

## **New-Act Documentation**

### **Project 4.0**

**Carlos Duarte Barbosa - Nicholas Coorevits - Tristan Berkman**

**Ehran Lenaerts - Wout Deelkens - Adam Abid**

<b>Introduction .....</b>	<b>3</b>
<b>1 Application .....</b>	<b>4</b>
1.1 WebShop .....	4
1.2 Contentful .....	4
1.3 Adyen .....	6
1.4 Algolia.....	7
<b>2 Infrastructure .....</b>	<b>9</b>
2.1 Cloud environment .....	9
2.2 Virtual Private Cloud .....	10
2.3 Beanstalk.....	12
2.4 Relational Database Service MySQL .....	15
2.5 EC2 Instances .....	17
2.6 Jenkins Pipeline .....	18
2.7 HTTPS & Domain Name .....	24
<b>3 Conclusion .....</b>	<b>25</b>
<b>4 Credentials.....</b>	<b>26</b>
4.1 Contentful .....	26
4.2 Webshop .....	26
4.3 Jenkins.....	26
4.4 SonarQube .....	26

# Introduction

For the company Elision we created a composable e-commerce website, for small to medium sized businesses rather than using a one-size-fits-all e-commerce functionality to meet business needs, using the nextJS react framework together with a Java backend and Microservices. This ensures that businesses can respond to today's ever-changing dynamic marketplace.

Our team is divided into 2 main categories, the application side where the students from APP and AI worked together on the front and back-end part of the project and the infrastructure side where the CCS students worked on the infrastructure part. This document contains all the technologies used by team New Act to create a composable e-commerce project.

# 1 Application

In this section we'll give an explanation of how we completed the application side of the project. The front end was done with NextJS which is integrated with Contentful and Adyen. The backend was done in Java which is used to implement Adyen and get data from our users and their shopping cart.

## 1.1 WebShop

For the web shop or rather the site itself we used, as mentioned before, the NextJS react framework which implements Contentful and Algolia. With the help of our Java back-end we were also able to implement Adyen for our shopping cart transactions. When first entering the site at:









<https://newact.technisme.com/> you will be greeted by the home page where we show you the categories of products available to us and provided via Contentful. Under the categories we have a list of trending items which will show the most viewed products. Next by clicking on one of the product categories it will redirect you to the product page which will list all products in cards. These cards will show you the price and name of the product as well as if it's out of stock which will be marked by the text in red "Out of stock!". Clicking on either the image or the details button will bring you to the detail page where more information is given about that specific product, here you can also add the item to your shopping cart. Once in the shopping cart you can clear them from the cart if you're not satisfied otherwise you can go to the checkout provided by Adyen. In the checkout we have 4 options to pay which all work as intended.

If you want to register for an account, you can do so by pressing on the person icon in the top right corner and clicking under the login button on "Register". Here you can give in your credentials which will be saved. Next you can try logging in with the information you have used to register.

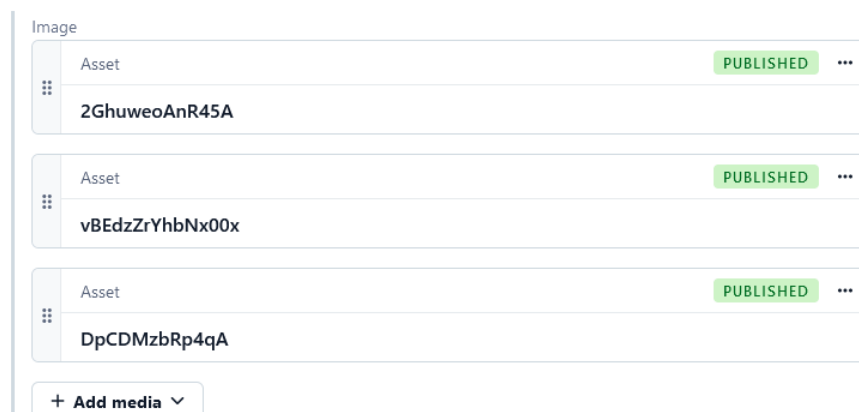
## 1.2 Contentful

Contentful is a content management system that allows its users to deliver content in a flexible and scalable manner. We set up an account on their site and began with creating different content models.





Since we are using Contentful as our main product database on top of its CMS properties we need to make sure that the content models we create have linked reference fields within them. The 'Category' content model is made first and will only have two fields: A name and an image linked to it. This category image is a media field in which we can then place an image asset. The next content model we require is the 'Specification' so that we can provide different information for each product, however since this information could be shared between different products, we would like to keep it in a database to be used later. Once again, this content model is rather simple as we only need two fields for the name of the specification and its value.

⋮	 <b>Product name</b>	Long text	Entry title	<a href="#">Settings</a>	⋮
⋮	 <b>Brand</b>	Short text		<a href="#">Settings</a>	⋮
⋮	 <b>Price</b>	Decimal number		<a href="#">Settings</a>	⋮
⋮	 <b>Image</b>	Media, many files		<a href="#">Settings</a>	⋮
⋮	 <b>Stock</b>	Integer		<a href="#">Settings</a>	⋮
⋮	 <b>Descriptions</b>	Long text		<a href="#">Settings</a>	⋮
⋮	 <b>Categories</b>	References, many		<a href="#">Settings</a>	⋮
⋮	 <b>Specifications</b>	References, many		<a href="#">Settings</a>	⋮

As seen above for our product content model we use many to many relationships for both categories and specifications but also allow for multiple image files so that we can show different pictures of a product on our shopping page.



For each product the customer can add multiple images which will be leveraged on the Webshop for the individual product carousels on the detail pages. We also create different content models for pages with information that can change over time such as the promotion page where we can choose different products to display or the 'about us' page where the customer can place information about their business or services.

⋮	 <b>Page Title</b>	Short text	Entry title	<a href="#">Settings</a>	⋮
⋮	 <b>About Us</b>	Long text		<a href="#">Settings</a>	⋮
⋮	 <b>What we do</b>	Long text		<a href="#">Settings</a>	⋮
⋮	 <b>Company brand</b>	Media		<a href="#">Settings</a>	⋮

The 'About Us' page can simply be adjusted to always show the up-to-date information about the company. Unpublishing items within Contentful will result in these items disappearing from the






Webshop, giving the customer easier control over the content that they can display. Customers can also unpublish whole categories in case they wish to pivot their focus.

<input type="checkbox"/>	Name	Content Type	Updated <sup>▲</sup>	Last Updated By	Status	⚙
2 entries selected:						
<input type="button" value="Duplicate"/> <input type="button" value="Unpublish"/> <input type="button" value="Add or remove tags"/> <input type="button" value="Export as CSV"/>						
<input type="checkbox"/>	High-pressure cleaners	Category	a few seconds ago	Me	PUBLISHED	...
<input type="checkbox"/>	Laptops, desktops & monitors	Category	27 Jan 2023	Me	DRAFT	...
<input checked="" type="checkbox"/>	Laptops	Category	26 Jan 2023	Me	PUBLISHED	...
<input checked="" type="checkbox"/>	Mobile phones	Category	26 Jan 2023	Me	PUBLISHED	...
<input type="checkbox"/>	Drones	Category	26 Jan 2023	Me	PUBLISHED	...
<input type="checkbox"/>	Wireless speakers	Category	26 Jan 2023	Me	PUBLISHED	...

## 1.3 Adyen

Adyen is an online payment solution that allows merchants to accept a variety of payment methods on their website or mobile app. It is designed to simplify the payment process and create a seamless experience for the customer.

For Adyen we use a Java API linked to an Adyen test account. In our test environment we have 4 different payment methods, which we can alter in our administration page for Adyen.

<input type="button" value="Payment methods"/> <input type="button" value="Status"/> <input type="button" value="Currencies"/> <input type="button" value="+ More filters"/> <input type="button" value="Reset all"/>								
<input type="button" value="Default column config"/> ▼								
<input type="checkbox"/>	Payment method	TX variant	Configured currencies	Available countries	Issuer countries	Status	Not usable reason	Acq
<input type="checkbox"/>	 Bancontact	bcmc	EUR	ANY	-	Active	-	-
<input type="checkbox"/>	 Bancontact app	bcmc_mobile	EUR	ANY	-	Active	-	-
<input type="checkbox"/>	 iDeal	ideal	EUR	NL	-	Active	-	-
<input type="checkbox"/>	 Mastercard	mc	ANY	ANY	-	Active	-	-
<input type="checkbox"/>	 Visa	visa	ANY	ANY	-	Active	-	-

In the developer tab we can also regenerate our API keys in case they have been compromised. Keep in mind however that the old key will remain available for another 24 hours.

## Server settings

Authenticate the API requests that you make to Adyen

### Authentication

API key   Basic auth

---

**API key**

\*\*\*\*\*t?&D

Generate API key

**JWS public key**

Add public key

When a customer wants to make a payment, they are presented with a payment form that is hosted by Adyen. Upon loading the checkout page the payment methods will be requested from Adyen through our back-end, displaying them within their own container. After choosing a payment method and filling in the details the payment information will be securely transmitted to Adyen's servers for processing. This process runs via our Java back-end where it will complete the transaction or provide an error message in case the payment was unsuccessful. Upon a successful payment the Webshop will show a 'Success' message and allow the customer to return to the homepage.

Elision\_...022\_TEST  
Elision

FAVORITES (0)  
★ No favorites yet  
Click on the star on the right side of a menu item to add as a favorite.

PAGES  
Home  
Transactions  
Offers  
Payments  
Finance  
Insights  
Reports

Payments - Default   Search...

Amount   Card / account number   Currency   Date: Last 180 days   Merchant account: All (1)   Status   Store   More filters   Reset all

PSP reference	Merchant reference	Account	Date	Amount	Payment method
D4VP6278VMK2WN82	bd05b6ed-1cf7-4976-9038-8a87cef42872	Elision_Stage2022_TEST	Feb 20, 2023, 23:44:32	EUR 10.00	VISA Visa
Z5JVR558VMK2WN82	payment: TimGrevendonk@gmail.comdfc60e8e-c275-43...	Elision_Stage2022_TEST	Feb 20, 2023, 21:57:12	EUR 1,797.00	VISA Visa
WRGXBM8VMK8NKGK82	payment: TimGrevendonk@gmail.com	Elision_Stage2022_TEST	Feb 20, 2023, 21:54:24	EUR 1,340.94	VISA Visa
F2XXN748VMK2WN82	502	Elision_Stage2022_TEST	Feb 20, 2023, 21:03:43	EUR 1,000.00	VISA Visa
T9863F6VK8NKGK82	a4488846-eee7-44a4-a6ee-0acadda19839	Elision_Stage2022_TEST	Feb 20, 2023, 19:46:48	EUR 10.00	VISA Visa

Adyen has access to and keeps track of all the transactions that are processed through your account. This means that Adyen provides the customer with a comprehensive view of all transactions, which can be useful for various purposes, such as tracking sales, monitoring payment trends, and identifying potential issues or fraud.

Adyen's transaction overview typically includes details such as the transaction date and time, the payment method used, the amount charged, and any related fees or charges. This information can be accessed through Adyen's online dashboard or reporting tools, which would allow customers to filter through and analyze the data in various ways.

## 1.4 Algolia

We are using Algolia to provide search functionality. On every page, there is a search bar at the top from where search can be accessed. There is also a dedicated search page, which can be reached by hitting enter during a search or clicking "view more results" at the bottom of the results. The dedicated search page provides the additional features of filtering by category or brand, limiting the price range, and viewing multiple pages of results. User events and purchases are also noted to Algolia so that it can provide recommendations and trending items. Trending items can be seen on

the home page and recommendations can be found on product pages. These events can also be viewed as analytics on Algolia's dashboard.



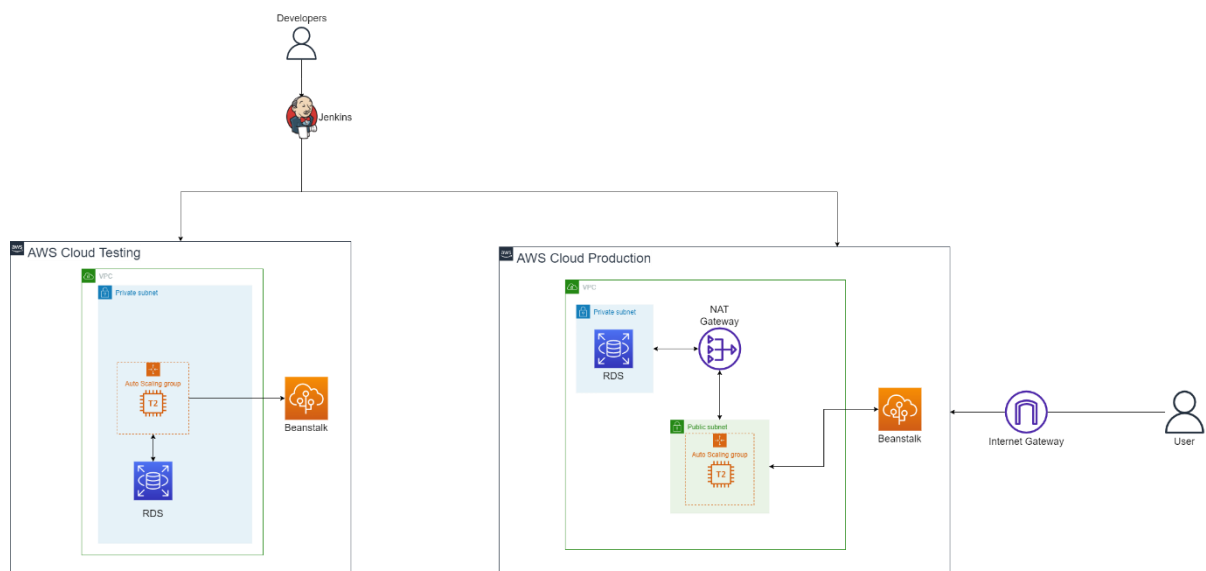
## 2 Infrastructure

Here we'll explain how we completed the infrastructure. Everything was done in AWS. We used the following AWS resources:

- Beanstalk
- Virtual Private Cloud with NAT gateway and an internet gateway
- Auto Scaling Group
- EC2 instances
- RDS MySQL database

We already gave a reason to why we used these resources but in this document we'll explain how we got them working and what workflow we used. We'll also use some screenshots to show a bit of more context on how we did things and provide some example code for our pipeline later in the document.

### 2.1 Cloud environment



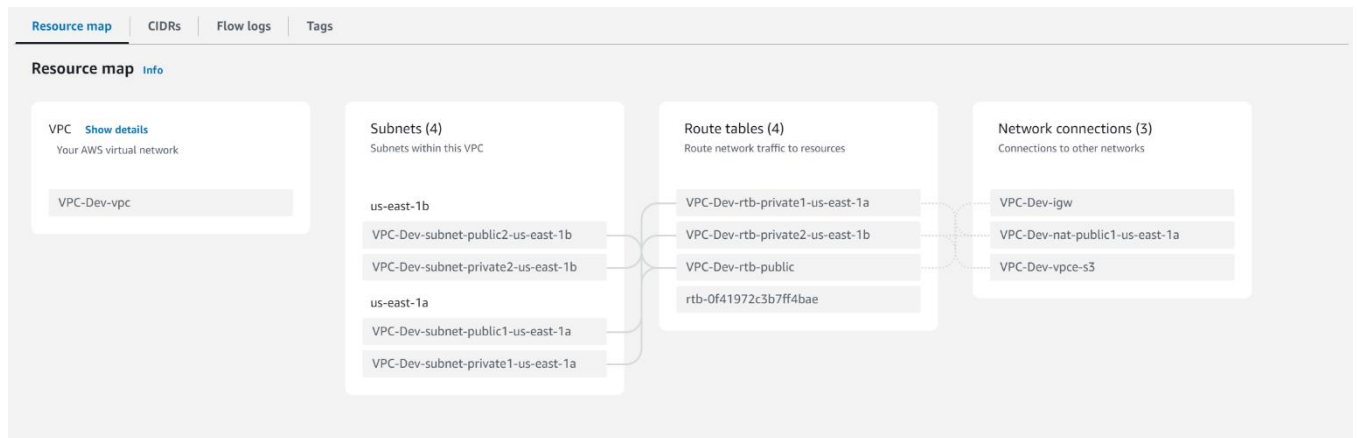
This is the AWS architecture that we ended up with, you can clearly see the mentioned aws resources that we use. We'll explain how we setup each resource and in what order, you can also see Jenkins but this will be explained later on in the document.

In the picture you can see that we use 2 different environments, one for testing and one for production. The testing environment is created first, this environment is not production ready and mistakes or bug are frequent, the main reason for this is to make sure no bugs are spread to the production environment.

The production environment will be created after we're production ready, in this environment will be publicly available and will also have some extra resources. In the production environment the user will access the website through an Internet Gateway, this is connected to the beanstalk that is hosting the frontend. The frontend beanstalk will then use the backend beanstalk for the API calls, only the backend is connected to the Private MySQL database, nothing else is connected to it. We'll now explain the different steps we took to accomplish this architecture

## 2.2 Virtual Private Cloud

The very first resource we create is the Virtual Private Cloud, here we'll also create 2 subnets one public and one private, together with these subnets we'll also create a NAT Gateway. This NAT Gateway will then automatically translate private addresses to public addresses and vice versa.



You can see the VPC configuration in the picture above, we've created 4 subnets 2 public and 2 private. We created 2 for each because we are using a Load Balancer for our beanstalks, this is to allow us to have High Availability.

We have 4 route tables, route tables are tables with all our IP addresses, we have 4 for each subnet. The private route tables are connected to the NAT gateway and are assigned a public address, the public addresses are connected to the Internet Gateway, this will allow the users to access our beanstalk.

<input type="checkbox"/>	Name	Subnet ID	State	VPC	IPv4 CIDR
<input type="checkbox"/>	VPC-Dev-subnet-public1-us-east-1a	subnet-0782123b5c73ae596	Available	vpc-04accb853ead8bafe   VPC-Dev-vpc	10.0.0.0/20
<input type="checkbox"/>	VPC-Dev-subnet-private2-us-east-1b	subnet-0057b437013f0bd84	Available	vpc-04accb853ead8bafe   VPC-Dev-vpc	10.0.144.0/20
<input type="checkbox"/>	VPC-Dev-subnet-public2-us-east-1b	subnet-045e72eebe77827a4	Available	vpc-04accb853ead8bafe   VPC-Dev-vpc	10.0.16.0/20
<input type="checkbox"/>	VPC-Dev-subnet-private1-us-east-1a	subnet-06e5a52ccb0a64d4a	Available	vpc-04accb853ead8bafe   VPC-Dev-vpc	10.0.128.0/20

Here we have the different subnets and their CIDR (Classless Inter-Domain Routing), it is of course important that each Subnet is available. These subnets are then automatically connected and attached to the VPC we created, this means that when we add a resource to this VPC we then can choose one of these subnets.

<input type="checkbox"/>	Name	Route table ID	Explicit subnet associat...	VPC
<input type="checkbox"/>	VPC-Dev-rtb-private1-us-east-1a	rtb-0ff580d28b539bd94	subnet-06e5a52ccb0a6...	vpc-04accb853ead8bafe   VPC-Dev-vpc
<input type="checkbox"/>	VPC-Dev-rtb-private2-us-east-1b	rtb-01ee5e181304a05e7	subnet-0057b437013f0...	vpc-04accb853ead8bafe   VPC-Dev-vpc
<input type="checkbox"/>	VPC-Dev-rtb-public	rtb-089cb6d566d78f050	2 subnets	vpc-04accb853ead8bafe   VPC-Dev-vpc

These are all our route tables that are connected to the VPC, these route tables will actually assign a IP address to the resource in this VPC. Route Tables contain the IP addresses, these IP addresses are either public or private and will be assigned to the resource you add to this VPC.

<input type="checkbox"/>	Name ▾	Internet gateway ID ▾	State ▾	VPC ID ▾
<input type="checkbox"/>	VPC-Dev-igw	<a href="#">igw-0471b9f50a2f9fc65</a>	✔ Attached	<a href="#">vpc-04accb853ead8baf6</a>   <a href="#">VPC-Dev-vpc</a>

This is the Internet Gateway attached to the VPC, each VPC has one Internet Gateway. The public subnets are connected to this Internet Gateway, this will allow us to have the application public.

<input type="checkbox"/>	Name ▾	NAT gateway ID ▾	Connectivit... ▾	State ▾	Primary public I... ▾	Primary private IPv4 address
<input checked="" type="checkbox"/>	VPC-Dev-nat-public1-us-east-1a	<a href="#">nat-00de4144a992d28f6</a>	Public	✔ Available	<a href="#">35.168.90.88</a>	10.0.10.199

One of the other things that are created with the VPC are the NAT Gateways, we have one Gateway. This NAT Gateway will allow the translation of the private addresses with public addresses.

Now we have a fully operational VPC, we now can add all the needed resources to this VPC.

## 2.3 Beanstalk

After we have a VPC ready we create 2 beanstalks that will host our frontend and backend, this will be done separately. When creating a beanstalk there are two things that will be created with it, the environment which we keep our application versions and the application itself. A beanstalk will automatically create a EC2 instance which will be used to host the actual application, when uploading an application to your beanstalk we call it a version.

Whenever a version doesn't work beanstalk will automatically go back to the working version, later in the document we'll explain more how we uploaded the application to beanstalk in our pipeline, but first we'll show all the beanstalk configurations.

All environments										
<input type="text" value="Filter results matching the display values"/>					<a href="#">Actions</a>		<a href="#">Create a new environment</a>			
Environment name	Health	Application name	Date created	Last modified	URL	Running versions	Platform	Platform state	Tier name	
Backend-env	Severe	Backend	2023-01-28 16:29:01 UTC+0100	2023-02-16 20:26:11 UTC+0100	<a href="#">Backend-env.eba-t38xikgm.us-east-1.elasticbeanstalk.com</a>	NewAct v167	Corretto 17 running on 64bit Amazon Linux 2	Supported	WebServer	
Frontend-env	Ok	Frontend	2023-01-20 10:26:06 UTC+0100	2023-02-16 20:25:40 UTC+0100	<a href="#">FrontendB1.us-east-1.elasticbeanstalk.com</a>	NewAct v167	Docker running on 64bit Amazon Linux 2	Supported	WebServer	

This is our 2 beanstalk environments, again one for backend and one for frontend. As you can also see the environment is linked to an actual application which is the EC2 instance, you are also presented with a URL. This URL is not secure and does not use https which is why we don't use, more on that later. Under running versions you can see the name of the most recent version, this name needs to be identical, next to the version you have the platform.

The platform is very important this decides what platform you use to run your app, we decided to put our frontend in a Docker image so that's why we use a docker platform for the frontend and for the backend we just a java platform which is named Corretto 17, this just runs the needed java versions and technologies. You can also see the Health of the environment, this is will give us information if the application crashed or not.

Our backend is showing 'Severe' this is because our backend uses a lot of API calls and this can interfere with the Elastic Load Balancer health check, this means that everything works but it just sometimes shows a severe health check.

For each beanstalk we created an elastic load balancer, we'll show them in more detail. Both look the same, both listen on port 443 for https and we also added our certificate to it so that it's actually encrypted with SSL, but more information on the domain name later in the document.

Listeners					
You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your environment processes. By default, we've configured your load balancer with a standard web server on port 80.					
					Actions ▾
					+ Add listener
<input type="checkbox"/>	Port	Protocol	SSL certificate	Default process	Enabled
<input type="checkbox"/>	443	HTTPS	technisme.com - 1770418e-5a64-4e51-baa9-ab34277b6491	default	<input checked="" type="checkbox"/>
<input type="checkbox"/>	80	HTTP	--	default	<input checked="" type="checkbox"/>

Listeners					
You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your environment processes. By default, we've configured your load balancer with a standard web server on port 80.					
					Actions ▾
					+ Add listener
<input type="checkbox"/>	Port	Protocol	SSL certificate	Default process	Enabled
<input type="checkbox"/>	443	HTTPS	technisme.com - 1770418e-5a64-4e51-baa9-ab34277b6491	default	<input checked="" type="checkbox"/>
<input type="checkbox"/>	80	HTTP	--	default	<input checked="" type="checkbox"/>

The picture above is the Application Load Balancer for the backend and the one below is the load balancer for the frontend, both use https and are assigned a SSL certificate by listening on port 443.

Virtual private cloud (VPC)	
VPC	vpc-04accb853ead8baf6 (10.0.0.0/16)   VPC-Dev-vpc

---

Virtual private cloud (VPC)	
VPC	vpc-04accb853ead8baf6 (10.0.0.0/16)   VPC-Dev-vpc

Both beanstalks are in the same VPC the frontend uses only public subnets, while the backend uses a private and a public subnet. The reason for this is that the backend is connected to the MySQL server through the private subnets, when creating a beanstalk you get the option to create a Database with it.

Database	Endpoint: <a href="https://awseb-e-myefkijryrn-stack-awsebrdsdatabase-bre8zgrfwxqe.cugiikt8ffqi.us-east-1.rds.amazonaws.com:3306">awseb-e-myefkijryrn-stack-awsebrdsdatabase-bre8zgrfwxqe.cugiikt8ffqi.us-east-1.rds.amazonaws.com:3306</a> Availability: Low (one AZ) Engine: mysql Instance class: db.t2.micro Retention: Create snapshot Storage: 5 Username: admin
----------	--

This is the database connected to it, only the backend EC2 instance can connect to this database, so no other system or machine can connect to it. Only the Beanstalk backend and the Jenkins agent can connect to it more on that later.

Every EC2 has it's one Security Group that dictates what can or cannot connect to them, this has been done for the EC2 instances for the beanstalks. This will be done so that not everything can connect to them

Inbound rules (4)									
<input type="text" value="Filter security group rules"/> <span>&lt; 1 &gt;</span>									
<input type="checkbox"/>	Name	Security group rule...	IP version	Type	Protocol	Port range	Source		
<input type="checkbox"/>	-	sgr-0547ae68bf1ee3db5	-	HTTP	TCP	80	sg-0896ffade3375		
<input type="checkbox"/>	-	sgr-032b18a6b41026...	IPv4	HTTPS	TCP	443	0.0.0.0/0		
<input type="checkbox"/>	-	sgr-0c9a0ea9a04e595cb	IPv4	SSH	TCP	22	0.0.0.0/0		
<input type="checkbox"/>	-	sgr-03c12cb834f36d4c1	IPv4	HTTP	TCP	80	0.0.0.0/0		

Outbound rules (1/1)									
<input type="text" value="Filter security group rules"/> <span>&lt; 1 &gt;</span>									
<input checked="" type="checkbox"/>	Name	Security group rule...	IP version	Type	Protocol	Port range	Destination		
<input checked="" type="checkbox"/>	-	sgr-01909d74ea4ee65ef	IPv4	All traffic	All	All	0.0.0.0/0		

This are the inbound and outbound rules for the backend EC2 instance, only the most important ports are available like port 80 and 443, this is of course to allow http and https for our application. Port 22 is for SSH so that we can securely connect to our instance, you can see also a second port 80 and this is for health checks of the elastic load balancer.

Inbound rules									
Inbound rules (4)									
<input type="text" value="Filter security group rules"/> <span>&lt; 1 &gt;</span>									
Security group rule...	IP version	Type	Protocol	Port range	Source	Description			
sgr-0a175fb35beab641b	IPv4	HTTPS	TCP	443	0.0.0.0/0	-			
sgr-0fb1e8eabed587e61	-	HTTP	TCP	80	sg-09d320c11f47b0e8...	-			
sgr-0ae49b540f2ea1612	IPv4	SSH	TCP	22	0.0.0.0/0	-			
sgr-03c4cf44f39edbc2	IPv4	HTTP	TCP	80	0.0.0.0/0	-			

Outbound rules									
Outbound rules (1/1)									
<input type="text" value="Filter security group rules"/> <span>&lt; 1 &gt;</span>									
<input checked="" type="checkbox"/>	Name	Security group rule...	IP version	Type	Protocol	Port range	Destination		
<input checked="" type="checkbox"/>	-	sgr-02ca3966e009ea0c8	IPv4	All traffic	All	All	0.0.0.0/0		

These are the inbound and outbound rules of the frontend EC2 instance, which are basically the same.

This is all the important configuration of all our beanstalks, the application deployment will be discussed later in the document where we'll also show some code examples and the actual automated procedure of the deployment.

The main importance of beanstalk is that you don't need to think about the infrastructure side and only need to deploy the actual code which makes it easier for developers, so beanstalk is actually PAAS (Platform As A Service). You still get the option to configure the needed infrastructure which is why it is very interesting to use Beanstalk and it really fits our application.

## 2.4 Relational Database Service MySQL

Together with our backend beanstalk we created a RDS database, this RDS database is only connected to the backend. The database is in the same VPC we created before and only uses private subnets, just like the EC2 instances it also has a security group that decides what resources are connected to it.

Connectivity & security		
Endpoint & port	Networking	Security
Endpoint awsdb-e-myefkjyryn-stack-awsebrdsdatabase-bre8zgrfwxqe.cugikt8ffqi.us-east-1.rds.amazonaws.com	Availability Zone us-east-1a	VPC security groups rds-ec2-4 (sg-00e80a037c4c2d702) Active rds-awseb-e-myefkjyryn-stack-awsebrdsdbsecuritygroup-1bq0khua1to3p-Sigc (sg-0767def48855e31d9) Active
Port 3306	VPC VPC-Dev-vpc (vpc-04accb85ead8bafef)	Publicly accessible No
	Subnet group awsdb-e-myefkjyryn-stack-awsebrdsdbsubnetgroup-jqah3hjgiw7q	Certificate authority <a href="#">Info</a> rds-ca-2019
	Subnets subnet-0782123b5c73ae596 subnet-0057b437013f0bd84 subnet-045e72eebe77827a4 subnet-06e5a52ccb0a64d4a	Certificate authority date August 22, 2024, 19:08 (UTC+02:00)
	Network type IPv4	DB instance certificate expiration date August 22, 2024, 19:08 (UTC+02:00)

This is the database configuration, the endpoint is basically the address/hostname of the RDS and the port is on which port the RDS is available. You can also see that the database is connected to the same VPC as the other resources, it also has all the private subnets.

Under security we have all our security groups for this database, you can also see that this database is not publicly accessible.

Security group rules (3)		
<input type="text" value="Filter by security group rules"/>		
Security group	Type	Rule
rds-awseb-e-myefkjyryn-stack-awsebrdsdbsecuritygroup-1bq0khua1to3p-Sigc (sg-0767def48855e31d9)	EC2 Security Group - Inbound	sg-0f16f3f481fc6b49f
rds-awseb-e-myefkjyryn-stack-awsebrdsdbsecuritygroup-1bq0khua1to3p-Sigc (sg-0767def48855e31d9)	CIDR/IP - Outbound	0.0.0.0/0
rds-ec2-4 (sg-00e80a037c4c2d702)	EC2 Security Group - Inbound	sg-0bdf79ddf0597dde5

These are all the security groups, two for different EC2 instances and one for the outbound rules. Only 2 instances can securely connect to this database, they can do so because they are in the same VPC and are allowed because of the security groups of the database.

Backup

Automated backups

Enabled (1 Day)

Copy tags to snapshots

Disabled

Backup target

AWS Cloud (US East (N. Virginia))

Latest restore time

February 19, 2023, 13:45 (UTC+01:00)

Backup window

07:30-08:00 UTC (GMT)

Replicate to Region

-

Replicated automated backup

-

Snapshots (2)

RestoreRemoveTake snapshot

Filter by snapshot name

Snapshot name

Snapshot creation time

Status

Snapshot type

Snapshot database time

rds:awseb-e-myefkjyryn-stack-awsebrdsdatabase-bre8zgrfwxqe-2023-02-18-07-39

February 18, 2023, 08:39 (UTC+01:00)

Available

Automated

-

rds:awseb-e-myefkjyryn-stack-awsebrdsdatabase-bre8zgrfwxqe-2023-02-19-07-39

February 19, 2023, 08:39 (UTC+01:00)

Available

Automated

-

Our database is also equipped with automated backups, every day at 7:30 a backup will be taken. Each backup or snapshot will be kept for a full day, you can always go back one snapshot.

This is all our important configuration of the database, we created this database automatically with the backend beanstalks so they were automatically attached with each other.



## 2.5 EC2 Instances

We know have explained all our important set up, but we also decided to add some other instances that would help us achieve the full project. We added a SonarQube server to one of our instances, this server is available to the public but protected with a strong password.

We also have Jenkins server in an EC2 instance, Jenkins will be our pipeline. To the Jenkins server we added an agent which will build our backend, this agent is also in its own a EC2 instance. There is also a webhook EC2 instance which acts as a webhook for contentful, and of course there also backend and frontend EC2 instances but these have already been explained.

<input type="checkbox"/>	Instance ID	Instance state ▾	Instance type ▾	Status check	Availability Zone ▾	Public IPv4 DNS ▾	Public IPv4 ... ▾	Security group name ▾
<input type="checkbox"/>	<a href="#">i-0786ba1ebae2df4a1</a>	Running	t2.medium	2/2 checks passed	us-east-1a	ec2-52-204-216-176.co...	52.204.216.176	ec2-rds-3,ec2-rds-4,ec2-rds-...
<input type="checkbox"/>	<a href="#">i-0a33bd4a11df3c102</a>	Running	t2.micro	2/2 checks passed	us-east-1a	ec2-54-221-31-123.co...	54.221.31.123	awseb-e-myefkjrjn-stack-A...
<input type="checkbox"/>	<a href="#">i-081d5af0d036febe4</a>	Running	t2.micro	2/2 checks passed	us-east-1a	ec2-3-95-214-167.com...	3.95.214.167	awseb-e-ixqzbmkwa-stack-...
<input type="checkbox"/>	<a href="#">i-0985785a58d146179</a>	Running	t2.medium	2/2 checks passed	us-east-1d	ec2-34-205-241-126.co...	34.205.241.126	Jenkins
<input type="checkbox"/>	<a href="#">i-078ca9e764efb76a0</a>	Running	t2.medium	2/2 checks passed	us-east-1c	ec2-52-205-118-4.com...	52.205.118.4	SonarQube
<input type="checkbox"/>	<a href="#">i-02217a0ae152a41e3</a>	Running	t2.micro	2/2 checks passed	us-east-1e	ec2-54-226-6-157.com...	54.226.6.157	launch-wizard-4

## 2.6 Jenkins Pipeline

We've shown how we've setup everything in the cloud but it's time to automate the whole DevOps process by using a Jenkins pipeline, we've already shown you where we've setup our Jenkins server.

Now we're going to show all the different steps of our pipeline and why we set them up, the pipeline is triggered by a webhook in the bitbucket repository. This is triggered when the developer pushes something to the specified branch in our case it's the main branch.

Title \*

Jenkins Pipeline

URL \*

http://34.205.241.126:8080/bitbucket-hook/

Status

☒ Active  
Inactive webhooks don't trigger requests.

SSL/TLS

☒ Skip certificate verification  
Untrusted or self-signed certificates may not be secure. [Learn more](#)

### Triggers

Repository	Issue	Pull request
<input checked="" type="checkbox"/> Push	<input type="checkbox"/> Created	<input type="checkbox"/> Created
<input type="checkbox"/> Fork	<input type="checkbox"/> Updated	<input type="checkbox"/> Updated
<input type="checkbox"/> Updated	<input type="checkbox"/> Comment created	<input type="checkbox"/> Approved
<input type="checkbox"/> Commit comment created		<input type="checkbox"/> Approval removed
<input type="checkbox"/> Build status created		<input type="checkbox"/> Changes Request created
<input type="checkbox"/> Build status updated		<input type="checkbox"/> Changes Request removed
		<input checked="" type="checkbox"/> Merged
		<input type="checkbox"/> Declined
		<input type="checkbox"/> Comment created
		<input type="checkbox"/> Comment updated
		<input type="checkbox"/> Comment deleted

This is the configuration of our webhook, in the URL we put the Jenkins URL with the bitbucket hook. In the triggers we checked the push box and the merged box so that the webhook only works in these cases.

These are all the pipeline stages that we have created:

	Git Checkout	Login to ecr	Build Frontend	Test-Image	Push Frontend	Deploy Frontend	Clean Up Workspace	build Backend	SonarQube Scan	Quality Gate	Deploy Backend	Workspace Clean Up
Average stage times: (Average full run time: ~3min 12s)	13s	943ms	1min 29s	6s	29s	3s	1s	10s	10s	106ms	8s	96ms
#167 feb. 16 2022 1 commit	26s	900ms	1min 26s	6s	30s	4s	1s	11s	15s	110ms (paused for 1s)	8s	104ms

#### - Stage 1: Git Checkout

- In this stage the webhook was triggered and the server will get the contents from the repository in bitbucket, everything from the repository will be copied to the Jenkins server. It's also important to note that we pass this stage as an artifact so that we then can use it for the backend agent, this will basically copy the stage.

#### - Stage 2: Login to ECR

- In this stage the Jenkins server will login into the elastic container registry, we need this to make sure we later on can upload our docker image.

#### - Stage 3: Build Frontend

- Here the frontend will be built with help of a Dockerfile that we provide, this stage will build the docker image. The docker image has the frontend all the needed configuration files, this image will then be pushed to the ECR.

#### - Stage 4: Tests Image

- After the image is created, we will test the image, these tests consist of basic file checks. They will make sure that .env file is actually there together with other files like the config.js file.

#### - Stage 5: Push Frontend

- Finally in this phase the image will be pushed to the ECR, this will make sure we then can deploy our application to beanstalk. We then also need a Dockerrun.aws.json file, this file is uploaded to the beanstalk and will run our docker image that is hosted in the ECR.

#### - Stage 6: Deploy Frontend

- In this stage our frontend gets deployed to the beanstalk, each beanstalk shares one S3 bucket where it keeps the contents of the versions. We upload our Dockerrun.aws.json file to this S3 bucket and then upload this file to the beanstalk env, we do this with the aws cli command, this will be more clear when we show the Jenkins pipeline script.

#### - Stage 7: Workspace Clean up

- After the frontend is done we clean up our workspace, this first part is done on the server itself the rest of the pipeline is done on the backend agent.

- **Stage 8: Build Backend**
  - In this stage we jump to the backend agent, here we also pass our previous artifact to this stage so that we can work with the same files in the repository. We build our backend here with the correct mvn command, here we don't use a Dockerfile. After the build is done, we have a .jar file which will be important in the next stages.
- **Stage 9: SonarQube Scan**
  - In this stage we do the needed SonarQube scan which will be send to our SonarQube server, in the server a result will be kept.
- **Stage 10: Quality Gate**
  - In this stage we check our SonarQube scan and we make sure there actually was a 80% coverage, if it's not the case the pipeline will stop. Nothing else will be done and the backend code will not be uploaded.
- **Stage 11: Deploy Backend**
  - This stage will upload our backend code to the beanstalk, this follows the same procedure as the frontend. The backend just uses java so we'll upload the .jar file to the S3 bucket using the aws cli command and then we upload it to the beanstalk using again aws cli.
- **Stage 12: Workspace Clean up**
  - After everything is done the pipeline cleans up the agent workspace, this will delete everything in the agent workspace.

After these stages the application should be up and running, this process is of course done automatically. This pipeline runs because of a script in the Jenkins server, the script look like this:

```
node{
  stage('Git Checkout'){
    git credentialsId: 'CarlosDB', url: 'https://bitbucket.org/xplorgroup/thomas-more-1-2023', branch: 'develop'
    stash 'git-checkout'
  }
  stage('Login to ecr'){
    script {
      sh "aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/13a2b0k8"
    }
  }
  stage('Build Frontend'){
    script {
      dir('./frontendProject/'){
        sh 'pwd'
        sh 'cp .env.local.example .env.local'
        sh 'sed -i "s+space_id+${CONTENTFUL_SPACE_ID}+g" .env.local'
        sh 'sed -i "s+access_token+${CONTENTFUL_ACCESS_TOKEN}+g" .env.local'
        sh 'sed -i "s+preview_access_token+${CONTENTFUL_PREVIEW_ACCESS_TOKEN}+g" .env.local'
        sh 'sed -i "s+preview_secret+${CONTENTFUL_PREVIEW_SECRET}+g" .env.local'
      }
      dockerFrontend = docker.build("public.ecr.aws/13a2b0k8/project4.0:frontend","-f ./frontendProject/Dockerfile .")
    }
  }
  stage('Test-Image'){
    script{
      try {
        def status = 0
        status = sh(returnStdout: true, script: '../container-structure-test test --image 'public.ecr.aws/13a2b0k8/project4.0:frontend' --config '/var/lib/jenkins/docker-tests/container-structure-test.yml' --output json | jq .Fail') as Integer
        if (status != 0) {
          error 'Image Test has failed'
        }
      } catch (err) {
        error "Test-Image ERROR: The execution of the container structure tests failed, see the log for details."
        echo err
      }
    }
  }
  stage('Push Frontend'){
    script{
      sh "docker push public.ecr.aws/13a2b0k8/project4.0:frontend"
    }
  }
  stage('Deploy Frontend'){
    script{
      sh "aws s3 cp ./frontendProject/Dockerrun.aws.json s3://elasticbeanstalk-$AWS_REGION-$USER_ID/Dockerrun.aws.json"
      sh "aws elasticbeanstalk create-application-version --application-name 'Frontend' --version-label 'NewAct v${BUILD_NUMBER}' --source-bundle S3Bucket=elasticbeanstalk-$AWS_REGION-$USER_ID,S3Key=Dockerrun.aws.json"
      sh "aws elasticbeanstalk update-environment --application-name 'Frontend' --version-label 'NewAct v${BUILD_NUMBER}' --environment-name 'Frontend-env'"
    }
  }
  stage('Clean Up Workspace'){
    script{
      echo 'Clean up local docker images'
      sh 'docker system prune -af'
      echo 'Clean up config.json file with ECR Docker Credentials'
      cleanWs()
    }
  }
}

node('Backend_node'){
  stage('build Backend'){
    unstash 'git-checkout'
    script{
      dir('./backendProject/'){
        sh 'sudo mvn package'
      }
    }
  }
  stage('SonarQube Scan'){
    script{
      withSonarQubeEnv('SonarServer1'){
        dir('./backendProject/'){
          sh 'sudo mvn sonar:sonar -Dsonar.projectKey=${Sonar_ProjectKey} -Dsonar.host.url=${SonarQube_URL} -Dsonar.login=${SonarQube_Login}'
        }
      }
    }
  }
  stage('Quality Gate'){
    timeout(time: 1, unit: 'HOURS') {
      def qg = waitForQualityGate()
      if (qg.status != 'OK') {
        error "Pipeline aborted due to quality gate failure: ${qg.status}"
      }
    }
  }
  stage('Deploy Backend'){
    script{
      sh "aws s3 cp ./backendProject/target/backendProject-0.0.1-SNAPSHOT.jar s3://elasticbeanstalk-$AWS_REGION-$USER_ID/backendProject-0.0.1-SNAPSHOT.jar"
      sh "aws elasticbeanstalk create-application-version --application-name 'Backend' --version-label 'NewAct v${BUILD_NUMBER}' --source-bundle S3Bucket=elasticbeanstalk-$AWS_REGION-$USER_ID,S3Key=backendProject-0.0.1-SNAPSHOT.jar"
      sh "aws elasticbeanstalk update-environment --application-name 'Backend' --version-label 'NewAct v${BUILD_NUMBER}' --environment-name 'Backend-env'"
    }
  }
  stage('Workspace Clean Up'){
    script{
      cleanWs()
    }
  }
}
```

This is our pipeline script we use some environment variables for sensitive strings and you can clearly see that we use aws cli commands to upload our code to the beanstalk. To give more context on the pipeline we're going to show important files we used to deploy our application to beanstalk.

/ frontendProject /

idea

.vscode

api





components

pages

public

services

styles

.env.local.example	19 feb. 2023 15:16:48	159 B		
.gitignore	19 feb. 2023 15:16:48	343 B		
Dockerfile	19 feb. 2023 15:16:48	232 B		
Dockerrun.aws.json	19 feb. 2023 15:16:48	188 B		
next.config.js	19 feb. 2023 15:16:48	572 B		
package.json	19 feb. 2023 15:16:48	775 B		
package-lock.json	19 feb. 2023 15:16:48	188.42 KB		
README.md	19 feb. 2023 15:16:48	1.70 KB		

 (alle bestanden in een zip-archief)

This the frontend folder structure, only the frontend folder uses extra files so that's why we only show them. In here the most important files are the Dockerfile which will be used to create a docker image of the frontend and the Dockerrun.aws.json file which has all the needed information to pull the docker image from the ECR, these files look like this:

```
FROM node:18
WORKDIR /frontendProject

COPY ./frontendProject/package-lock.json ./
COPY ./frontendProject/package.json ./
RUN npm install

COPY ./frontendProject/ .

RUN npm run build

EXPOSE 8000
ENV PORT 8000
CMD ["npm", "start"]d
```

This is the docker file which copies al the needed folder/files to run the actual frontend, it also exposes the port 8000. Port is very important because on a Docker beanstalk only port 8000 is open for applications, this port is then also specified in the Dockerrun.aws.json file.

```

frontendProject > {} Dockerrun.aws.json > [ ] Ports > {} 0
1  {
2      "AWSEBDockerrunVersion": "1",
3      "Image": {
4          "Name": "public.ecr.aws/l3a2b0k8/project4.0:frontend"
5      },
6      "Ports": [
7          {
8              "ContainerPort": 8000
9          }
10     ]
11 }

```

This is how the Dockerrun.aws.json file looks like, in this file we mainly specify the image name and from what repository we get it from. We also specify the port which is again port 8000, this file will actual run the docker container from that image hence the name Dockerrun.aws.json.

In AWS we also created a ECR which is called an elastic container registry, this registry keeps all our docker images that we need to upload to our frontend beanstalk. We have a public registry that we use for our frontend, we did this because we then can easily pull our image from that registry. The registry looks like this:

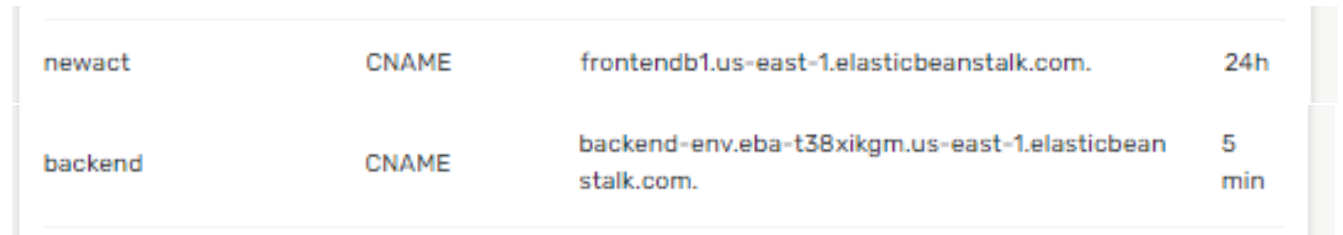
The screenshot shows the AWS ECR console interface. At the top, there are tabs for 'Private' and 'Public'. Below this, a section titled 'Public repositories (1)' contains a search bar and a table with one repository: 'project4.0'. The repository's URI is 'public.ecr.aws/l3a2b0k8/project4.0' and it was created on '23 januari 2023, 20:49:31 (UTC+01)'. Below the repository list, the 'project4.0' repository is selected, showing 'Images (23)'. A search bar for artifacts is present. A table lists the images, with one visible: 'frontend', which is an 'Image' type, pushed on '16 februari 2023, 20:24:51 (UTC+01)', with a size of '543.76' MB. The image URI is 'Copy URI' and the digest is 'sha256:676a85bfa35c3...'.

This is how our pipeline is setup, everything works automatically because of this pipeline. The only thing that needs to be changed are the aws credentials, these credentials always get reset after 36 hours, that's why we always have to change. This procedure cannot be automated as it would be a very big security risk and we also MFA (Multi Factor Authentication), which means every time we change our credentials we need a different MFA code which is always random.

## 2.7 HTTPS & Domain Name

To get https on our application, we need to own a domain name. One of our team members had one on another provider(vimexx.nl).

The domain called technism.com we created a subdomain named newact. This will be the URL for our website. [Newact.technisme.com](https://Newact.technisme.com)



newact	CNAME	frontendb1.us-east-1.elasticbeanstalk.com.	24h
backend	CNAME	backend-env.eba-t38xikgm.us-east-1.elasticbeanstalk.com.	5 min

As you can see on the above screenshot we used CNAME to make newact. We also made one for the backend. We wanted that all data was SSL encrypted.



### 3 Conclusion

In this project we learned a lot about working in team and how you correctly use the agile and scrum methodology, this was enlightening to many of the team members. During the project we also learned how you could work under pressure and stress which gave us a real look into to the real work environment of a IT company.

Using all these technologies was also very helpful which can give us a huge advantage later in our careers. We did get stuck in some areas like the login of the users, this is the reason we fell short on the login. Other then that we have accomplished the most important aspects of our application and infrastructure.

On the infrastructure side things didn't run smoothly at first because we didn't know how we should implement our pipeline, but luckily, we got it working. We as team personally thought this project was challenging but also fun.

This accompanied with a lot of restarting was a bit of annoying but we did accomplish what were the most important things, as a team we're proud of our teamwork and are also very happy we got to do something this challenging.

## 4 Credentials

### 4.1 Contentful

Username: r0791262@student.thomasmore.be

Password: C0ntentful!

### 4.2 Webshop

Username: [LexWatkins@hotmail.com](mailto:LexWatkins@hotmail.com)

Password: admin1234

### 4.3 Jenkins

URL: <http://34.205.241.126:8080/>

Username: TeamB1

Password: NEW-ACTB1

### 4.4 SonarQube

URL: <http://52.205.118.4:9000>

Username: admin

Password: Team@b1