Inhoud

Version management

Version	Date	Description
1.0	09-01-2021	Added needed headers and filled in the content I could fill in.
1.1	17-01-2021	Added SonarCloud explanation.
1.2	18-01-2021	Added UX example for admin dashboard.
1.3	18-01-2021	Added explanation about the group project and where to find more information about this.

Introduction

In this portfolio I'm going to document my progression during semester 3. This is mainly for my individual project.

1. Learning outcome 1: Web Application

Full stack

My full stack application will contain the following:

- 3 front-ends (user dashboard, admin dashboard, Webshop) this will be made in Angular.
- A gateway for all the API connections I will be using Ocelot for this.
- Authentication and authorization by using Auth0.
- 4 separate components (order component, product component, user component, license component) I will be using ASP.NET Web API Core for this.
- A MSSQL database.

In order to use my application in a container to container level I've attempted to use a service bus for the database this was based on an article from Microsoft about microservices (see bibliography (Microsoft, 2021)). I eventually got this done, but left this aside for later on because I had more core related problems with my progression and decided that this would be to much of a distraction to fully implement.

At the start I struggled a lot with my architecture this was more my misunderstanding at the start what a microservice was eventually I got right but it took 3 revisions sadly enough.

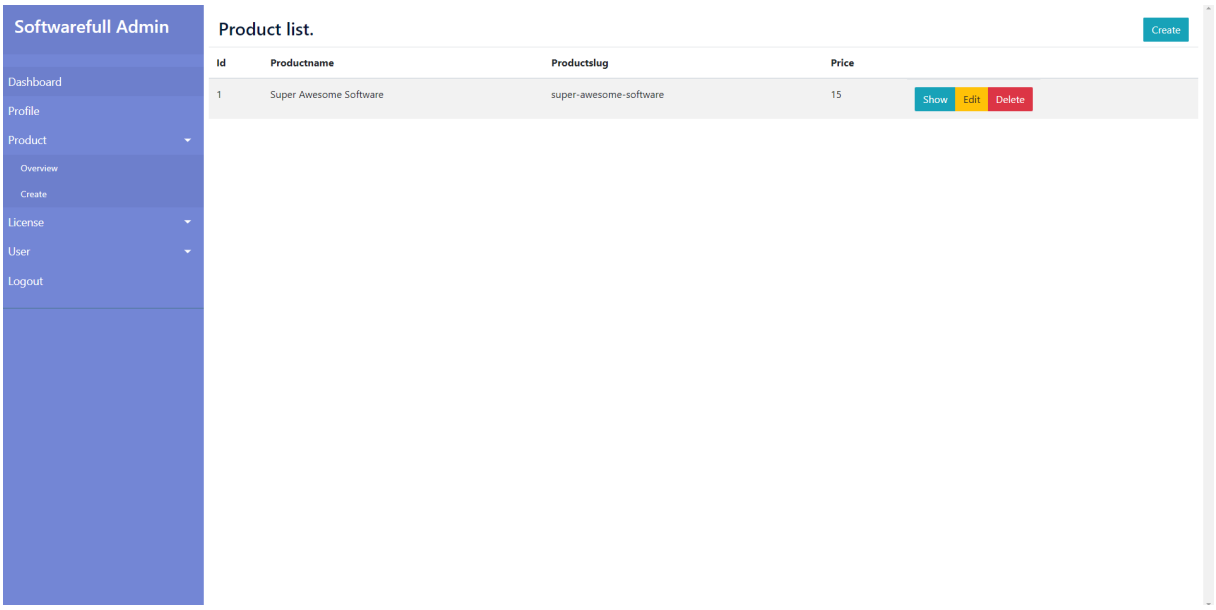
I started the project with the components and the communication between them this is because here where most of the new subjects for me especially microservices.

User-friendliness

User friendliness is an important aspect of an application but, I personally decided to wait with this because I first wanted to understand the microservices and everything related to them. Once a backend programmer always a backend programmer it seems 😊.

I improved my UX for the admin dashboard in order to make it more easy traverse in comparison to before by adding a sidebar instead of a regular navbar and adding easy to travers buttons for the

possible crud operations. I did this after receiving from both my classmates and teachers feedback about this.



Approach

Dataflow

The data within my application will follow the following route:

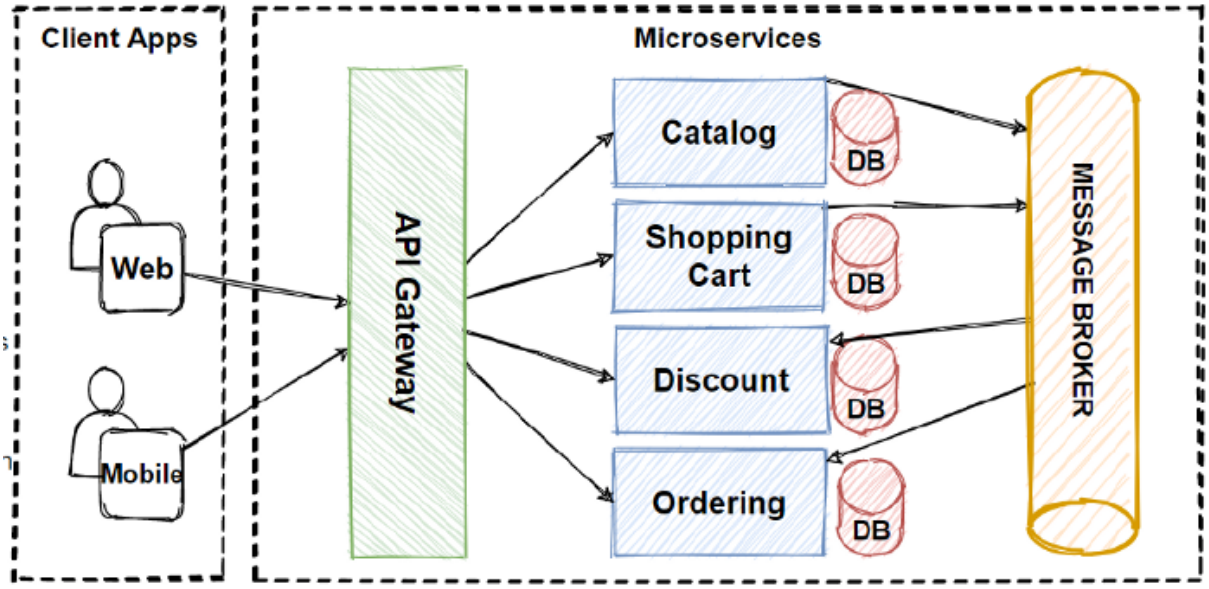
1. Database
2. API
3. Frontend

The above list looks simple but is just a simplified version of the flow I will at a later date add a diagram to visual show it for clarity.

Database

While investigating about microservices I noticed that a lot of developers who use them tend to use the following pattern “Database per services”. The short explanation for this is that a service must be loosely coupled so they can develop / scale independently by using a single database you can not achieve this. More about this you can find on the bibliography (Microservices.io, 2019) or a more recent article about it is (Özkaya, 2021).

I found the perfect diagram to explain the dataflow this from (Özkaya, 2021).



My data will be a bit different but the general concept is the same by using for each service a separate database we will be able to make it more scalable but also optimize the related processes for example for the products we could use just a MSSQL database but for searching for example Elasticsearch data storage. Different databases have different strong and weak points by optimizing this it is possible to further optimize the application.

API / WebSocket’s

Each of my components will also at the same time be an API that provides information to my Webshop and dashboards about their contents also components will communicate which each other bi-directionally using web sockets for example when creating a license.

2. Learning outcome 2: Tooling & Methodology

Tooling & Methodology

The main purpose of “Tooling & Methodology” is that the quality of the produced software stays consistent during the development process. This is achievable by doing regression testing for example.

ORM

I use ORM (Object Relational Mapping) for my database because keeping your database up to date is an important step in the development process. By using migrations I can keep track of this and revert the database potentially back if a problem occurs. Also ORM helps me with managing all the different databases that my application has because of the “Database per service” pattern.

Testing

Testing is an important part of the development process to keep your application consistent. There are a lot of different tests. I decided to use the following tests for my full stack application:

1. Unit tests
2. Integration tests
3. Component tests
4. End to end tests

Security

In regards to the many security risks related with developing an application I’ve decided to use Auth0 for my login and registration system by doing this I’m able to outsource potential risks and issues to a team of experts instead of a novice like me.

Auth0 makes use of JWT’s for doing the authentication and authorization of the user. By using these JWT in combination with Auth0 I’m able to authorize or deny users based on their tokens this is possible because I use in my gateway to my services Auth0 to check these tokens.

By isolating these services and only making them available by using the gateway I can make sure that only authorized users access them I achieve this by using scopes which are the permissions an user has inside the API.

More about Auth0 you can find in the bibliography (Auth0, sd).

3. Learning outcome 3: Design & Implementation

Design & Implementation

In order to fulfil this learning outcome I use the following:

1. GitHub (GitHub actions) for version management and CI.
2. Docker for deployment.
3. SonarQube for Cod coverage / management.

Version Management

GitHub is used by many developers for producing private and open source software. There are a lot of big projects on the platform this is possible because it keeps track of you’re application versions.

CI

What is CI?

The goal of CI is to control the code that is being pushed to make sure it doesn't break the current application. By using you're tests in combination with your CD in the CI you can check if you're application is still working. This is called a pipeline which gets called after each push to GitHub.

My pipeline

I chose GitHub actions for my pipeline because it allows me to push my docker images to my Docker hub repositories. I first tried this with Azure Devops where I could successfully setup my containers inside Azure it's own platform but after trying to push it to Docker hub what was my end goal I saw that it lacked version management of the plugins that supported Docker hub sadly enough so I moved to GitHub actions where I got it working without a problem.

SonarQube

SonarQube is a "Code Quality Assurance Tool", this tool will analyse the code in your project and check the quality of you're program. It helps with code coverage and code quality even the security of your application is taken into account. I've decided to setup SonarQube using SonarCloud by using this platform I'm able to connect my testing to my GitHub actions making code quality a part of my CI/CD.

Setting up SonarQube

Setting this up is really easy because SonarCloud has a good onsite tutorial with literally a workflow file pre written for you.

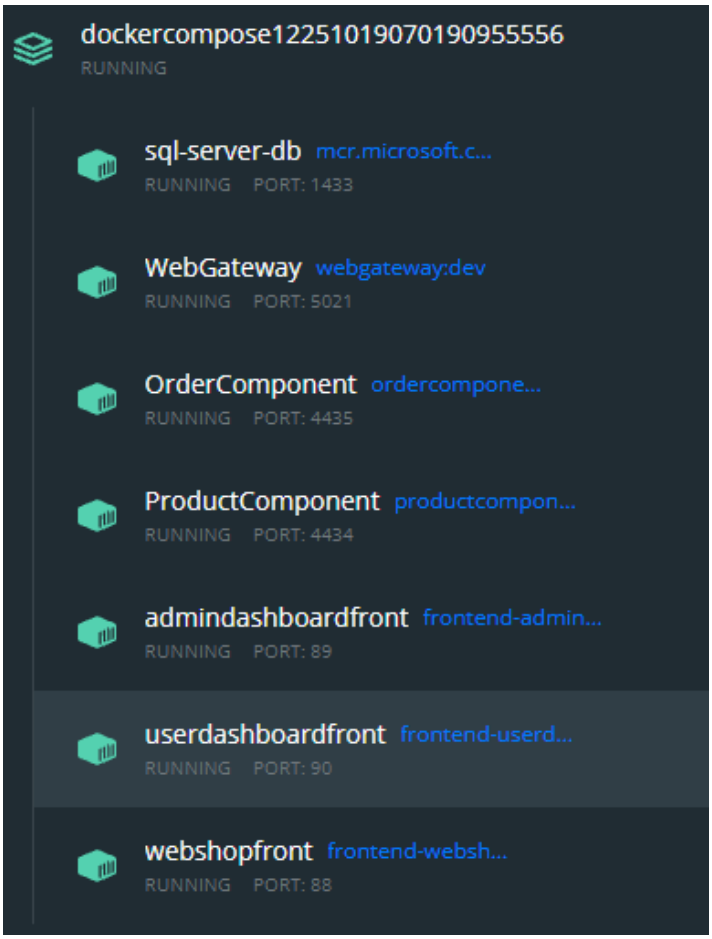
CD

Why docker?

Docker is used to distribute applications, so that your application can run at different hardware (By Docker is this called a "Container"). A container exists out of an image the main advantage of Docker is that it allows you're application to run at this image instead of you're direct pc preventing a lot of configuration issues related to the hardware of the applications. It is also easy to import and setup which make it a great tool for Open source software.

My implementation

For my Docker setup I choice docker-compose which allows me to create multiple containers with a click of button simply. You can see that I've got multiple containers running currently in a group the group gets created by docker-compose and based on the data I submit to the file I can bind specific ports to the container.



I will go through my docker-compose and my docker files to explain what is happening in order to produce this group.

The first file is the compose file.

```
webshopfront:
  container_name: webshopfront
  image: ${DOCKER_REGISTRY-}frontend-webshoplayer
  depends_on:
      - productcomponent
  build:
      context: ../SoftwareFull.WebshopFrontEnd
      dockerfile: ../SoftwareFull.WebshopFrontEnd/Dockerfile
  ports:
   - '88:80'
```

The first piece of code we got is the first container our front-end. The container name property is used to name the container as you can see looking at the two pictures. The image is the name given to this specific image by docker in my case it looks like this.

The depends_on property is used to decide if the container needs another container in my case this is my product component. Because I want to be able to get my products. The following property is the build parameter this specifies both in my case the context which is the root folder and the dockerfile contained in that folder at which we will view later on. At last we got our ports inside the dockerfile we can expose specific ports by exposing these and binding them to a free port on the hosting pc we will be able to access them on the same pc.

```
  productcomponent:
    image: ${DOCKER_REGISTRY-}productcomponent
    build:
      context: .
      dockerfile: CommunicationLayer/Components/ProductComponent/Dockerfile
    environment:
        ConnectionString: "Data Source=host.docker.internal,1433;Initial
Catalog=ProductComponent-6705ec26-21d6-4425-9120-1df6d6cb958a;User
ID=sa;Password=Root1234"
    ports:
      - "8081:80"
      - "4434:443"
    depends_on:
      - sqlserver
```

The above file is in many aspects the same as the front-end file but the main difference is that we add here an environment variable for our database connection inside docker. By doing this we are able to connect with the database but when run outside docker we will be using a different one.

```
  sqlserver:
    container_name: sql-server-db
    image: mcr.microsoft.com/mssql/server:latest
    user: root
    ports:
      - "1433:1433"
    environment:
      SA_PASSWORD: "Root1234"
      ACCEPT_EULA: "Y"
    volumes:
      - dbdata:/var/opt/mssql/data
```

In the above file we create the database inside docker I will be using mssql for this by using a hosted image created by Microsoft we are able to quickly obtain an empty database server image. The parameter user is specified to make the root-user. Then in the environment SA_PASSWORD is set which indicates the root users password. Then the ACCEPT_EULA is specified the skip the questionnaire. But the most important thing to prevent possible surprises is the volumes parameter this allows us to save the current data within the image separately which has the advantage that we will have no data loss at restart. At the last lines of the docker-compose you put the following.

```
volumes:
  dbdata:
```

This let's docker compose know about the volumes.

The frontend docker file

```
### STAGE 1: Build ###
FROM node:12.7-alpine AS build
RUN apk update && apk add git
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build

### STAGE 2: Generate self signed ssl cert ###
FROM node:12.7-alpine AS nodessl
### Needed this because nginx for some reason doesn't allow apk :)
RUN apk update \
```

```
    && apk add openssl
RUN mkdir -p /etc/nginx/ssl
RUN chmod 700 /etc/nginx/ssl
RUN openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/nginx/ssl/example.key -out /etc/nginx/ssl/example.crt -subj
"/C=NL/ST=Noord-Brabant/L=Wouw/O=Softwarefull/OU=IT Student/CN=example.com"

### STAGE 3: Run ###
FROM nginx:1.17.1
RUN mkdir -p /etc/nginx/ssl
RUN chmod 700 /etc/nginx/ssl
COPY --from=nodessl /etc/nginx/ssl /etc/nginx/ssl
COPY nginx.conf /etc/nginx/nginx.conf
COPY --from=build /usr/src/app/dist/login-demo /usr/share/nginx/html
EXPOSE 433
EXPOSE 80
```

At stage 1 I want to build and install the libraries on the image. Then in stage 2 I want to generate the ssl cert but this doesn't yet work for some reason I don't know. Then last but not least stage 3 where I create an nginx image and import the data from both stage 1 and 2.

Backend C# dockerfiles

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["CommunicationLayer/Components/ProductComponent/ProductComponent.csproj",
"CommunicationLayer/Components/ProductComponent/"]
RUN dotnet restore
"CommunicationLayer/Components/ProductComponent/ProductComponent.csproj"
COPY . .
WORKDIR "/src/CommunicationLayer/Components/ProductComponent"
RUN dotnet build "ProductComponent.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "ProductComponent.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "ProductComponent.dll"]
```

First in the above file I expose the ports for http and https. Then I will just like in the frontend import and install all packages and when this is finished cop them over to image folder. Then we will run the publish in order to publish the dotnet application. And last but not least we will put the content of the application from publish inside our final image and then execute the command "dotnet ProductComponent.dll" from the same folder causing our application to start running.

4. Learning outcome 4: Professionality

Personal project

See design, technical documents.

Group project

Inside the group project we worked in a agile way in order to fulfil the learning outcomes and to provide ourselves as a group with an overview for what still needs to be done or not. Each week at Monday we normally perform a stand-up to together in order to assign new tasks and reassign task that couldn't be completed the week before. For more information about this see our Group project document.

Bibliography

Auth0. (n.d.). *Secure access for everyone but not just anyone*. Retrieved from Auth0.com: Auth0.com

Microservices.io. (2019). *Database per service*. Retrieved from https://microservices.io: https://microservices.io/patterns/data/database-per-service.html

Microsoft. (2021, 09 12). *Introducing eShopOnContainers reference app | Microsoft Docs*. Retrieved from docs.microsoft.com: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/introduce-eshoponcontainers-reference-app

Özkaya, M. (2021, september 8). *Microservices Database Management Patterns and Principles*. Retrieved from medium.com: https://medium.com/design-microservices-architecture-with-patterns/microservices-database-management-patterns-and-principles-9121e25619f1