**Portfolio**

Portfolio of Damian Hoogsteder

Semester 3

Contents

[Introduction 2](#_Toc82532948)

[The Learning outcomes 3](#_Toc82532949)

[What do i want to achieve? 4](#_Toc82532950)

[User stories 4](#_Toc82532951)

[Requirements 5](#_Toc82532952)

[Architecture 7](#_Toc82532953)

[How am i going to realize the final product? 7](#_Toc82532954)

[Distributed software 7](#_Toc82532955)

[Front-end development 8](#_Toc82532956)

[Back-end development 8](#_Toc82532957)

[CI/CD 9](#_Toc82532958)

# Introduction

# The Learning outcomes

I’d like to give a quick rundown of the learning outcomes I will have to achieve this semester and how to achieve them.

I’ll be assigning multiple tasks and concepts to each learning outcome to give an indication of what i will have to achieve each sprint. These tasks will be updatet at the start of every sprint. This way I’ll have a good indication of the sprint backlog and progress I’m making compared to other sprints.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| LO ID | Sprint | Tasks Needed to achieve | Status | | Feedback | |
| 1 | *You design and build* ***user-friendly****,* ***full-stack*** *web applications****.*** | | | | | |
|  | *0* | *Researching Ideas and theory* | Doing (Through all iterations) | |  | |
|  | *1* | *Javascript framework* | DONE | |  | |
|  |  | UI | Doing | |  | |
|  | 2 | Deciding on a color scheme | DONE | |  | |
|  |  |  |  | |  | |
| 2 | *You use software* ***tooling and methodology*** *that continuously monitors and improve the software quality during software development.* | | | | | |
|  | 1 | *Testing plan* | To do | |  | |
|  |  | *Encapsulation* | To do | |  | |
|  | 2 |  |  | |  | |
| 3 | *You* ***design and implement*** *a (semi)automated software release process that matches the needs of the project context.* | | | | | |
|  | *0* | *Research ideas and designing a architecture.* | | Doing | |  |
|  | *1* | *Research about docker* | | To do | |  |
|  |  | *REST api design* | | To do | |  |
|  | *2* | *Setting up pipelines and running/ building tests and application to repository.* | | *To Do* | |  |
|  |  |  | |  | |  |
| 4 | *You act in a* ***professional manner*** *during software development and learning.* | | | | | |
|  | *0* | *Making sure I communicate my ideas with the teachers. This way I can prevent making something that can’t meet the Learning outcomes.* | | To Do | |  |
|  | *1* | *Asking and applying feedback* | | DONE | |  |
|  |  | *Sprint Review* | | DONE | | *Documented in feedpulse.* |
|  | *2* | *Asking for feedback* | | To Do | |  |

# What do i want to achieve?

The most obvious goals for this semester would be to achieve all the skills that are written in the learning outcomes. Besides this I would like to be able to understand the choices I’m making a little bit more and make decisions based on research and concious research.

On top of that i would like to create software I enjoy and have interest in. The main plan I have currently is making a online trading marketplace. The main focus for this marketplace will be digital items and ways of trading with them. I’m also interested in making it possible for artists to sell their work through blockchain. Non fungible tokens can be stored in a blockchain such as ethereum and this stored data can be show as an image, video or other form of media. This makes art into sort of a currency.  
On top of this I’d be interested in the trading of in game items such as skins or characters (Compared to like the steam market).

## User stories

Currently I have a pretty good idea what I want to make. But this is only an idea, by making user stories I’m trying to make sure that it is feasable and that I’m thinking of what functionalities are important and why they are neccesary.

These user stories are subject to change, but I want to have a clear as an image of what the end result is going to be before I start planning for the first real iteration of this semester.

**User story 1**As a User  
I want to be able to buy digital items in a central place  
So that I can acces a large assortment of items.

*Acceptance criteria:*

1. The item that is selected is sorted in price.

**User story 2**As a user  
I want to be able to compare prices of items  
So that I cansave as much money as possible

*Acceptance criteria:*

**User story 3**As a User  
I want to be able to sell all my digital items in one place  
So that i have a clear image of the items I am selling

*Acceptance criteria:*

1. Different kinds of digital items can be sold in the same place.
2. There is a place where all pending sell orders are shown.

**User story 4**As a user  
I want to have a screen where i can see my purchase and selling history  
So that I can look back at my purchases and sold items.

*Acceptance criteria:*

1. The user can see all records of previous purchaces
2. The user can see all of his/her sold items

**User story 5**As a user  
I want to be able to see other peoples opinion on the seller  
So that i can know whether they are trustworthy.

*Acceptance criteria:*

1. *The seller has a seller rating (A numeric display of the reviews about this seller)*
2. *The seller has a verified account.*
3. *The user is able to read other reviews and expiriences with the seller.*

**User story 6**As a user  
I want to be able to create an account  
So that i can trade and buy from other verified sellers

*Acceptance criteria:*

1. User can create an account on a seperate page

## Requirements

Before i start building a architecture for my application I’ll have to set up some requirements. These will be a little more descriptive and technical. And will go into the technical application of the user

***Functional requirements***

**FR-01** The user is able to put their digital items up for sale.

**K-01.1** Bids can be placed upon the item up for sale.  
**K-01.2** A suggested price can be made.  
**B-01.1** The digital item must be know to the system to ensure safe trading.  
**B-01.2** The user has to have a verified account.

**FR-02** The user is able to purchase a digital item from another user.

**B-02.1** The user has to have a verified account.

**FR-03** The user is able to trade digital items with another user.

**K-03.1**

**FR-04** The user is able to create a account.

**K-04.1** The password must contain a capitalized letter and at least 6 characters.  
 **K-04.2** Usernames must be unique to avoid fraud.  
 **B-04.1** All specified fields must be filled in.

**FR-05** The user is able to log in if he/she has a account.

**B-05.1** A account with the log in credentials must exist.

**FR-06** The user is able to view his/her account.

**FR-07** The user is able to change his/her account.

**B-07.1** The user must be logged in.

**FR-08** The user can view his/her purchase history

**K-08.1** A message is shown if there are no purchases.

**B-08.1** The user must be logged in.

**FR-09** The user can view his/her sell history.

**K-09.1** A message is shown if there are no sold items.

**B-09.1** The user must be logged in.

**FR-10** The user is able to see information about the seller when trading or purchasing.

**FR-11** The user is able to give feedback and review a seller that he/she traded with.

**FR-12** The user is able to see a numeric rating of the seller.

**K-GEN-01** Proper error messages are shown when something goes wrong.

***Non-functional requirements***

**NFR-01** The application backend will be ASP.NET.

**NFR-02** The application is distributed in multiple smaller applications.

**NFR-03** The frontend is made using a Javascript framework, like Angular or React.

**NFR-04** A proper UX is taken in to account.

# Architecture

## How am i going to realize the final product?

### Distributed software

This semester one of the most important goals, in my opinion, would be to properly distribute my software into multiple small applications that each function on their own and could run on a seperate server and hardware.

The frontend will be made on a seperate server than the backend. And the backend will be split up into smaller applications that have their own specific function and don’t have to worry about their interactions with other parts of the application.

Currently I will split my back-end up into 3 different applications. This might become more in the future, but this is what I think is feasable.

1. TradingService  
   This ‘’service’’ will handle the initializisation of the trades. It will make sure the trade is succesfull and the items ownership has been changed.
2. AccountService  
   This service will verify the account and keep tabs of their activity. Being able to view and modify a account will also be the responsibility of this service.
3. MarketService  
   This services will be responsible for the market place itself and the functionality to add new items and put them up for sale.

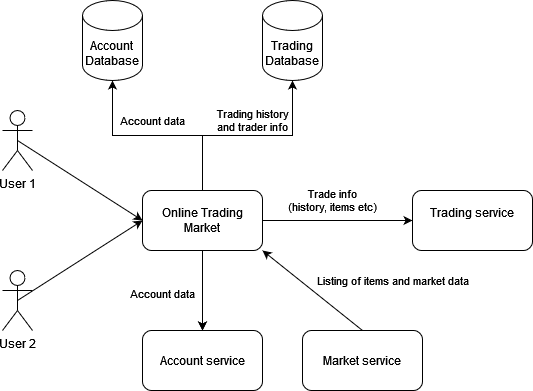
These will all be put in a seperate docker container and will communicate with each other through interfaces and API’s.

#### DOcker research

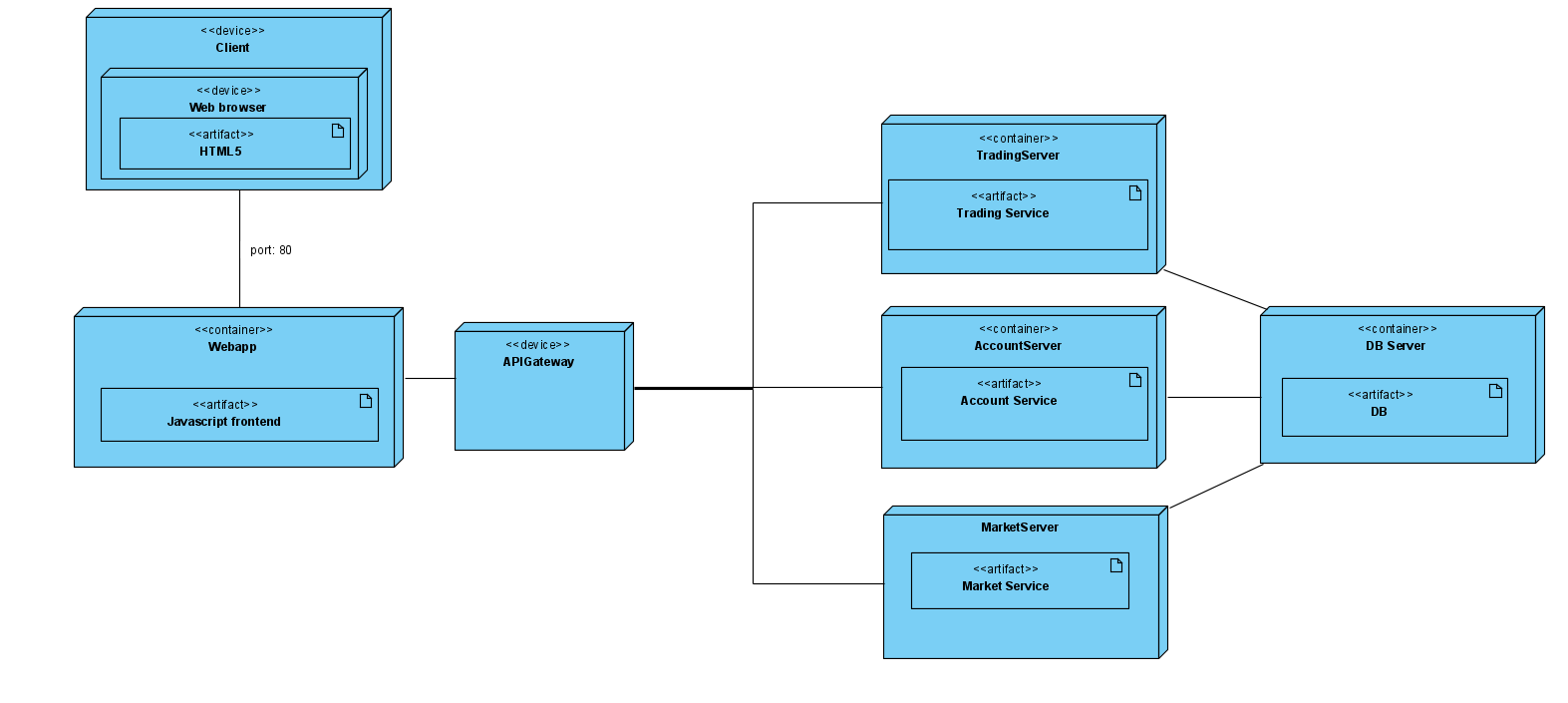
Docker is a software that allows me to seperate a part of my computer that is completly isolated. These isolated spaces are called containers. Since the container is completly isolated it does not use your computers OS. That’s where images come in, to make a container be able to function properly there needs to be something that controls what is going on in the container.

Now each part of my distributed software will run in a seperate container. Since they are isolated from each other I need to figure out a way for my software to communicate with each other. Using API’s (Application programming interface) it is possible to make HTTP requests to send and recieve data from one container to another. API’s define the functionality that is locked behind it. I’ll go further into API’s and API design later.

#### System context diagram



#### Deployment diagram



### Front-end development

#### Which javascript framework will i be using?

There are several javascript frameworks that each have their own positives and negatives. There are several aspects I’m taking into account while deciding which frameworks fits my assignment best. While trading and interacting with other users speed is a thing I’m taking into account.

Before deciding what front-end framework i will be using I need to set up a couple criteria.  
Based on these criteria I’ll make a decision.

***Acceptance criteria***

1. There needs to be enough recources to study and learn on my own.
2. The framework needs to be fast and reliable.
3. Trades shouldn’t be dependent on speed of the client.

Some of the well know javascript frameworks are:

1. AngularJS

Angular is a open source javascript framework and is implemented for the development of SPA’s (Single Page Applications) and extends HTML into the application. Angular is cross platform and promises to deliver high speed and performance through the use of server-side rendering. By rendering everything server side, speed is not affected by the client (in this case the browser). Another positive is that the community around AngularJS is very big and the documentation is expansive and clear. This makes it easier to get into and makes the chance of finding solutions easier.

1. React  
   React is a javascript framework made by Facebook. It is made with the intention of creating dynamic UI that allows large amounts of traffic. It uses a virtual DOM wich makes it easier to intergrate the front—end into the rest of your application. Facebook has made a large documentation about React and its tutorials are supposedly very good.
2. Node.js  
   Node.js is a open source server-side javascript runtime enviroment. Node is fast in its code excecution, due to the usage of of the V8 javascript version of google chrome.
3. Vue.js

Vue.js also uses a virtual DOM. Due to this the user is able to make changes that are implemented in real time. Vue also makes use of dual integration which makes it possible to quickly make SPA’s.

1. Ember.js

Ember is a trustworthy platform that uses two way databinding. This makes it possible for the user to directly makes changes from the frontend to the component class. Due to this there is no reloading necessary. Ember also makes it easy to implement complicated user interfaces.

1. Polymer.js

Polymer is a javascript library that utilizes both one-way and two-way databinding. This makes it a very large and expansive library that offers a great deal of possibilities. It is developed by google and makes it possible to create and use website elements without having to add a lot of complexity.

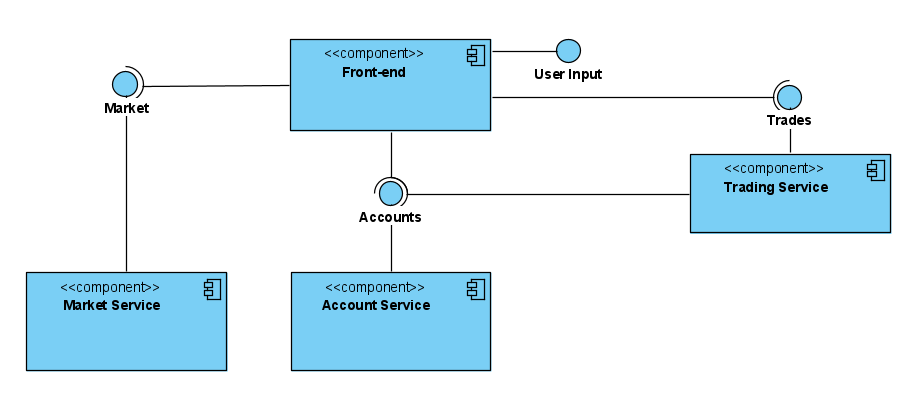
After looking through the different frameworks and comparing them to my acceptance criteria, I’ve found angular to be the most fitting. This will thus be the framework I will use.

### Back-end development

#### Which techniques will i be using?

I have decide to work with a c# backend.

#### COmponent diagram



#### Api Descriptions

**MarketService**

**TradeService**

**AccountService**

#### RESTful APi design

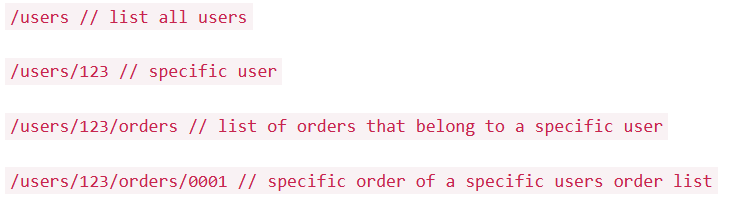
***What is a RESTful API?***

An API is a layer between software application. API’s are made to send requests which will return a response. API’s are used to communicate through the whole internet. An API is RESTful when it is implemented according to the rules and principles of REST (Representational State Transfer). This is the underlying architectural principle of the web.

RESTful API is a service that follows these rules (hopefully) and uses [HTTP methods](https://code-maze.com/the-http-reference/#requestmethods) to manipulate the set of resources.

**Best practices**

I’ll be covering a couple best practices to make my api’s readable and universal. This will make using and managing the API a lot easier down the line.

1. *JSON requests*  
   Api’s should both accept and send requests in a JSON format. This is the universally used formatting for processing and sending data. Pretty much all modern HTTP clients have libraries that can decode JSON and show it properly without any issues.
2. *Using proper status and Respons codes*  
   Status codes are able to fetch pretty much any errors. They are also much easier to identify and integrate into your api. This makes implementing error handling very easy. I’ll be going into status and respons codes further down.
3. *Proper routing*  
   There are a couple of best practices to make routes more readable and understandable. Try to refrain from using verbs in url’s and use nouns instead. If you’re talking about a collection of items or markets use plural nouns like ‘markets’ or ‘items’.
4. *Resource nesting*Resource nesting is the way you extend your url into separate pages that all extend from the same key word.  
   Here’s a good example:  
   
5. *Security*Things like encrypting communication with SSL/TSL (Secure Sockets Layer / Transport Layer Security). are pretty much a must. And while handling sensitive data you must consider probable databreaches. So protecting the network is very important. I’ll be going into this further in my research report.

**HTTP verbs**

HTTP defines a bunch of request methods that tell the client what it needs for a given resource.

Here are a couple of HTTP verbs:

1. GET
2. HEAD
3. POST
4. PUT
5. DELETE
6. PATCH

**Status and respons codes**

HTTP response status codes indicate whether a specific [HTTP](https://developer.mozilla.org/en-US/docs/Web/HTTP) request has been successfully completed. Responses are grouped in five layers:

1. [Informational responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#information_responses) (100–199)
2. [Successful responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#successful_responses) (200–299)
3. [Redirection messages](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#redirection_messages) (300–399)
4. [Client error responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses) (400–499)
5. [Server error responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#server_error_responses) (500–599)

*Src:* [*https://developer.mozilla.org/en-US/docs/Web/HTTP/Status*](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

**Web sockets**

|  |  |  |
| --- | --- | --- |
| Accounts | Market | Trades |
|  | GET /markets |  |
|  | GET  /markets/{id} |  |
|  | GET  /markets/{id}/items |  |
|  | GET  markets/{id}/item/{id} |  |

### CI/CD