            var client = new TelemetryClient();
            client.TrackException(contextFeature.Error);
            client.Flush();
        }
    });
});

```

For our projects front-end, an ASP.NET MVC project targeting .NET Core, basic data is available by calling “UseApplicationInsights” when configuring a WebHost. The installed packages provide a class allowing anything to be posted to Application Insights. We use this class, in the applications global exception handler to send each exception to Azure as shown in the code below.

For the back-end project, which uses older Windows Communication Foundation(WCF) technology and targets the .NET Framework, the process is slightly more verbose. First of all, all relevant Application Insights packages need to be installed. Then, two HTTP modules need to be added, both referencing classes from the installed packages. This will allow them to hook into the web configuration and automatically send some statistics to Azure.

```

<httpModules>
  <add
    name="TelemetryCorrelationHttpModule"
    type="Microsoft.AspNet.TelemetryCorrelation.TelemetryCorrelationHttpModule, Microsoft.AspNet.TelemetryCorrelation"
  />
  <add
    name="ApplicationInsightsWebTracking"
    type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule, Microsoft.AI.Web"
  />
</httpModules>
<add name="errorLogging"
  type="HotelSystem.HotelBehaviorExtension, HotelSystem, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

```

To go beyond basic data, we define a behavior extension which we then add to all service behaviors for our WCF services.

```

private readonly TelemetryClient client = new TelemetryClient();

public bool HandleError(System.Exception error)
{
    client.TrackException(error);
    return false;
}

```

All this extension does is register a new service behavior that goes through all Channel Dispatchers and adds our custom error handler. A ChannelDispatcher essentially just maps service URIs to EndpointDispatchers which handle incoming WCF messages. This allows us to catch exceptions from all the processing that our application does. Our custom error handler(shown below), similarly to the code in the front-end, uses the TelemetryClient to send the exception details to Application Insights and then returns false, telling the framework that

it was unable to handle the exception and it should pass it on to the next handler, which in our case just converts it to an exception response.

All of the logging mentioned until now has only been intended for post-mortem analysis. That is, to gather data about when things go wrong. Another option that we considered, is audit logs: logging information about the program flow even when things are going as expected. Not only does this provide valuable information about the state and actions taken in the post-mortem analysis, it can also provide information to satisfy regulations and so on. Due to time constraints and the lack of need for it, our solution contains no audit logs. Even though we did not implement it, Application Insights and Azure does support it and the same TelemetryClient seen before, can be used to send free-form strings to Azure.

Problems with CD and Azure

We attempted to set up continuous deployment. The intent was that whenever someone pushed to the main branch of the git repository on Github, it would automatically deploy the newest version to the server.

We intended to do that using Azure DevOps. We started by setting up the deployment process, and actually did get it working.

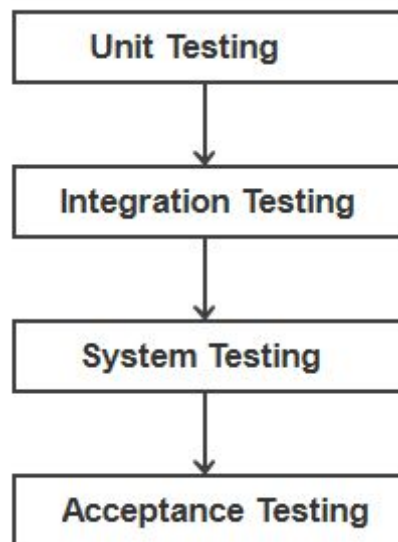
However, due to the way the Azure account is set up, we only have a trial period, which expired. Therefore continuous deployment stopped working when the trial period ran out.

Tools used for testing

Overview

In this section we will be talking about the tests created based on the image below, and the tools we have used in order to conduct these tests.

Conducting our Unit- Integration tests we have been using Nunit and Xunit framework in C#.



In the figure above we can see the timeline of testing, which we will explain briefly in this paragraph. We have based the creation of tests on the image since for example having Unit tests can be crucial when making integration tests and so on.

Unit Testing

When conducting unit testing, we are looking at specific parts/units of a program, to be able to determine if they are fit to use.

Unit testing is usually the first test that is created in an application, as seen in the image in the overview section. The reasoning behind it is, because unit tests are necessary to conduct the other types of tests.

Unit tests in the project are created on the backend part, where we are using different types of assertions in order to make sure that our methods run and return a value that we expect.

```
[Test]
public void FindBookingsTest()
{
    Assert.Greater(0, hotelsystem.FindBookings(12345678).Count);
}

[Test]
public void FindBookingByIdTest()
{
    Assert.DoesNotThrow(() => hotelsystem.FindBookingById(new BookingIdentifier(1)));
}

[Test]
public void FindRoomsTest()
{
    // DateTime is in the format year, month, day, hour, minute, second
    Assert.Greater(0, hotelsystem.FindRooms(new System.DateTime(2019, 3, 1, 7, 0, 0), new HotelIdentifier(1, "Lüksurum").Count);
}
```

Integration Testing

Integration testing is usually made after the Unit Testing, in order to provide an overview if our components work together as expected.

It proves useful when we need to check all the implemented modules and logic is produce the expected results and render the correct values.

We have conducted integration testing in the front end part of the project for the purpose of assuring us that the search form for hotels is showing the expected results. Since for this search to work, the search needs to use components and various modules.

The integration test can be seen in the following [link](#).

System Testing

System testing generally is done by a testing professional who is handed a “finished” product, then conducts tests. In our case we handled the system testing internally, and our main focus points were:

- Usability Testing
- Load Testing
- Functional Testing

While carrying out **Usability Testing** we were focusing on user experience, specifically how difficult or easy users find the applications usage.

Load Testing was an important part of our process, since we aimed to see how the software would perform in real-life cases. !!!!!!!!!!(MAYBE? CAN BE DELETED)!!!!!!

In **Functional Testing** our focus was to identify if the existing functionality is lacking any functions, in this phase we have created suggestions for what we could improve in regards to how the software is behaving.

Acceptance Testing

Acceptance testing is also known as Beta testing, where the software is being tested by the end-user.

Acceptance testing is still yet to be done, since we need to release it to external testers in order to obtain valuable feedback.

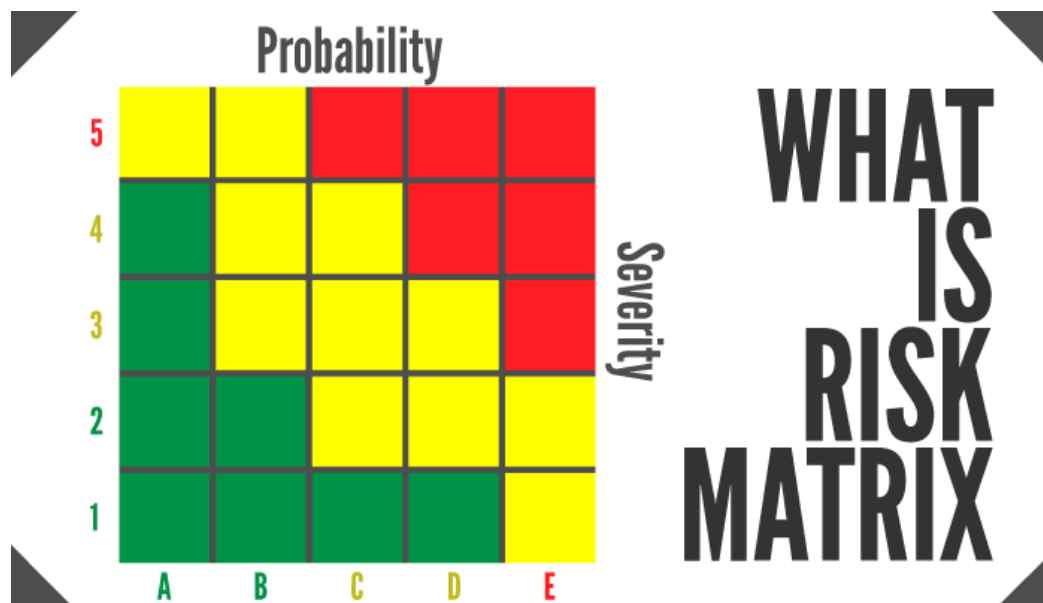
Risk Matrix

Given our time window and having to focus on implementing functionalities and making sure our backend is compatible with the frontend, we did not have a lot of time to look into risk matrix. However, we can talk about future work and how we would include risk matrix into our project.

If we talk in general, risk matrix is a matrix which is used during risk assessment to determine risk level by using probability and severity to quantify the scope of a real or hypothetical safety scenario. We can break the quantification into three categories:

- 1) Acceptable risk
- 2) Unacceptable risk
- 3) Risk that is as low as possible

The way companies differentiate between the levels of the risk are usually by attaching colours to each of the categories and then in the end your risk matrix might look something like this:



The matrices are broken into the grid, as can be seen on the picture and the grid is used to assign a value, in this case a number, which is a combination of probability x severity and it is used as a representation of the scope.

When it comes to Software Development, we do not focus on the risk matrix in particular but there is this approach which is called Risk Management and in particular developers are focusing on making the Risk Management Plan. The benefit of the plan is to mitigate and contain bugs/dysfunctionalities to ensure project success. To further explain, the Software

Risk Management includes the identification and classification of various risks, such as technical, programmatic or process ones.

There are different ways of making your Risk Management plan but for agile development and small groups of devs(as is our case) the best approach would be to make a Risk Register.

Risk Register usually consists of these properties:

- 1) Description of risk
- 2) Recognition Date
- 3) Probability of occurrence
- 4) Status
- 5) Severity
- 6) Loss Size
- 7) Risk Exposure
- 8) Priority

And many more.

If we were to apply the Risk Register to our own project, focusing on implementation of functionalities as well as the deployment it would look something like this:

Risk Description	Probability of occurrence	Status	Severity
Insufficient time to set up CD on azure due to having issues with setting up accounts	90%	Not resolved	High
Insufficient Front end testing missing which could lead to system breakdown in case of unexpected bug	22%	Resolved to an extent	Low
Not handling a case where user unexpectedly changed passport number which is tied to a booking	15%	Not resolved	Medium-low
Nuget packages might cause errors if their respective versions are not compatible with each other	9%	Not Resolved	Low

Of course, we could have included many other risks into the table with proper preparation and planning but this gives us an overview of how we would approach it in the future.

Reflection & Conclusion

In retrospective in regards to the project and the outcome of the project, we believe that there was a slight lack of management which caused confusion and difficulties in collaboration.

In a real-life situation there are representatives/managers that have an overview of the tasks, and can assign people to them. In our case since there was no designated person to handle managing the project, the amount of work was not evenly distributed, meaning that some people had to do more, while others did not have the opportunity to be as active. We believe that this issue could have been solved, if there would have been a hierarchy/roles assigned by the instructors of the class, resulting in possibly better teamwork, especially with a group as large as ours.

Furthermore, we feel like we gained some valuable experience by working in a large group, we have learned about the challenges and opportunities of working on large systems. We have been introduced to new technologies and libraries that can be beneficial in a real-life work environment, and the challenges we can expect when working at a company in a large team.

On the other hand we have experienced some external issues with the organization, that stopped us from completing our tasks, as an example, issues with DevOps on Azure, which was quite unexpected, uncomfortable and disappointing.

In conclusion, given our circumstances, we've taken steps in the right direction, but also faced a wall during some moments in development, where we've both experienced success in our tasks, but also ran into major issues that were unexpected as mentioned. However we've ended up with a product that carries out the service we intended from the beginning. It's not faultless however as there are loopholes which might interrupt the service causing it to act abnormally or crashing, these are mainly due to our complications we have encountered when using Azure.

In the end we feel like we have a solid foundation for further implementations including features and fixes and that the challenge was overall very welcome in our group, regardless of our many suggestions and differences in how to solve and carry out this task.