Western Norway
University of
Applied Sciences

BACHELOR'S THESIS

Utvikling av Sandkasse for API-Testing
Creating a Sandbox for API Testing

Are Dæhlen
Christopher Ishaque Jamil
Anders Kvamsøe

Bachelor, Computer Science/ Engineering
Department of Computer Science, Electrical Engineering and Mathematical Sciences
Faculty of Engineering and Science
02.06.2020

Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Utvikling av sandkasse for API-Testing Creating a Sandbox for API Testing	*Dato:* 02.06.2020
Forfattere: Are Dæhlen Christopher Ishaque Jamil Anders Kvamsøe	*Antall sider u/vedlegg:* 32 *Antall sider m/ vedlegg:* 37
Studieretning: Dataingeniør / Informasjonsteknologi	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Violet Ka I Pun	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: ZData AS	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Kjetil Tollevsen	*Telefon:* 93637637

Sammendrag:
Målet med denne oppgaven har vært å lage en sandkasse for testing av APIer som oppdragsgiver kan bruke i sin utviklerportal. Det har vært et sentralt fokus på å ta i bruk eksterne teknologier for å kunne øke kvaliteten på det ferdige prosjektet. Produktet vi har utviklet og beskrevet i denne oppgaven, er en løsning ZData kan fortsette utvikling på og utvide det til å passe alle deres spesifikasjoner.

Summary:
The goal of this assignment has been to create a sandbox for testing APIs that ZData can use in their developer portal. There has been a central focus on using external technologies to improve the quality of the completed project. The product we have developed and described in this thesis, is a solution ZData can continue to develop and expand it to fit all its specifications.

Stikkord:

ASP.NET Core	Sandbox	API

Preface

Through this document we describe the development of our project and bachelor thesis: “Creating a sandbox for API testing.” The project was done by Are Dæhlen, Christopher Ishaque Jamil and Anders Kvamsøe.

Thank you to ZData for giving us the opportunity to work on this interesting project. Special thanks to Henrik and Kjetil at ZData, who were our primary contacts. They were a tremendous help to us throughout development of the project.

Also, a big thanks to our HVL supervisor Violet Ka I Pun for guiding us on the project and giving us invaluable feedback on our report.

Table of contents

Table of figures

1 Introduction

1.1 Motivation and goal

1.1.1 Project owner's motivation and goal

The scope of this project is to develop a sandbox for testing the APIs delivered by ZData AS (henceforth referred to as ZData). The motivation behind this project for ZData is to make their APIs easily accessible for testing by interested clients, as well as a useful tool where both developers and business managers can judge if their application would benefit from such an API. The developer sandbox will work as an effective way to advertise their services to a larger audience, as well as putting pressure on their competitors. The goal of this project will be to have an easily accessible, accurate and user-friendly developer sandbox in place to work as an efficient marketing tool for ZData's services.

1.1.2 Project group's motivation and goal

Our motivation for this project was largely rooted in the interest in learning more about the different techniques and technologies that the IT industry uses for development. In addition, we were also motivated by the opportunity to receive guidance from professionals in the field. The goal of this project was to deliver a high-quality product as a testament to our personal skill and ability, and as a referenceable project for future use. This also was a large expansion of competence in the IT-development field as there were many new things to learn. During the implementations of this project, we gained experience in a large number of new technologies and their application, in addition to the ability to practice collaboration in a professional setting. To help us achieve these goals we cooperated closely with ZData and our advisor from HVL.

1.2 Context

ZData is a broker that acts as the intermediary between banks and their commercial clients. Their client base consists of companies such as Unicef, Toyota, Plantasjen, etc. Their aim is to streamline the experience of connecting payments and financial services of these clients to various banks. All of these clients rely on the ability to invoice their own customers, pay their bills, transfer funds internally between departments. Doing this efficiently and automatically decreases the running costs of the business, and increases profit margins. Efficiency and automation are key principles of the services offered by ZData, which makes them attractive to their clients.

ZData is currently experiencing tremendous growth as a result of global trends in the fintech industry, having doubled their employee count since last year. Automation and open-banking are making software brokers like ZData extremely relevant on the global stage. The creation of a developer sandbox in which prospective clients can test and verify various aspects of ZData's

software services before committing themselves to establish a collaboration, is a way to ensure quality of the product for clients.
ZData is not the first business to construct a developer portal. Some other competing businesses already offer their own sandboxes. These sandboxes range widely from design and capability, with most of them lacking in some way or another. For instance, one has a very robust interface which is easy to use and understand, while it lacks some of the essential functionality that is needed for a complete testing environment. Another competitor has the exact opposite issue, where all functionality is present, but the overall design is not very user friendly. ZData wants to be competitive and offer their own developer sandbox that is more user friendly and offers a greater ease-of-use experience with all of the necessary tools to raise their attractiveness to potential clients.

1.3 Limitations

The main limitations the project faced were related to time constraints and a lack of experience. Many of the technologies and techniques required to complete this project were new to us. Technologies such as Azure DevOps, Docker and Mailgun are tools we had never worked with before. Developing in sprints would also be a new experience. Therefore, a period of adjustment, where things go more slowly and more errors and mistakes are made, was to be expected. The main coding portion of the project was estimated to take approximately six weeks, which is relatively short for the estimated workload.
ZData believed the amount of time was sufficient, evidenced by the additional tasks the company set for us if the project was finished ahead of schedule. All the technologies and techniques made available to us were meant to make the project develop more easily and quickly, so as we became acquainted with them, development would speed up exponentially in comparison to working without them.

1.4 Resources

ZData is offering their software and expertise. We used our own computers for development. ZData provided software including Azure DevOps, MailGun, API etc. There was also a senior software engineer available for questions and assistance throughout the development of the project. Workspace was supplied by both HVL and ZData. Development of the project was done in sprints and everything was developed in Visual Studio 2019.

2 Project description

2.1 Practical background

2.1.1 Project owner

ZData is a Norwegian software enterprise established in 1990. They specialize in all types of electronic banking services for commercial use. The main business model of ZData is to increase profit margins of their clients by facilitating and automating their clients in and out payment routines, as well as giving them a better overview and control over their own liquidity and finances. Because of the EU directive PSD2 (EU, 2015) all banks must now open their systems. It means that third parties will be able to access customer payments accounts, and therefore offer account information and payment information services. ZData intends to offer more and better services to their clients as a direct consequence of this directive.
ZData has APIs developed for connecting to all banks in the Nordic countries, and a number of banks in Europe. Their clients range from small local businesses, to large, multinational businesses, numbering over 5000 in total.
Our contact person is located in their main office in Bergen. (ZData AS, 2020)

Figure 2.1: *ZData logo*

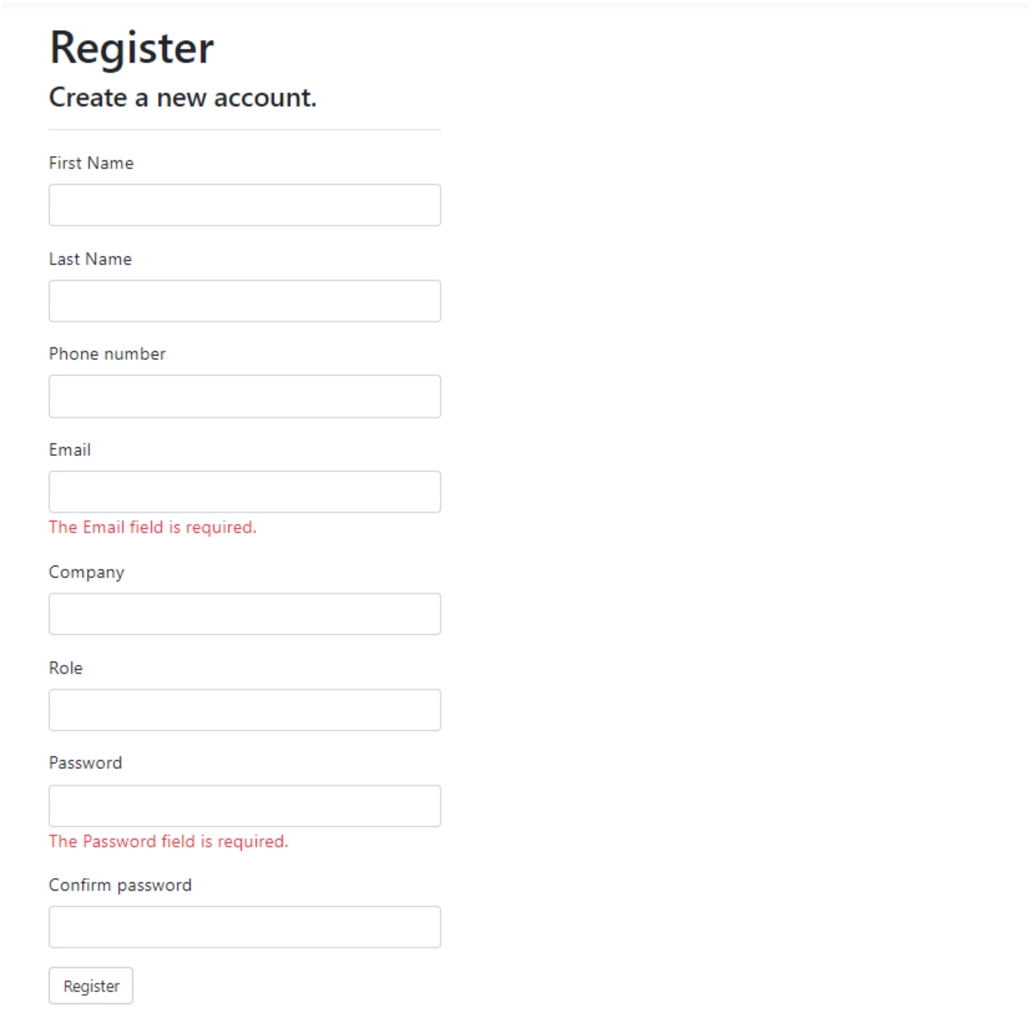
2.1.2 Initial requirements specification

Specified by ZData, the primary requirement specifications for our project were to create a sandbox used to automate testing, by generating mockup data, of their existing banking related APIs. This sandbox is planned to be a part of their developer environment, and to be one of the main ways to attract new customers with easy implementation and testing of their services. There is also a requirement for a simple login system for the development environment that can facilitate the creation and authentication of users.

The primary goal of this project is to provide any interested customer with a convenient and simple way to test their APIs, regardless of their technical background. The sandbox should be understandable and user-friendly for both developers and business strategists.

2.2 Simple application flow

Our application has a relatively easy and understandable flow, starting with a registration system. Here the user will enter information about themselves, as well as an email, phone number and password. Figure 2.2 illustrates how the registration page looks.



Register

Create a new account.

First Name

Last Name

Phone number

Email

The Email field is required.

Company

Role

Password

The Password field is required.

Confirm password

Figure 2.2*: The registration page of the application*

Once a user has registered an account, an email will be sent to the submitted email address with a confirmation link to validate the account. A login page is also available for users with an existing account. After logging in or verifying their email the user is taken to the dashboard page, where they can either select a sandbox that they have already created or create a new one. See

Figure 2.3 for a visual representation of the dashboard page with one sandbox already created.

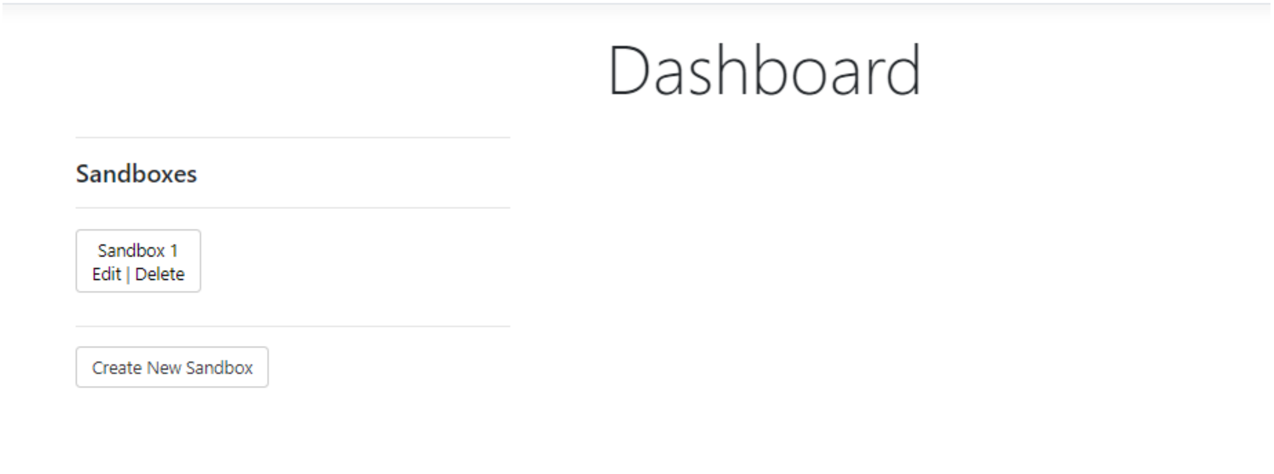
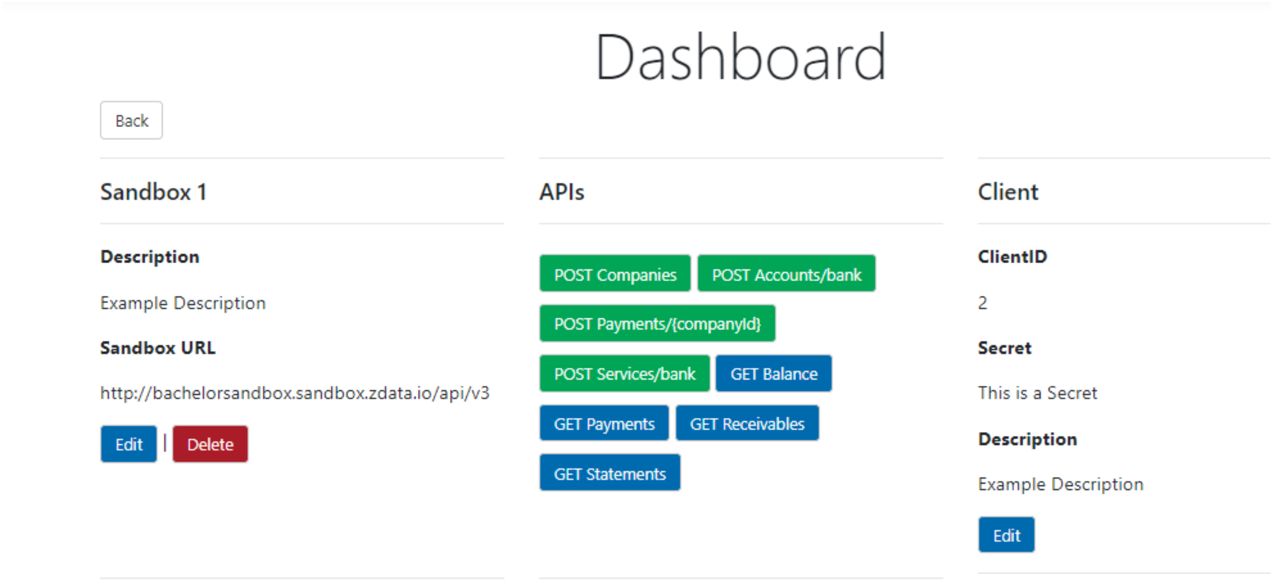


***Figure 2.3**: An example user's sandbox dashboard with one sandbox created.*

When clicking the Create New Sandbox button the user is taken to another page where they are asked to enter relevant information such as a sandbox name and a client id. When submitting this information, a new sandbox is created and can be viewed from the dashboard alongside other existing sandboxes that the user has created. The user also has the option to edit or delete sandboxes they have created from the dashboard.

When the user clicks on the name of a sandbox they have created they are taken to that sandbox's dashboard. From here all of ZData's different APIs can be selected and tested (See Figure 2.4).


Dashboard

Back

Sandbox 1

Description

Example Description

Sandbox URL

http://bachelorsandbox.sandbox.zdata.io/api/v3

Edit | Delete

APIs

POST Companies
POST Accounts/bank
POST Payments/{companyId}
POST Services/bank
GET Balance
GET Payments
GET Receivables
GET Statements

Client

ClientID

2

Secret

This is a Secret

Description

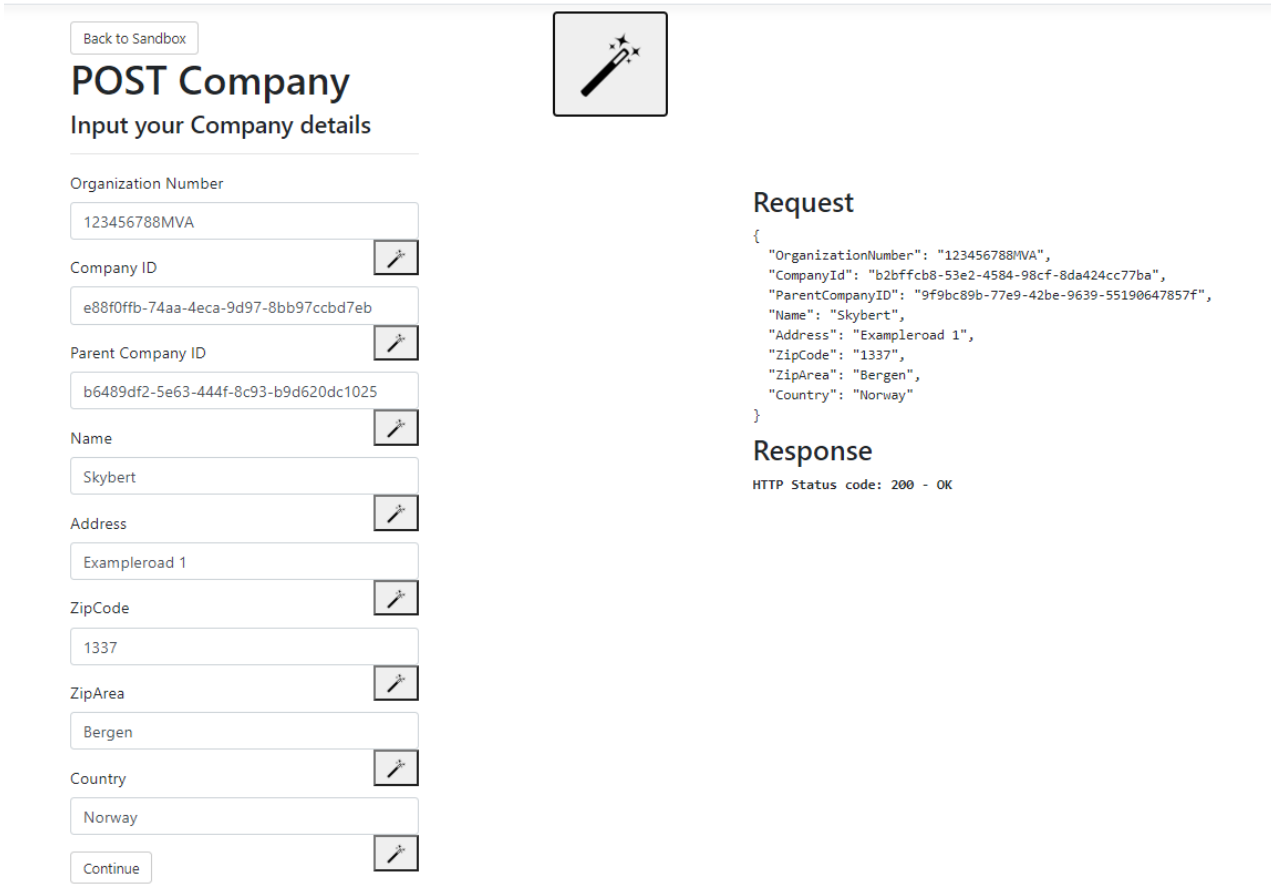
Example Description

Edit

***Figure 2.4**: An individual sandbox's dashboard with an overview of all available APIs*

When selecting an API from a sandbox's dashboard the user is sent to a page where they will need to either fill in or automatically generate the necessary information that the API requires to make its call. More information about how this data is generated can be found in section 4.2.8.

Below in Figure 2.5 an example of a call to the POST Companies API using automatically generated data with a response and the request details visible in a json format.



POST Company

Input your Company details

Organization Number	123456788MVA
Company ID	e88f0ffb-74aa-4eca-9d97-8bb97ccbd7eb
Parent Company ID	b6489df2-5e63-444f-8c93-b9d620dc1025
Name	Skybert
Address	Exampleroad 1
ZipCode	1337
ZipArea	Bergen
Country	Norway

Request

```
{
  "OrganizationNumber": "123456788MVA",
  "CompanyId": "b2bffcb8-53e2-4584-98cf-8da424cc77ba",
  "ParentCompanyID": "9f9bc89b-77e9-42be-9639-55190647857f",
  "Name": "Skybert",
  "Address": "Exampleroad 1",
  "ZipCode": "1337",
  "ZipArea": "Bergen",
  "Country": "Norway"
}
```

Response

```
HTTP Status code: 200 - OK
```

Figure 2.5: *Test of the POST Company API with automatically generated data.*

3 PROJECT DESIGN

3.1 Possible approaches

There are a few different options available for web development, both modern and classic, and selecting the appropriate approach is important. All options have their pros and cons, and individual needs and requirements should influence the decision. We considered several options before landing on what we found to be the most appropriate method of development for this project.

3.1.1 Development using .NET Core

The approach of developing the solution in .NET Core would be a more challenging process for our group, seeing as we had less experience with this than other possible solutions. Using .NET Core would, however, make the development of the sandbox application smoother, as this is what ZData has used in their other solutions. .NET implements the Common Type System (CTS) (Microsoft, 2016a), meaning any implementation is language independent. This means that a programmer can compile code in languages such as C++, Java or Visual basic, and with the proper set of supported types this code can be used in a .NET implementation. This feature is an important factor when selecting a language to build applications with, as it will have the benefit of making the application easily scalable with other required features that may not be achievable with .NET. The framework also comes with the benefit of being easily interfaceable with Windows or other Microsoft systems, seeing as the framework is built by Microsoft themselves
Some disadvantages of .NET Core include slower performance due to the management of code behind the scenes, limited object relational(OR) support as it comes only with Entity Framework, and it does not come with multi-platform support from Microsoft (Rongala, 2015).
A personal advantage for us with using .NET is that we could easily seek guidance from our internal advisors at ZData. Another advantage was that it will make it easier to connect our solution to their website. Furthermore, developing the solution in .NET will make the project easier to maintain for ZData in the future. If this project was made in another language their internal developers have less experience with, maintaining our solution may be difficult. (See section 3.2.2 for more information.).

3.1.2 Development using Java Servlets and JavaScript

Java is the programming language that is most commonly used in the world (Oracle, 2019). Because of this, the Java community is bigger than Microsofts, offering a larger user base with relevant experience. Java is very OS independent, allowing applications to be deployed on any OS easily. It is also backwards compatible, which facilitates migrating a project between Java versions. The versatility of Java enables it to be used for development in a wide range of industries. Java has a very efficient GUI, making it attractive for user clients. However, Java's great versatility comes at a price. As described by Rongala (2015) it is considered slower than many other languages, and generally requires more memory for optimal function. He also describes that it is more susceptible to security breaches because it is platform independent. Java does support most of the APIs the project owner requires, either through JavaScript implementation (Mozilla, 2020) or the back-end of the Java Servlets (Oracle, 2019).

3.1.3 Discussion of alternative approaches

Although using Java servlets and JavaScript for this project would be perfectly suitable, the project owner has decided that this project will be developed in .NET Core. This will also bring certain advantages as explained above, but the learning process may take a longer time. We do feel comfortable working in .NET, as the internal advisors will be able to guide us, and we believe that this will produce the best end result.

We chose .NET Core as the approach to implement the project mainly because it allows integrating the sandbox we developed in this project seamlessly into ZDatas development environment, as it implements the same framework. Although developing this project in a different environment was possible, it would be more difficult. It would be especially difficult to make the project communicate efficiently with ZData's other services, and we would therefore simply be making things more difficult for ourselves by developing in another environment.

3.2 Selection of tools and programming languages

3.2.1 Visual Studio 2019

Visual Studio 2019 is an IDE specially designed for programming in Windows. It is developed by Microsoft, and its primary focus is developing in C++ and C#. As an IDE with built in Azure DevOps integration, it is a logical choice for this specific project (Microsoft, 2019a).

3.2.2 ASP.NET Core

ASP.NET Core is an open-source, cross-platform framework for developing modern web applications. It supports many modern programming paradigms and should offer the required performance for the project. Microsoft supports ASP.NET Core on Windows (Microsoft, 2020a).

3.2.3 ASP.NET Core Identity

ASP.NET Core Identity is a service that provides an easy-to-use solution for handling user authentication in ASP.NET Core development. With the use of the APIs the service provides everything from identity management to single sign-on (Anderson, R., 2020).

3.2.4 Entity Framework Core

Entity Framework Core (EF Core) is a cross-platform version of Entity Framework and is used for database communication. EF Core can serve as an object-relational mapper, which allows developers to work with a database using .NET objects as well as abstracting the underlying database logic. It also supports many different database engines, which means that the same code will work if the database engine needs to be changed in the future (Microsoft, 2016b).

3.2.5 Microsoft SQL Server

Microsoft SQL Server is a relational database management system, which is built for storing and retrieving data by other applications (Tutorialspoint, n.d.). It was chosen by the project group as the database management system, as this is a system we have familiarity and experience with.

3.2.6 Mailgun

Mailgun is an email transaction service, built with API's in mind. The service provides a framework for sending and receiving emails. They also offer a service to simplify messaging the users of their clients (Mailgun Technologies Inc, 2020).

3.2.7 Docker

Docker is a platform as a service that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated and self contained. They can communicate with each other only through specific channels. All containers can run on a single OS, and therefore use fewer resources than virtual machines (Docker Inc, 2020).

3.2.8 Kubernetes

Kubernetes is an open-source container-orchestration system for automating the deployment, scaling and management of the containers in the application. With many of these containers running on a single OS, it becomes necessary to manage them effectively (Kubernetes, 2020).

3.2.9 Postman API Client

Postman API Client is a tool used for API development. It allows the user to create and execute queries which will be useful when the project group wants to debug queries made by the application (Postman, n.d.).

3.2.10 RestSharp

RestSharp is an open-source HTTP client library developed for use in .NET development. It allows developers to build applications that communicate with APIs, without having to deal with raw HTTP requests and responses (Stackify, 2017).

3.3 Specification

With the choice of approach and selection of tools the project specification is building a web application in ASP.NET Core, with a back-end communicating with both the SQL Server and Kubernetes clusters. This is illustrated in Figure 3.1 below.

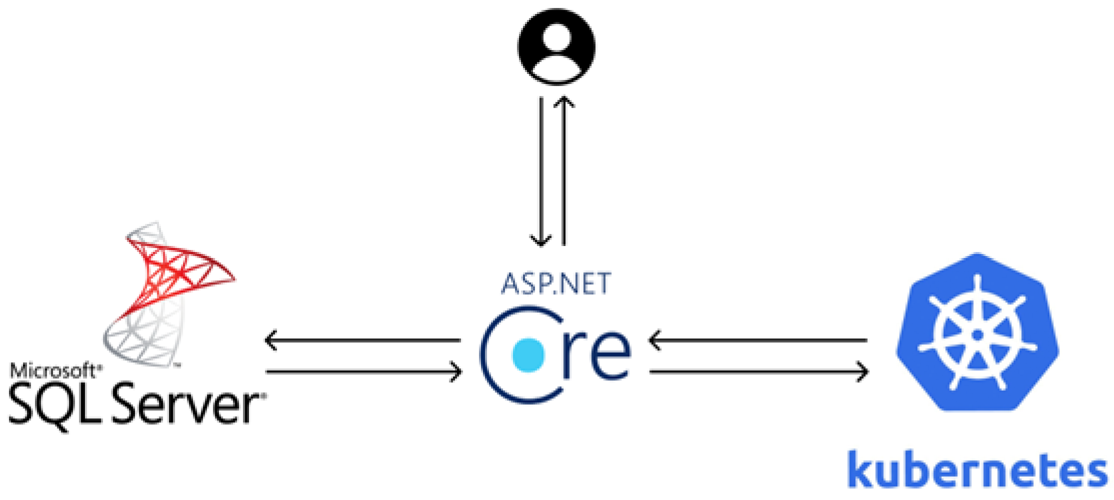


Figure 3.1: *Model of interaction*

3.4 Project development method

3.4.1 Development method

We developed the project through three sprints, each lasting two weeks. In advance of the start of each sprint we defined the size of the scope for that sprint, and decided what elements to include. Sprints were managed through Azure DevOps, using backlogs and taskboards. The backlog contained a list of all known requirements and tasks, and in which sprint the task should be completed. With the help of a velocity chart it was possible to coordinate and plan the average speed of a sprint, or how many tasks at a certain complexity could be completed per sprint. After a sprint was finished, the project group reviewed how the sprint went with the project owner.
With this development method it was possible to include bugs as a requirement of the project, meaning that any bugs we encountered could be included in the workload and handled during the sprint, as they are expected and even required to finish the sprint. This made our group agile and gave us the ability to deal with unforeseen problems.

3.4.2 Project Plan

The project plan was split into four different phases: Initialisation, implementation, evaluation and documentation. In the initialisation phase we focused on understanding the project specification and exploring different possible solutions. This phase also included gaining access to relevant resources like ZDatas DevOps environment, Mailgun and other APIs needed for the start

of the project, as well as most of the research required for the project. In the implementation phase the focus shifted to implementing the solution found in the initialisation phase. We worked iteratively in three sprints to develop the project while maintaining communication with the project owner. The final phase was the evaluation phase. In this phase the project group evaluated how the project went based on the evaluation methods described in section 3.5. The documentation phase was carried out throughout the other three phases. In this phase we developed the required documentation artifacts for the project.
The detailed project plan in the form of a Gantt chart can be found in Appendix A.

3.4.3 Risk analysis

Risk analysis can be found in Appendix B.

3.4.4 Risk management

This section describes details and possible mitigations to the risks discovered by the preformed risk analysis.

Virus outbreak

With the COVID-19 virus spreading rapidly at the startup phase of this project we will have to limit physical interactions with ZData, our HVL advisor and other parties. To mitigate the risks of the virus and to keep everyone healthy, meetings and interactions should be done digitally through video conference calls and emails. If someone gets ill, the project group must take the necessary precautions to contain the spread of the virus. In addition, the project group members must have knowledge of all aspects of the project to ensure that progress on the project is not halted.

Inexperience

Lack of experience could be a problem for our group. We are asked to use several technologies we have little to no experience with, which can lead to both delays and reduction in quality of the final product. To mitigate this risk, we should have frequent meetings with the project owner, where they can help us with the technical problems we might have with the task.

Time constraints

Considering that the duration of this project is relatively short, time management is crucial. The project consists of two parts, developing an application and writing a report. If too much time is dedicated to writing the project report, the development of the application might suffer as a result and vice versa. If this is not handled, it could lead to a reduction in quality of the two parts in the final product. The project group needs to make sure that our use of time is as efficient as possible, creating detailed plans for time management as well as making sure of what tasks lie ahead. When the development phase starts, we need to make sure that everything needed for development is ready, thus making the most of the time available to the project group.

Failure of equipment

As with every piece of technology, computers might fail. The project group needs to mitigate the work and time lost if such events were to occur. To mitigate this issue the project group will frequently perform backups by using a git-repository in Azure DevOps.

Misinterpreting project specification

When agreeing upon specific project specifications, misinterpreting certain aspects is always likely to happen. This needs to be handled as quickly as possible to ensure a productive workflow. By having frequent meetings in the startup phase, we can make sure that everyone has a common understanding of the specifications and requirements. Such meetings should be held at least once a week.

Poor communication with client

ZData is already an established company, and it is important that we understand that we may not always be their main priority. This can lead to poor communication, which may again lead to disappointing results. This risk can be mitigated through arranging meetings whenever the project group feels the need to have parts of the project clarified or evaluated. It will be important that the project group takes initiative to arrange such meetings when necessary.

Poor collaboration of group members

In any group project there is some risk of poor collaboration and if this risk is not handled the project result will suffer. As identified by Liu et al. (2010) poor group collaboration is often the result of three main factors: Poor motivation, a lack of accountability and negative interdependence. To reduce the chance of poor collaboration the group members must try to reduce the impact of the three factors mentioned above, which can be done by having weekly meetings where we discuss how we feel the workload has been distributed and if there needs to be made any changes.

3.5 Evaluation method

Since the project was developed through sprints, we initially planned to have bi-weekly reviews following each sprint to evaluate progress, workload, goals and time management. These reviews would be conducted by us and ZData in unison. We then planned to evaluate our own work after each sprint and cross-examine each other's code, expand the test cases to include corer cases for increased robustness, and run the project on different hardware to check stability. ZData would also evaluate our progress and finished tasks with their own bespoke metrics and give feedback on what they are satisfied with and what needs improvement. We intended to have a consistent and frequent evaluation period of the project to increase the quality of the finished product. Upon the completion of the project, ZData and the project group planned to evaluate the project through a code review, as well as an evaluation form, and decide if they are satisfied with the result.

4 Design and Creation

The project was created in .NET Core in VS and included a lot of prebuilt modules such as Razor Pages (explained in section 4.2.1) and MVC (Model-View-Controller). At first the project was built exclusively on MVC, but this made unit testing very difficult, so the design pattern was changed to View↔ Controller ↔ Handler ↔ Repository ↔ Model (MVCHR). It is important to include unit testing in the project as this makes it easier to debug and verify that the code and its associated methods are working correctly. Changing the design pattern was not a huge task, and mostly consists of moving methods out of Controller classes, and into relevant Handler and Repository classes.

4.1 Design patterns

4.1.1 MVC

The design pattern we ended up using throughout the project is a combination of the Model View Controller(MVC) pattern and the Repository design pattern. The MVC pattern is based on separating an application into three main groups of components: The Model, the View and the Controller. In this pattern user requests are sent to the controller where it communicates with the model to perform user actions and/or query a database. The controller then selects the view to show to the user, together with any relevant model data. The illustration in Figure 4.1 shows the main dependencies of the MVC design pattern.

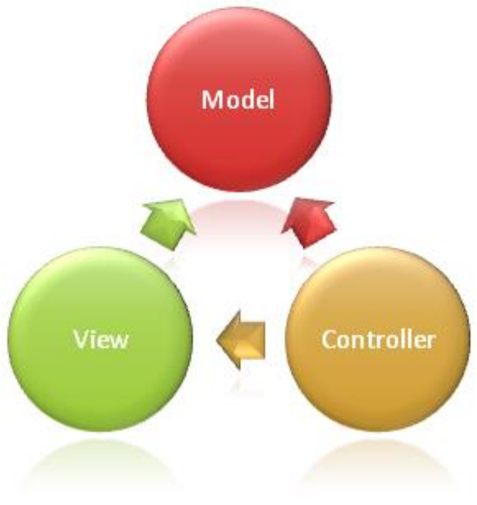


Figure 4.1: Dependencies in the MVC design pattern taken from Overview of ASP.NET Core MVC (Smith. 2020)

The main reason for using this design pattern is the scalability of the solution that is produced with MVC in mind. The reason for this is that it is far easier to test, update and debug parts of your application when they only have one job. With this clear division between different parts of our application it is easy to understand what component is related to a specific task if it fails or needs to be updated (Smith, 2020).

4.1.2 Repository design pattern

The other design pattern we chose to integrate into our application was the repository design pattern. The purpose of this design pattern is to separate the data access logic from other parts of the application, and map it to business entities in the business logic (Cubet, n.d.). With the help of interfaces, we can hide the unnecessary details of data access logic from the business logic. In our application this was done by creating a repository class for each handler, together with an interface. A controller would then use the handler, calling methods from the repository where the data access logic resides. This way we could disconnect our controllers from the data access logic that was necessary to perform queries to the database. This makes the application more scalable, as well as making testing possible with databases through a mocking technique. Mocking will be described further in section 4.2.4. See Figure 4.2 for a visual representation of the repository design pattern.

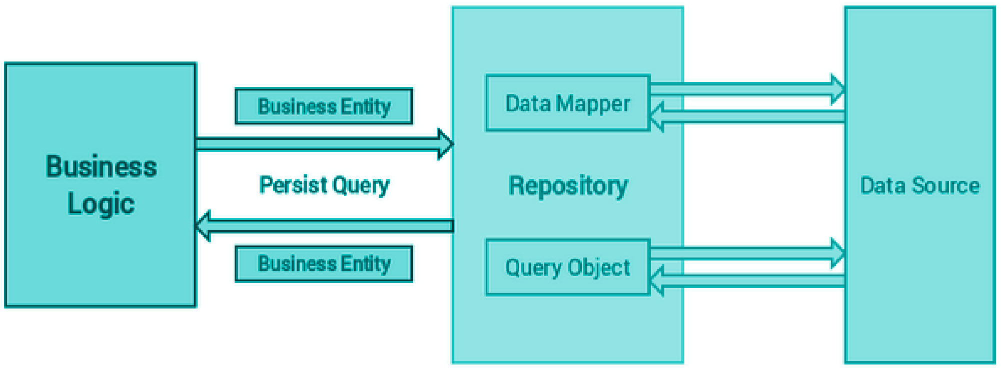


Figure 4.2*: Visual representation of the repository design pattern taken from Introduction to Repository Design Pattern, Cubet (Cubet, 2020)*

4.1.3 MVCHR

The main reason for using the combination of these design patterns in our project was to allow efficient testing of the application. This was specifically requested by ZData, who told us to implement their form of tests, making it easier for them to expand tests for the application later on. By using these combined design patterns we have a consistent structure for our application, making data access logic and business logic separated, as well as promoting scalability and testability. When creating a project for a business over such a short time period, scalability of the application becomes a major priority. The reason for this is that the developer portal will not be released directly after our six week period as agreed upon with ZData, and it is important that we focus on laying the foundations for an application that can be further developed by the internal developers at ZData. Since the developer sandbox will be a part of their larger developer portal application, making our work easily implementable into the remaining part of the application should be prioritized. If the finished solution of our project is a solid and scalable application, it will be simple for the project owner to further develop our solution to the developer sandbox. A well-thought design pattern is therefore essential to both the ultimate goal and the satisfaction of the problem owner.

4.2 Creation

4.2.1 Initialisation

In the early stages of the project we had a couple of specific goals to achieve:

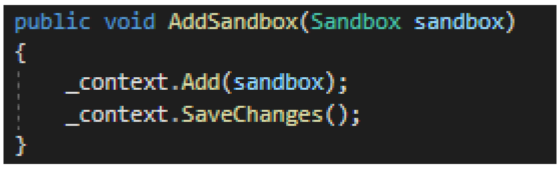
- Creating an MVC application with a membership system.
- Creating input forms for sending data to ZData's relevant API endpoints.
- Implementing the repository pattern to allow tests with mocked database connection.

As mentioned in the introduction to this chapter the project was created as a .NET Core application in Visual Studio using pre-built modules such as MVC and Razor pages. By creating a project with the MVC modules, certain aspects of early stages of development are automatically generated, such as the correct file structure for the solution, a couple of default pages of the website, and a standard CSS file which gives a unified theme for the website straight away. Using the automatically generated content is efficient in setting up the very basic components of the application. We use this standard CSS file for most of the pages throughout the whole project, with only very minor changes. This allowed us to almost completely ignore the visual design part of development, and to focus on the various features that were necessary to achieve the end goal of the application.

Another tool the project group took advantage of during early stages of development were Razor Pages. This module is built on top of ASP.NET Core MVC and includes a lightweight framework with full control over HTML. Using this in combination with ASP.NET Core Identity we could generate all manner of user authentication pages such as a user registration and login systems. Razor Pages provided us with a simple way to edit these pages to fit the application. ASP.NET Core Identity is an open source service which provides a user friendly solution for user authentication, making membership systems trivial to implement. The group was tasked with implementing an email confirmation system using the Mailgun API, which would generally have been a difficult task. This was made very simple due to the implementation of Identity, where different APIs for email confirmation are supported. This is only an example of the many unique benefits of ASP.NET Core Identity and Razor Pages, where larger tasks were simply replaced with the ability to swap a few lines of code with the respective information required such as saving users to a database and submitting information via forms.

4.2.2 Database integration

The next step was database integration. This step included both creating the database and handling communication between the application and the database. There are numerous ways to achieve this task, but with the use of EF Core a lot of heavy lifting can be avoided. The framework can deal with the task of mapping objects in the application to tables and columns in the database. It also comes with the ability to create database connections and execute database commands. The commands can be generated from abstracted commands given in the application as seen in Figure 4.3.



```
public void AddSandbox(Sandbox sandbox)
{
    _context.Add(sandbox);
    _context.SaveChanges();
}
```

***Figure 4.3**: Example of abstraction from database commands in C#*

The framework uses a model to perform data access. The model consists of entities and a context-object which is the object used to communicate with the database and allows the application to query and persist data (Microsoft, 2016b). EF Core uses a set of conventions when building a model based on how the entities are shaped. The conventions that are discovered can be extended and/or overwritten based on what is required by the project (Microsoft, 2019b).

The group implemented EF Core through creating entities for sandboxes, clients and users to match the ER-model found in Figure 4.4

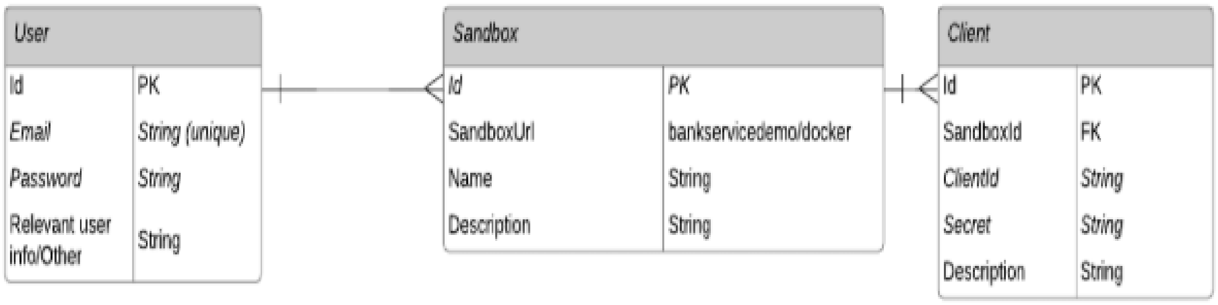
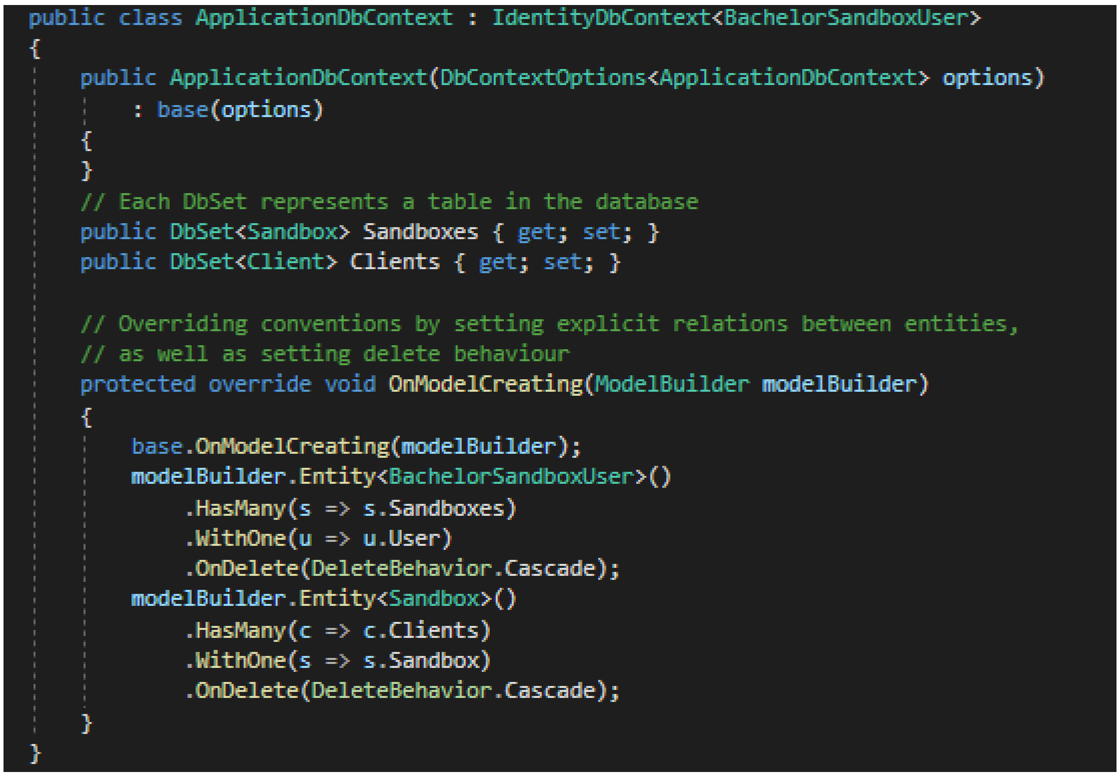


***Figure 4.4**: ER-model from ZData*

With the entities in place we created the context class containing the entities that are stored in the form of the property DbSet. Each DbSet can be seen as a table in the database. To map the relation between a user and a sandbox the context class had to inherit from the IdentityDbContext class. This handles the necessary entities for an Identity user, and the conventions were overwritten in OnModelCreating to ensure that both the relations were created as intended and to specify the wanted delete behaviour. The context class is shown in Figure 4.3.

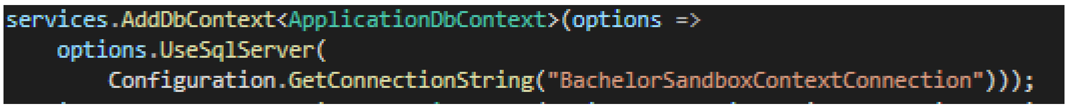


```
public class ApplicationDbContext : IdentityDbContext<BachelorSandboxUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
    // Each DbSet represents a table in the database
    public DbSet<Sandbox> Sandboxes { get; set; }
    public DbSet<Client> Clients { get; set; }

    // Overriding conventions by setting explicit relations between entities,
    // as well as setting delete behaviour
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Entity<BachelorSandboxUser>()
            .HasMany(s => s.Sandboxes)
            .WithOne(u => u.User)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<Sandbox>()
            .HasMany(c => c.Clients)
            .WithOne(s => s.Sandbox)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

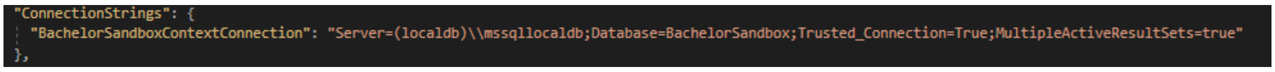
Figure 4.5: *ApplicationDbContext class which inherits from IdentityDbContext*

To be able to connect the database to the application, the context class had to be added to the configuration of services, which is run at the startup of the application. This is an easy process, where the context class is set as the type of the added service and the type of database management system is set as an option, as can be seen in Figure 4.5. The option also contains a connection string, which can be set in a JSON-file called appsettings cf. Figure 4.6. When the project is deployed, the connection string must be changed to match a hosted database. More details on how this was accomplished can be found in section 4.2.7.



```
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(
        Configuration.GetConnectionString("BachelorSandboxContextConnection")));
```

Figure 4.5: *Code snippet from ConfigureServices which adds our DbContext and selects a SQL Server*



```
"ConnectionStrings": {
  "BachelorSandboxContextConnection": "Server=(localdb)\\mssqllocaldb;Database=BachelorSandbox;Trusted_Connection=True;MultipleActiveResultSets=true"
},
```

Figure 4.6: *Connection string for locally hosted SQL server*

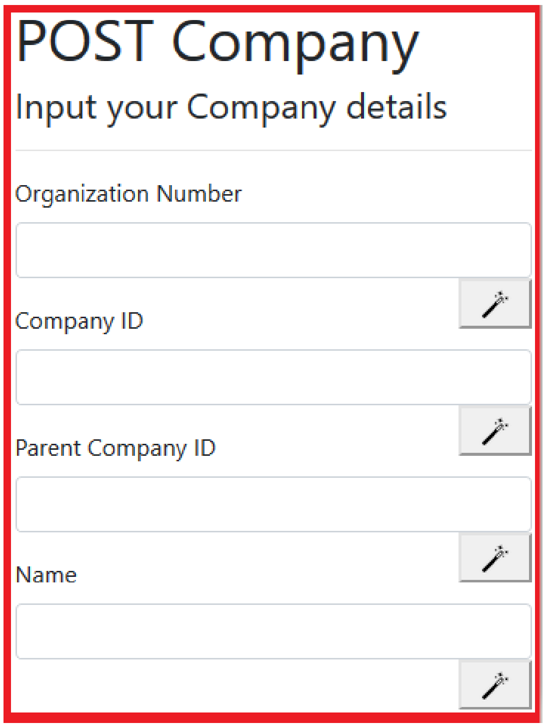
4.2.3 Mailgun integration

After having successfully created and integrated a database into our application, the next upcoming task was to implement an email confirmation system using the Mailgun API. This API let the project group easily implement the system by using a code snippet made by Eric L Anderson (Anderson, E., 2020).

Since ASP.NET Core Identity supports email confirmation, we could call the method in the registration form’s backend as specified via automatically generated comments, and the API call was successfully implemented into the application. Another necessary element to implement this task was adding a json user secrets file with Mailgun credentials received from ZData. Since user secrets are locally stored and not checked into source control (Anderson et al., 2020), another approach had to be taken when deploying the project. This approach is described in section 4.2.6. Implementing these API calls was a relatively straightforward task in the project, and thus no further commentary on this was necessary.

4.2.4 API information from input forms

The layout of the sandbox web page consists of multiple input fields as seen in Figure 4.7 and in some cases radio buttons, which the user will need to fill out to get their API response. Some of the fields are optional, and every field can be filled with mock sample data for the convenience of the user. More information about mocking data can be found in section 4.2.8. After the user has input all of their data and clicked the submit button, all of the data is collected and stored in C# classes. This is then organized and formatted so that it is easier to understand and observe the information. The data is then sent through the response of the web page and the page is reloaded with the newly submitted data. This means the data can easily be displayed on the page as a JSON (JavaScript Object Notation) object for the user to look at so they can get an idea of how the data is collected and handled. See Figure 4.15 for an example of the JSON format.


POST Company

Input your Company details

Organization Number

Company ID

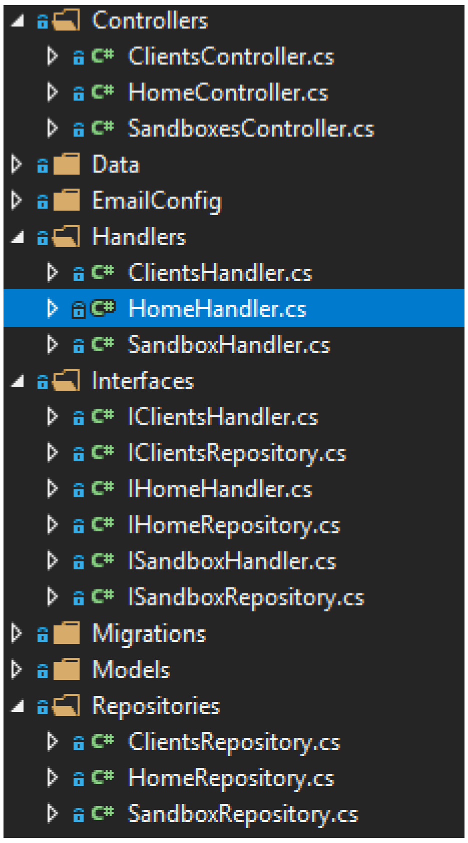
Parent Company ID

Name

Figure 4.7: *Forms for the user to input data. Can also generate sample data by clicking the magic wand.*

4.2.5 Testing with XUnit

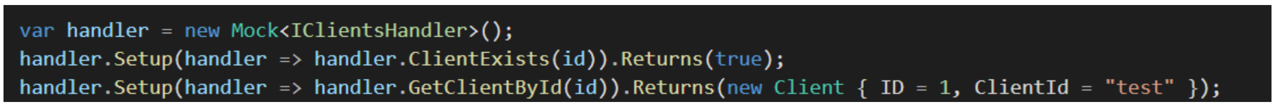
As assigned by the problem owner we were requested to implement tests for the application at the end of the first sprint. We had a short demonstration by ZData showing how they wanted us to test our application, and in how their tests usually function. This task turned out to be the most challenging part of our first sprint, as testing an application with data access logic required a large-scale reconstruction in the software architecture. It was at this point we were advised to implement the repository design pattern on top of the already existing MVC pattern, opening the application to testing with data access logic separated from controllers. To achieve this goal we created an interface for each repository with methods for CRUD operations, as well as methods for specific queries that needed to be executed in the respective controller. In these repositories our project group implemented the relevant methods to be used for database access, so that they could later be mocked for testing. This will be further elaborated on in section 4.2.7. We also created interfaces and handlers to further separate this logic, making the controllers completely separate from all business and data access logic. See Figure 4.8 for a visual representation of the file structure.



```
Controllers
    ClientsController.cs
    HomeController.cs
    SandboxesController.cs
Data
EmailConfig
Handlers
    ClientsHandler.cs
    HomeHandler.cs
    SandboxHandler.cs
Interfaces
    IClientsHandler.cs
    IClientsRepository.cs
    IHomeHandler.cs
    IHomeRepository.cs
    ISandboxHandler.cs
    ISandboxRepository.cs
Migrations
Models
Repositories
    ClientsRepository.cs
    HomeRepository.cs
    SandboxRepository.cs
```

***Figure 4.8**: Overview of file structure for the repository design pattern.*

With the architecture being properly set up, we could start creating tests for our application. This was done by creating a separate test project inside the same solution. Using XUnit we created three test classes that would function as a testing environment for our controllers. Each method in the respective controller would have its functionality tested via the creation of Mocks for each handler using the Moq NuGet package. When mocking an interface in this way we could specify which methods should be run, and what they should return in the case where they were executed. See Figure 4.9 below for an example related to our client controller test class, where we created an instance of the handler using the Moq package and specify the output for two methods.



```
var handler = new Mock<IClientsHandler>();
handler.Setup(handler => handler.ClientExists(id)).Returns(true);
handler.Setup(handler => handler.GetClientById(id)).Returns(new Client { ID = 1, ClientId = "test" });
```

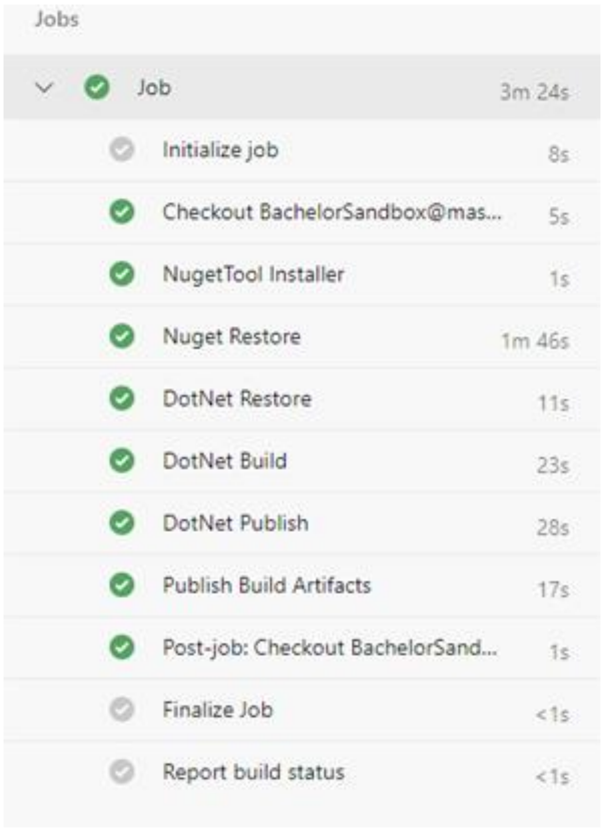
Figure 4.9: *Code snippet from ClientsControllerTest which mocks the ClientsHandler and defines the return values for two methods.*

By using this technique, we were able to make passing test cases for our application with relevant data that was expected to be returned from the database, as well as mocking expected behavior from other required method calls.

4.2.6 Azure pipelines and deployment

Deploying an ASP.NET Core app can be done in several ways, but in general it requires a hosting server and a process manager. The process manager handles starting the application when requests arrive, as well as restarting the application if it crashes or the server reboots (Microsoft, 2020b). Since the project group had been given access to Azure resource groups, we chose to use Azure Pipelines to automate deployment of the app to Azure.

Azure Pipelines is a cloud service that can be used to automatically build and test a project, as well as deploying it to any target (Microsoft, 2019c). The process of deploying to an Azure service is split into two pipelines when using Azure DevOps. The first pipeline is responsible for building, testing and publishing the project to a folder used in the other pipeline. To achieve this, the pipeline consists of multiple tasks as seen in Figure 4.10.



```
Jobs
Job                                  3m 24s
    Initialize job                   8s
    Checkout BachelorSandbox@mas...  5s
    NugetTool Installer              1s
    Nuget Restore                    1m 46s
    DotNet Restore                   11s
    DotNet Build                     23s
    DotNet Publish                   28s
    Publish Build Artifacts          17s
    Post-job: Checkout BachelorSand...  1s
    Finalize Job                     <1s
    Report build status              <1s
```

Figure 4.10: Pipeline with all tasks after successful execution

These tasks are specified in a YAML-file that is bundled with the project. The second pipeline is a release pipeline. This pipeline takes the artifact created by the first pipeline, and runs the tasks required to deploy the app to the targeted service. The project group only used one task in this pipeline as seen in Figure 4.11, as that is all that we required to target an Azure App Service.

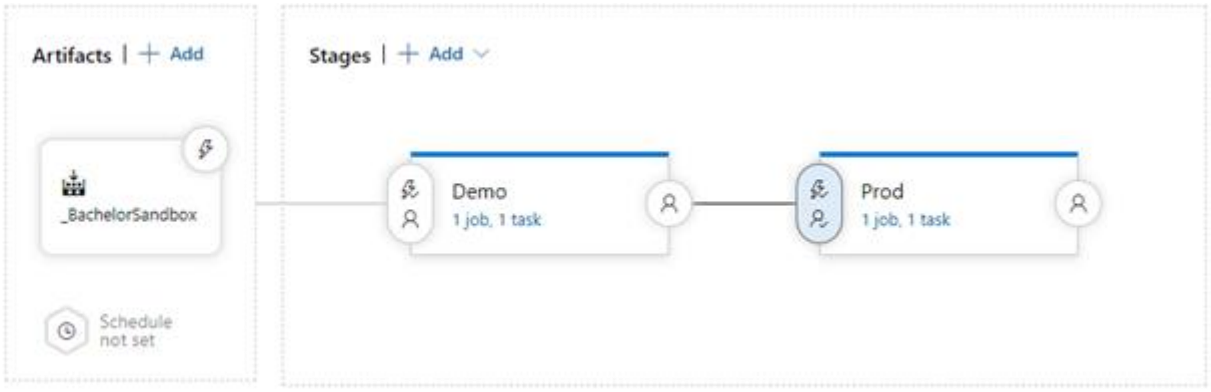
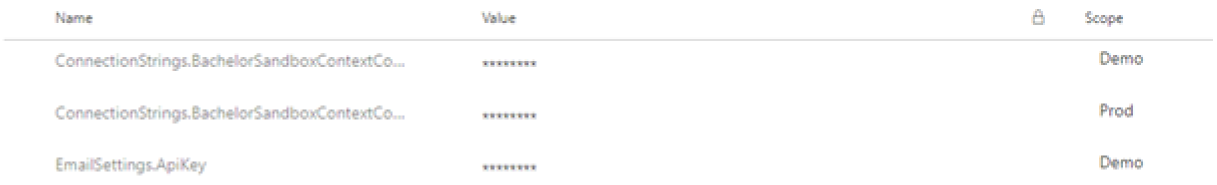


Figure 4.11: *Release pipeline with deployment to two different environments*

Both pipelines can either be triggered manually, or automatically based on a condition set in it. For example, the first pipeline can be triggered by changes committed to the master branch, which in turn can trigger the second one if the build is successful.

As mentioned in section 4.2.3 user secrets for the Mailgun settings were no longer a viable option when deploying the application, and we could no longer use the connection string to the database hosted locally (see section 4.2.2). An easy solution to these concerns is using pipeline variables. Pipeline variables can be used in the release pipeline to perform variable substitution as part of the deployment task. As the connection string contains login information to the database and the API key for Mailgun should be kept private these values were hidden using a feature built into pipeline variables, as seen in Figure 4.12.

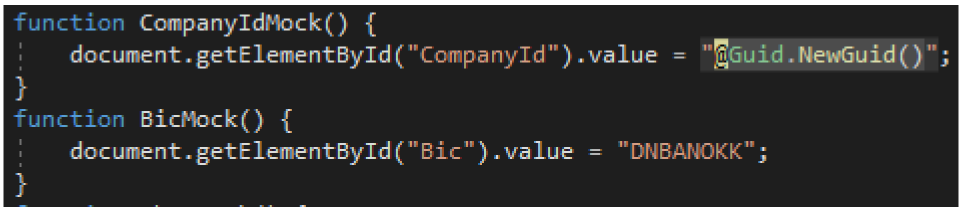


Name	Value	🔒	Scope
ConnectionStrings.BachelorSandboxContextCo...	********		Demo
ConnectionStrings.BachelorSandboxContextCo...	********		Prod
EmailSettings.ApiKey	********		Demo

Figure 4.12: *Pipeline variables*

4.2.7 Mocked input

To make the application more user friendly, we have built in the functionality for the sandbox to autofill its own fields. There is no need for the user to painstakingly fill out every field if they just want a quick overview of the format and type of data that they should fill in. This also lets users quickly test each API with predetermined data. All of the mocked input is handled through javascript, with every instance of data being mocked statically. We found no particular need for the data to be unique every time, so to simplify the process slightly, the data was mocked statically. However, there is one exception, and that is in regard to the field CompanyId and MessageId. Since these two fields are used as identifiers to keep each data package separate, they have to be unique, or the program will end up replacing data continuously. Therefore, the data is mocked by random generation every time, quite simply by using Guid. Guid is a built in function in .NET that generates a unique ID, so the only thing needed for this to work seamlessly is a quick razor reference to C# code, allowing the C# method to inject data into the javascript function, which populates the HTML field with the relevant data. See figure 4.13 for an example of the mocking functions.



```
function CompanyIdMock() {
    document.getElementById("CompanyId").value = "@Guid.NewGuid()";
}
function BicMock() {
    document.getElementById("Bic").value = "DNBANOKK";
}
```

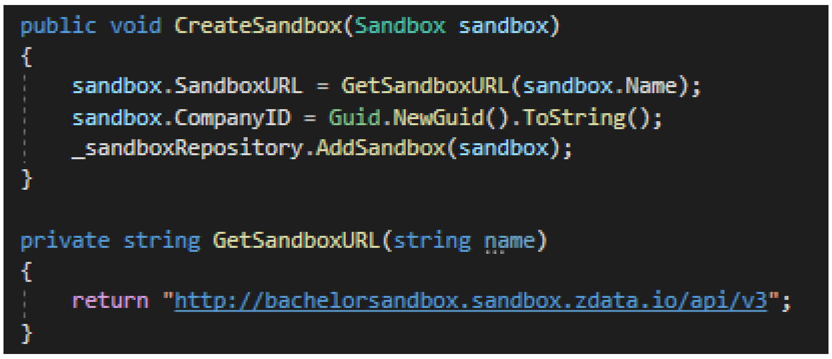
***Figure 4.13**: Code snippet from AccountPostForm, populating HTML fields with random and example data.*

4.2.8 Kubernetes and Docker

A large part of the second sprint was based on Docker and Kubernetes. These were both complicated technologies that we had no previous experience with, and a lot of time was spent on researching their implementation and functionality. Docker is a tool which is designed to create, run and deploy applications using containers. These containers are units of software that package up code and all relevant dependencies to ensure that the application runs reliably from one computing environment to another (Docker Inc, 2020a). Unfortunately, due to the lack of time and experience, we ended up not using docker to its full extent. The original idea was to implement a system where a unique container is created for each individual user, so that there would be some separation between users. Instead we ended up with a system where every sandbox is created on a shared cluster. This is adequate for small scale production, as long as the number of users remains low, but should probably be improved and fixed in the future.

4.2.9 API Communication

We successfully hosted the docker image of ZDatas APIs to a Kubernetes cluster. To connect it to a sandbox from the application, we simply removed the user’s ability to set what the “Sandbox URL” should be and instead set it to the URL of the hosted docker image, as shown in Figure 4.14. This was implemented in such a way that customised client URLs could be created by new Kubernetes clusters in future iterations.

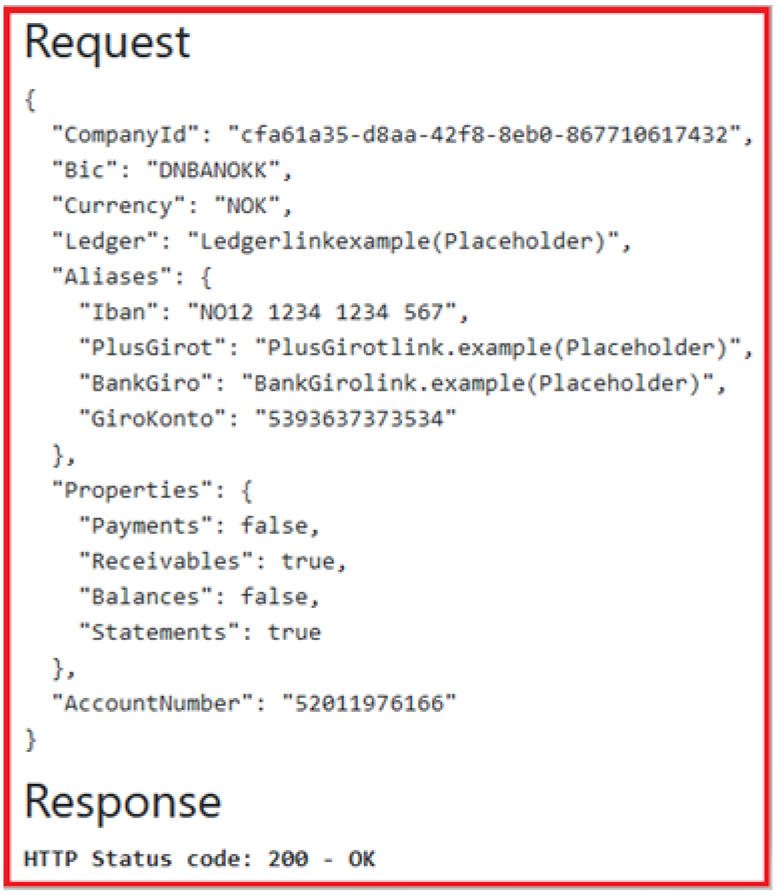


```
public void CreateSandbox(Sandbox sandbox)
{
    sandbox.SandboxURL = GetSandboxURL(sandbox.Name);
    sandbox.CompanyID = Guid.NewGuid().ToString();
    _sandboxRepository.AddSandbox(sandbox);
}

private string GetSandboxURL(string name)
{
    return "http://bachelorsandbox.sandbox.zdata.io/api/v3";
}
```

***Figure 4.14**: Code snippet of creating a sandbox from SandboxHandler.cs*

With the connection in place, new sandboxes now held a URL reference to a client that could receive requests. To send the requests the project group used a HTTP client library called RestSharp. This library contains methods to create a connection to a client, creating requests, as well as methods for executing requests sent to the client. For the GET endpoints all that was required was creating a client, creating a request, executing the request and finally parsing the response from the client. A similar route was used for the POST endpoints, with one additional step to add the JSON object created by the user to the request body (see figure 4.15).



Request

```
{
  "CompanyId": "cfa61a35-d8aa-42f8-8eb0-867710617432",
  "Bic": "DNBANOKK",
  "Currency": "NOK",
  "Ledger": "Ledgerlinkexample(Placeholder)",
  "Aliases": {
    "Iban": "NO12 1234 1234 567",
    "PlusGirot": "PlusGirotlink.example(Placeholder)",
    "BankGiro": "BankGirolink.example(Placeholder)",
    "GiroKonto": "5393637373534"
  },
  "Properties": {
    "Payments": false,
    "Receivables": true,
    "Balances": false,
    "Statements": true
  },
  "AccountNumber": "52011976166"
}
```

Response

```
HTTP Status code: 200 - OK
```

***Figure 4.15**: Displayed JSON and HTTP Status on the web page after a user submits data.*

The project group had up until this point not been able to test if the serialized JSON-objects created by our input-forms would be valid request bodies to the APIs. It turned out that all but one request body was valid, as the last request returned the HTTP response code “400 Bad Request”. To debug this the project group used Postman and identified that the model for the JSON object missed an array compared to the model the API expected. This was fixed by updating the affected model to match the required model. Thus, we had created a functional sandbox.

5 Evaluation

5.1 Evaluation method

The intended evaluation method for this project was discussed in section 3.5. This chapter includes a thorough explanation of how these evaluations were made and how they changed. Furthermore, this chapter will discuss downsides and advantages of the techniques we used throughout this project. When developing an application of this size, frequent evaluations and reviews are necessary to ensure good quality in the end result. In our project we utilized two distinct methods of evaluation, direct feedback from the project owner, and expanding test cases.

5.1.1 Direct feedback from the project owner

At the beginning of the project it was suggested by the project owner that the project group should use Azure DevOps for storing the current solution via Azure Repos. They also used it to monitor the progress for the ongoing sprint and overall progress using Azure Boards. Azure DevOps is an online platform with several cloud services, of which we used:

- Azure Pipelines: Supports continuous deployment of applications.
- Azure Boards: Used to plan projects using various development methods.
- Azure Repos: Cloud hosting for private Git Repositories.

(Cool, 2018)
By using these tools we could push our application to the repository, have it automatically deployed via Azure Pipelines and then change the status of the current task we were working on in Azure Boards. This was an effective tool for getting feedback from ZData as this would allow them to see the current solution quickly. It also gave them information about what tasks we were currently working on with details about progression and the amount of time spent.
By using all of these integrated systems in DevOps we could rely on quality feedback from ZData, as they had good insights into our workflow and contemporary solution. Meetings were held at the end of every sprint to inform them about our progress and to review changes that needed to be made according to their feedback. We would also start planning the next sprint together, by taking into account time allocations, difficulty and priority for different tasks based on the results from the previous sprint.
Another tool we utilized was an online chatting service called Slack. This tool was used for short direct messages in a group where we could ask about technical difficulties and other problems related to the project. It was not until the second sprint we started using this service more effectively, but the responses were also lacking. Sometimes we would get replies within minutes and other times it could take hours. This however became less of a problem as the project went on, as we developed a better understanding of the times when they were available for questions. Eventually the time to get a reply was relatively short. This was an unforeseen risk, and it would have been more efficient to have a predetermined time slot where we could ask questions or at least an estimated timeframe for when we should ask for guidance.

At the end of the project we sent an evaluation form to our advisor at ZData, see Appendix D. By using this evaluation form we got an overview of the problem owner's satisfaction with different parts of the application, as well as some feedback about our workflow and communication.

5.1.2 Expanding Test Cases

One approach to evaluating the project that we decided to use is unit testing. Using tests we can run certain parts of our code in a specified testing environment, without having to worry about security issues that can happen and impact the performance of the live application. This can help you detect problems in your code before deploying the current solution, and is an effective way to evaluate what seems to be a finished application.
More information about the process of writing the tests can be found in section 4.2.5.
Implementing the tests required restructuring the whole architecture of the application. The project group has lacking experience in properly writing tests of this type, which made it a huge task. It took about 40 hours over the original estimate of 20. This was addressed when planning the upcoming sprints, but had a big impact on the total time commitment for sprint one.
Through these tests we were able to ensure the quality of the application, although there were very few unseen issues when running them. The total time spent on creating the tests may have been large during the first sprint, but when implementing new tests in the subsequent sprints the implementation was swift, making the evaluation method effective overall. These tests made us more confident in the application, and with limited time extensive user testing would not have been possible.

5.2 Evaluation results

To get an overview of the project owner’s opinions on the solution and project period we created an evaluation form (see Appendix D). ZData was in general pleased about the outcome of the project, especially our communication and workflow. We were also pleased with the communication throughout the project, and seeing as both the project group and the problem owner were satisfied with how we communicated, this way of evaluating was a huge contributor to the overall goal of the project.

A less successful part of the project was the integration of different software and technologies into the application. This is largely because of the compromises we had to make for the implementation of Docker, as explained in section 4.2.8. ZData does however explain in question 8 of the evaluation form (see Appendix D) that this could have been handled smoother if they were better prepared regarding these technologies.

To summarize, the project owner was pleased with the solution that we developed, where communication with ZData proved to be the most effective way to ensure the quality for the final product. Getting feedback at the end of each sprint and having an open communication channel through Slack for guidance was an efficient way to make sure the problem owner was participating in the solution, and to make the best product possible.
The implementation of unit tests for the application proved to be a less effective

evaluation tool, but worked well together with tight communication with ZData. The tests were a solid way for the project group to evaluate the solution before presenting it to ZData or when feedback from ZData was unavailable due to lack of time.

6 Discussion

6.1 Planning

When working on a project of this scale it is important to prepare sufficiently. Due to being inexperienced in the field, the project group never quite realized the sheer scale this project would have before starting the implementation. When planning for the project together with the problem owner a lot of different technologies were often mentioned, and the breadth of knowledge required to implement these was definitely underrated. It was difficult to understand how all of these technologies would fit together when starting out, and going over this with the problem owner would have been a good idea to plan the overall structure of the application before even starting on the implementation.

In this project our planning stage went on for longer than expected, and contributed to the problem with lack of time that we experienced when closing in on the end of the project period. Regardless of how detailed our Gantt chart for time management was and how much research we did for the different technologies, developing the solution is going to take a long time. In hindsight we should have started earlier with the implementation, researching the different technologies underway, rather than trying to go too in depth into the different technologies before really knowing how they would fit into the application.

Another resource that could have been used to have a more efficient project period would have been to have other reference projects from ZData that could have been reviewed early on to gain insight into what kind of a solution we were creating. These reference projects could have had similar implementations for the different technologies we were not familiar with, as well as an example for how to write ZData's way of Unit testing. With example projects with similar elements as our own project we could have saved a lot of time on trying to figure out how to implement new technologies such as Docker and Kubernetes. This is a difficult task as there was little information about their own products ZData could share with us, but having another project for reference in early stages of development would have been helpful.

6.2 Execution

A consistent and reliable workflow is important in a group project like this, making sure each member knows what steps are necessary to go through when implementing a new feature into the application. Our workflow can be described in 6 steps:

1. Select a task from the Azure DevOps sprint board.
2. Develop a solution to the current task.
3. Test locally to make sure the implementation works as intended.
4. Use push requests to commit solutions to the current branch.
5. Have the implementation verified by another group member, by the use of pull requests, to ensure high quality.

6. Close the relevant Azure DevOps task and start over from point 1.

This workflow worked well throughout the project, minimizing the amount of time used between tasks. An essential part for this to work was the use of Azure DevOps sprint boards, where we were given multiple tasks from the problem owner at the beginning of each sprint. By using these larger tasks that were provided, we could split them up into smaller more manageable tasks to select from when choosing from the sprint board.

6.3 Advantages and disadvantages of selected approach

6.3.1 Planning

As described in 6.1 our approach to planning for the project was based on envisioning how all elements and features of the solution would fit together when implemented. This process could have been streamlined by having more conversations with our ZData advisor early on, and preparing more relevant questions related to the project to get better insight into how they would approach this kind of development. Doing this would have been more efficient, leading to more time for development. The advantage of our current approach is that we, through research, obtained more in-depth knowledge about different technologies, which means integrating multiple and different technologies into a single project becomes much easier. This has the added benefit of allowing us to take a similar approach in future projects, not just the current one.

6.3.2 Execution

In section 6.2 we described our approach to execution of the project, with a focus on the workflow during development. Working on smaller tasks that were parts of larger tasks provided by the problem owner made each individual task seem less daunting when selecting what to work on next, and provided a clear motivation to collaborate on the same feature since each group member could work on similar tasks at the same time. One problem we encountered in our workflow was that when the larger tasks got more complex and contained technologies we had little to no experience with, it was difficult to separate them into small manageable tasks. When the technology is unknown it takes a long time to figure out what components are necessary during implementation. Then the original workflow naturally transforms into each person working on a larger task by themselves, with meetings to help other group members understand their implementation. This also became a problem in step five of our workflow. Since everyone was working og separate parts of the application which were substantially different, having group members verify each other's code became less relevant. This could have been improved by taking more time to separate tasks into smaller ones in the later stages of development, or asking ZData for assistance with separation of tasks. The ultimate goal of the project was still reached, but this approach was volatile and could have caused major issues in a larger project.

7 Conclusion

7.1 Achievement and goals

At the start of this project, the initial specifications of the project were as follows:

1. To develop a sandbox for testing API's (Specifically those that belong to ZData).
2. To create a user authentication system.
3. To implement Kubernetes clusters for each user.
4. To create a project that supports further development by external parties.

All of these goals were achieved, although some compromises had to be made.

Developing a sandbox for testing API's

The sandbox we created consists of two different parts, request and response (or POST and GET). Building these systems progressed smoothly and continuously, and provided us with no particular challenges. We successfully created a sandbox that lets the user input their data and visualize the JSON object created from this data, as well as the user match their sandbox with a CompanyID to fetch a JSON object from ZDatas external server.

Creating a user authentication system

This was another relatively straightforward goal, in large credit due to ASP .NET CORE Identity which automates almost everything you need for an authentication system. After automatically adding everything needed for identity in the project, we tweaked the files we needed so that they would match what we required. We only spent substantial time on the implementation of Mailgun. Since we wanted users to validate their email addresses for the sandbox, Mailgun serves as the automatic sender of the emails. In the email there is a link that validates the email through the project backend. After this implementation, the goal was successfully completed.

Implementing Kubernetes clusters for each user

Originally the purpose of this element was for a unique Docker container to be created for each individual user. However, integrating Docker containers proved to be a real challenge, and after spending two full days, with every member researching and experimenting, we decided the effort needed to fully complete this task would exceed the time we had remaining on the project. We were still able to use Docker containers and Kubernetes, but we opted for a simplified solution, where all users were situated on a single Docker container. This means that the data of each individual user is no longer completely secure since they are all sharing a container. Separating the users is a necessary step to improve security.

Creating a project that supports further development by external parties

An important requirement of the project was that ZData would be able to expand upon the original project after we had completed our development process. This meant that we had to make sure the code was readable, well commented, and orderly structured. The addition of Unit Tests also means that the code has a higher level of quality, and makes it easier for other developers to test our code, and potentially their own. After the final code review performed by ZData, and some minor issues were resolved, the feedback we got was that they were satisfied with the result, and believed they could take this project and continue development without major delay. We conclude that this constitutes a successful completion of the goal.

7.2 Future possibilities

This project was developed much in the style of client/provider, in which we were the providers of a product, and ZData was our client requesting a product. As such, should ZData release this project as a fully-fledged service to supplement the rest of their product lineup, it seems unlikely that any other student groups or others would be allowed to view the source code. ZData would likely want to protect their business interests. However, the contents within the project would surely be useful to anyone interested in designing their own API sandbox, as many of the same principles would apply.

After our work on the project ends, we expect ZData to start working on relevant expansion. Ideally, we should have delivered a complete product ready for release, but the time constraints of the project meant this was not feasible. Had we continued development ourselves, then we would have implemented the following:

1. Security and encryption of sandbox data
2. Styling and design of web pages
3. Greater amount of options for users to control sandboxes

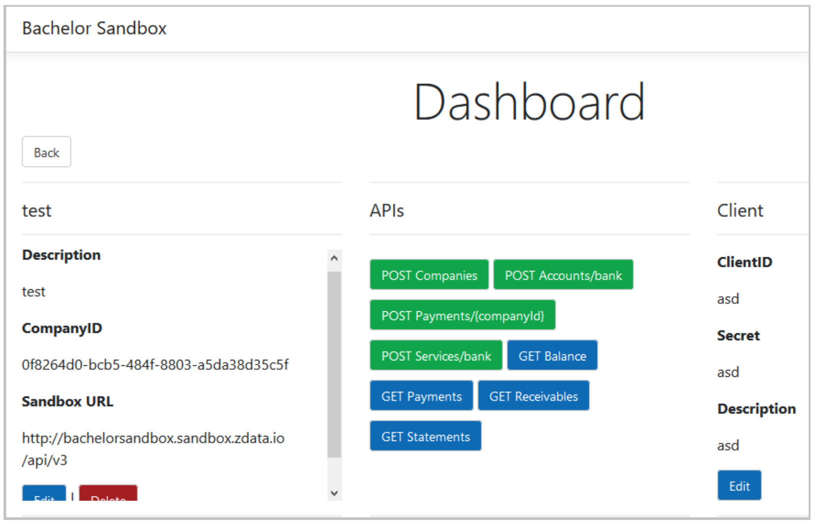
Should ZData release the project publicly, it would be logical to develop these elements first.

Security and encryption of sandbox data

The sandboxes each individual user creates are only protected by the authentication of the user during login. More advanced attacks could theoretically expose data, as this was not something we designed against. However, implementing a level of encryption, where the data becomes obfuscated, and only agents with key access can view data, would fix this issue entirely. This would mean that only the user themselves and ZData would be able to view the data. There could also be an even heavier layer of security where ZData can not even view the data, only the server can, which would be in a closed system.

Styling and design of web pages

Having a modern and pleasing design makes web pages more attractive and desirable. The basic web page designs created through ASP .NET CORE are not unacceptable, but they are simple and generic as seen in Figure 7.1. Considering ZData has their own unique styling and design on their web pages, it is only natural that the project would need to be updated to match their overall look. Regardless of time constraints on the project, we would always have saved this step for the very end, as until the final flow of interaction had been decided. Then an overall design style could be applied, that would complement the content on each page.


Bachelor Sandbox

Dashboard

Back

test

Description

test

CompanyID

0f8264d0-bcb5-484f-8803-a5da38d35c5f

Sandbox URL

http://bachelorsandbox.sandbox.zdata.io
/api/v3

APIs

POST Companies | POST Accounts/bank | POST Payments/{companyId} | POST Services/bank | GET Balance | GET Payments | GET Receivables | GET Statements

Client

ClientID

asd

Secret

asd

Description

asd

Edit

Figure 7.1*: Final styling and design of web pages*

Greater amount of options for users to control sandboxes

In general, each individual sandbox works exactly to specification, but there is no interplay between them. It could have been meaningful for users to create a project of sandboxes, and decide which sandboxes to include in the project, and then send data through the various forms. As the data is sent and manipulated by each individual sandbox, the user could follow along and get a great overview over the different services that ZData offers. It might also be an idea to add some coloring options for each sandbox, especially in a multiple-project-scenario where color coding would help users immediately identify which project they are working in.

7.3 Closing remarks

Completing development on this project that has massively increased our knowledge and competency in the field. Learning more about the principles behind MVC and Repositories and Handlers, and working with modern technologies like Docker and Kubernetes has been a great challenge, both in difficulty and enjoyment. The unprecedented COVID-19 situation also meant that the entire project was developed separately, with no meetings being conducted in person. Overcoming this was an opportunity to grow and learn. It also made individual and independent contributions to the project more meaningful. Overall, we feel satisfied and content that the project we sign off on and hand over to ZData meets their requirements and is of acceptable quality.

References

Anderson, E. (2017) *Email With ASP.NET Core using Mailgun from:* https://elanderson.net/2017/02/email-with-asp-net-core-using-mailgun/ (Date viewed: 04.05.2020)

Anderson, R. (2020) *Introduction to Identity on ASP.NET Core*. Available from: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1&tabs=visual-studio (Date viewed: 01.05.2020)

Anderson, R. et al (2020) *Safe storage of app secrets in development in ASP.NET Core*. Available from: https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-3.1&tabs=windows (Date viewed: 05.05.2020)

AngularJS (2018) *AngularJS*. Available from: https://angularjs.org/ (Date viewed: 30.03.2020)

Cool, J (.2018) *Introducing Azure DevOps*. Available from: https://azure.microsoft.com/en-us/blog/introducing-azure-devops/ (Date viewed: 19.05.2020)

Cubet (n.d.) *Introduction to repository design pattern.* Available from: https://cubettech.com/resources/blog/introduction-to-repository-design-pattern/ (Date viewed: 02.05.2020)

Docker Inc (2020a) *What is a Container*. Available from: https://www.docker.com/resources/what-container (Date viewed: 25.03.2020)

Docker Inc (2020b) *Docker.* Available from: https://www.docker.com/ (Date viewed: 30.03.2020)

EU (2015) *Info about Directive (EU) 2015/2366.* Available from: https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366/law-details_en (Date viewed: 30.03.2020)

Invensis Pvt. Ltd. (2015) *Advantages and Disadvantages of .NET and Java.* Available from: https://www.invensis.net/blog/it/advantages-and-disadvantages-of-dotnet-and-java/ (Date viewed: 02.04.2020)

Oracle (2019) *Java Servlet Technology.* Available from: https://www.oracle.com/technetwork/java/index-jsp-135475.html (Date viewed: 30.03.2020)

Kubernetes (2020) *Production-Grade Container Orchestration.* Available from: https://kubernetes.io/ (Date viewed: 30.03.2020)

Liu, S., et al (2010) *Students' perceptions of the factors leading to unsuccessful group collaboration.* In: **Proceedings, 10th IEEE International Conference on Advanced Learning Technologies** (ICALT 2010), IEEE, pp. 565–569

Mailgun Technologies Inc (2020) *Mailgun*. Available from: https://www.mailgun.com/ (Date viewed: 30.03.2020)

Microsoft(2016a) *Common Type System & Common Language Specification.* Available from: https://docs.microsoft.com/en-us/dotnet/standard/common-type-system (Date viewed: 02.04.2020)

Microsoft (2016b) *Overview of Entity Framework Core.* Available from: https://docs.microsoft.com/en-us/ef/core/ (Date viewed: 04.05.2020)

Microsoft (2019a) Visual Studio 2019. Available from: https://visualstudio.microsoft.com/vs/ (Date viewed: 30.03.2020)

Microsoft (2019b) *Creating and configuring a model.* Available from: https://docs.microsoft.com/en-us/ef/core/modeling/ (Date viewed: 18.05.2020)

Microsoft (2019c), *What is Azure pipelines?* Available from: https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops (Date viewed: 25.05.2020)

Microsoft (2020a) *What is* ASP*.NET Core?.* Available from: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core (Date viewed: 30.03.2020)

Microsoft (2020b), *Host and deploy ASP.NET Core.* Available from: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/?view=aspnetcore-3.1 (Date viewed: 25.05.2020)

Mozilla (2020) *JavaScript Reference.* Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference (Date viewed: 30.03.2020)

Postman (n.d.) *Postman API Client.* Available from: https://www.postman.com/product/api-client/ (Date viewed: 21.05.2020)

Smith, S. (2020) *Overview of ASP.NET Core*. Available from: https://docs.microsoft.com/nb-no/aspnet/core/mvc/overview?view=aspnetcore-3.1 (Date viewed: 15.05.2020)

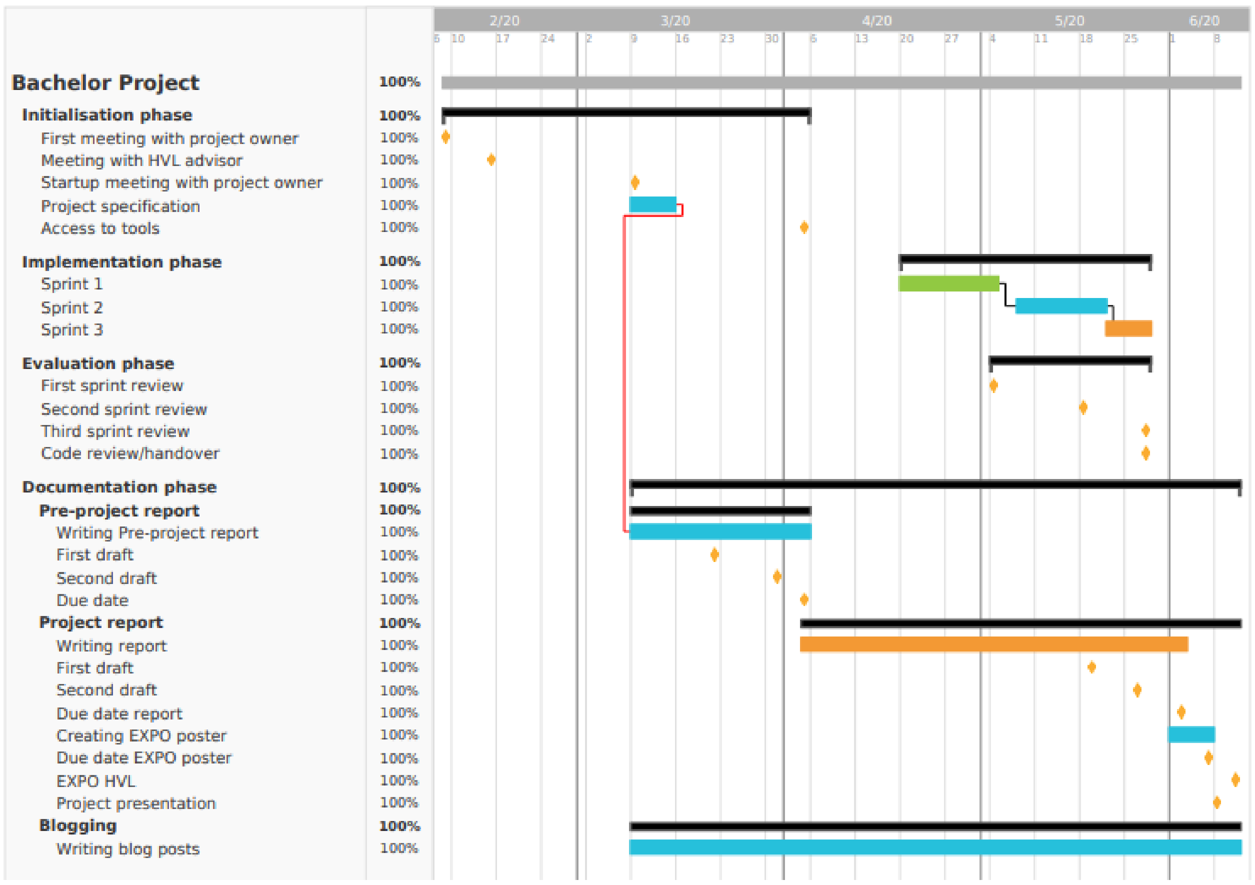
Stackify (2017) *What is RestSharp? An introduction to RestSharp's features and functionality.* Available from: https://stackify.com/restsharp/ (Date viewed: 28.05.2020)

Tutorialspoint (n.d.) *MS SQL Server Tutorial.* Available from: https://www.tutorialspoint.com/ms_sql_server/index.htm (Date viewed: 26.05.2020)

ZData (2020) *Om ZData.* Available from: https://www.zdata.no/om-zdata/ (Date viewed: 30.03.2020)

APPENDIX

Appendix A - Gantt chart



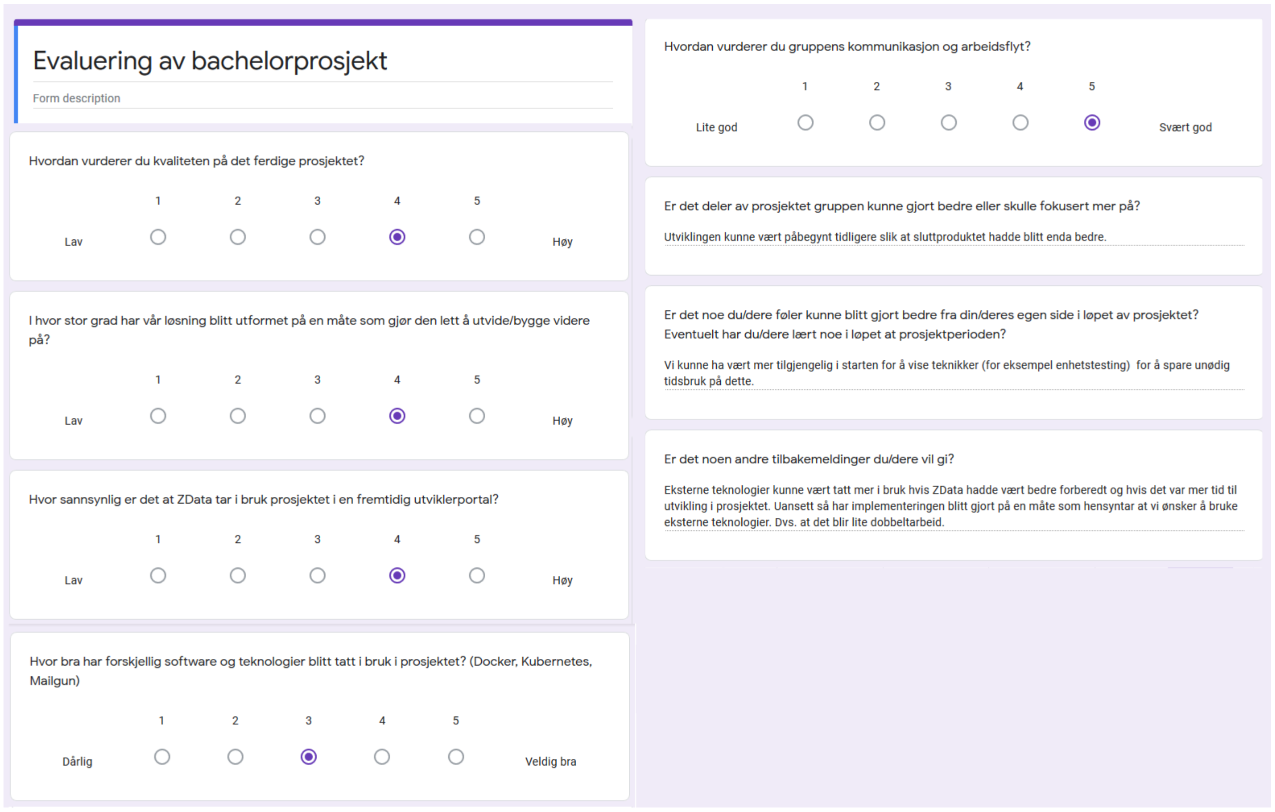
Appendix B – Risk analysis

Risk	**L**	**C**	**RF**	**Affected Groups**	**Phase**
Virus outbreak	4	5	20	Client, Project group	Startup phase, may be of concern throughout the entire project.
Inexperience	5	2	10	Client, Project group	Startup phase, development phase.
Time constraints	3	3	9	Client, Project group	Development phase.
Failure of equipment	2	3	6	Client, Project group	Development phase.
Misinterpreting project specification	3	2	6	Client, Project group	Startup phase, development phase.
Poor communication with client	3	4	12	Client, Project group	Startup phase, development phase.
Poor collaboration of group members	1	4	4	Client, Project group	Continuous.

L - Likelihood of risk
C - Severity of risk
RF - Risk factor (L multiplied with C)
Scale 1-5, where 1 is low and 5 is high

Appendix C – Evaluation form with response from ZData

After completing the project we sent a form with a few evaluation questions to ZData, so they could leave us some feedback.



Evaluering av bachelorprosjekt

Form description

Hvordan vurderer du kvaliteten på det ferdige prosjektet?

Lav 1 2 3 [✓ 4] 5 Høy

I hvor stor grad har vår løsning blitt utformet på en måte som gjør den lett å utvide/bygge videre på?

Lav 1 2 3 [✓ 4] 5 Høy

Hvor sannsynlig er det at ZData tar i bruk prosjektet i en fremtidig utviklerportal?

Lav 1 2 3 [✓ 4] 5 Høy

Hvor bra har forskjellig software og teknologier blitt tatt i bruk i prosjektet? (Docker, Kubernetes, Mailgun)

Dårlig 1 2 [✓ 3] 4 5 Veldig bra

Hvordan vurderer du gruppens kommunikasjon og arbeidsflyt?

Lite god 1 2 3 4 [✓ 5] Svært god

Er det deler av prosjektet gruppen kunne gjort bedre eller skulle fokusert mer på?

Utviklingen kunne vært påbegynt tidligere slik at sluttproduktet hadde blitt enda bedre.

Er det noe du/dere føler kunne blitt gjort bedre fra din/deres egen side i løpet av prosjektet? Eventuelt har du/dere lært noe i løpet at prosjektperioden?

Vi kunne ha vært mer tilgjengelig i starten for å vise teknikker (for eksempel enhetstesting) for å spare unødig tidsbruk på dette.

Er det noen andre tilbakemeldinger du/dere vil gi?

Eksterne teknologier kunne vært tatt mer i bruk hvis ZData hadde vært bedre forberedt og hvis det var mer tid til utvikling i prosjektet. Uansett så har implementeringen blitt gjort på en måte som hensyntar at vi ønsker å bruke eksterne teknologier. Dvs. at det blir lite dobbeltarbeid.