Elder Guardians

Visi-Age Project

Prolonging independent elderly care
with computer vision

Salih Ekici

Tibo Geeraerts

Matthias Heylen

Brent Peeters

Lucas Syroit

Hube Van Loey

mediaan.

Table of contents

Introduction

In this documentation, we are going to take you through the project. The realisation consists of different parts such as AI, APP and CCS. For everything to work properly, these parts need to be perfectly aligned to make the project work. This is where we are happy to explain more.

We would like to give another brief description of the assignment to refresh it. The aim of the project is to help older people living alone at home live more independently. Through the cameras and the manual button on the iPad, the older person can still get help when needed. This can of course be from their confidant or a caregiver. They will have access to various functionalities in the application. What is also important is the person's well-being. This can be checked through a dashboard that shows the person's daily movement. If anything noteworthy is detected here, action can be taken by the caregiver or confidant.

The AI section explains how they use keypoint detection to detect the falls and how they track the person's movement.

The App section consists of 2 major parts, namely front-end and back-end. It explains how to set up an angular project for the front-end and what things to install. For the back-end, it also explains how best to set it up in Java. Of course, you can always download the repo and then only a few things need to be done.

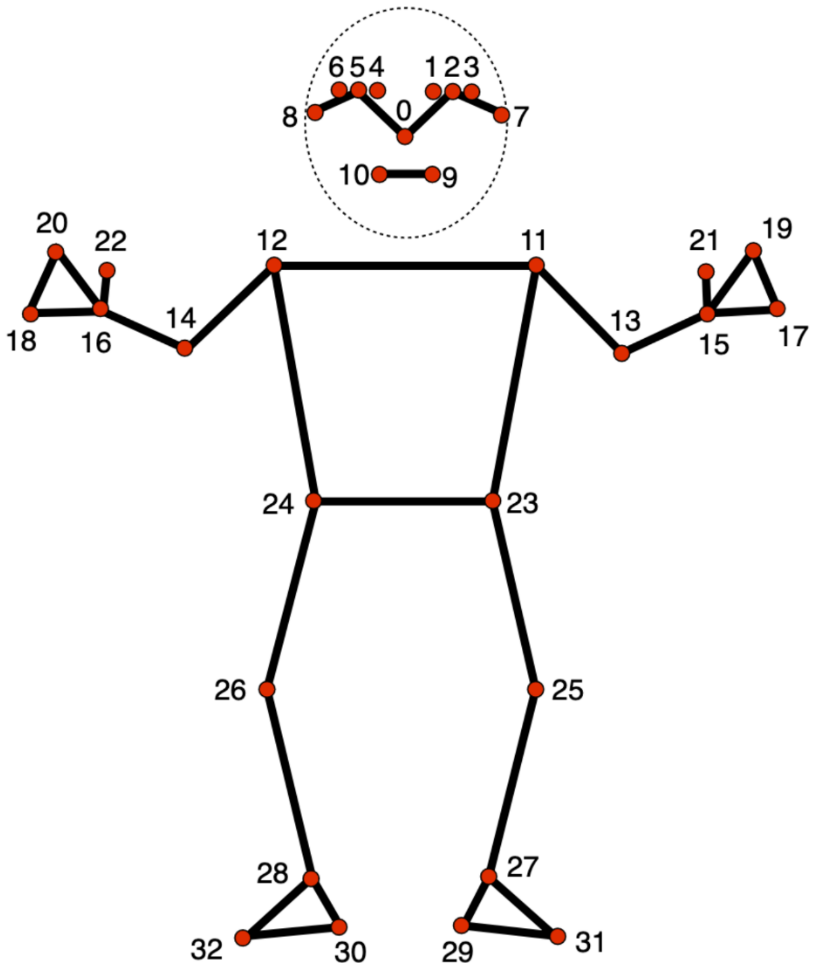
The last part is the security part. We explain how we use the pipeline to keep everything secure. This is extremely important due to privacy reasons. All parameters and variables are also clearly discussed.

1. AI

Our Fall detection system sends alerts when a fall is detected and sends daily movement information regarding the dependant.

Choosing a model

The first thing on the list was finding a suitable model that suited all our needs. After some thorough research we landed on the Mediapipe pose detection model, an open-source model from Google. It was very easy to implement and use and gave us a lot of possibilities to work with. The image below gives an idea of how the model sees a person.



Important to note here is that the model itself isn't capable of determining wether a person has fallen or not. The only thing it does is attaching keypoints to a persons body. In the next paragraph is explained how this can be used to obtain the functionalities we need.

Defining functionality

There are three key functionalities the model should have. The first one is being able to register a fall incident within camera view. A second functionality is to register a fall incident even if the person falls out of frame or behind some sort of furniture. The last functionality is to count the amount of time the person moves throughout the day.

How it works

The fall detection mechanism is based on the person's vertical nose position relative to a dynamically adjusted threshold height. The reason for this to be dynamic is that a person can fall right in front of the camera or 10 meters from the camera. Since the model looks at the video in 2D, we had to find a way to adjust the threshold based on the persons distance to the camera (think about the optical illusion of train rails meeting in the horizon). In case a person falls out of frame or behind furniture, we can still define this as a fall by defining this as the nose keypoint having a certain velocity and then suddenly disappearing.

As soon as the camera starts recording, a buffer for the video frames is generated. This way, we can store a certain amount of 'history footage'. In case a fall occurs, we simply add this history footage to the footage recorded after the fall, so the video contains footage before and after the incident. The fall is asynchronously sent to an Azure Blob Storage container while also notifying our backend server about the fall.

Other than the fall detection, the person's movement is tracked using the horizontal velocity of the center of the body. The amount of frames this value is higher than a certain threshold are counted and converted to minutes.

At the end of each day at 23:59pm, a JSON object filled with the movement data is sent to our backend system which will display the information on each person's dashboard.

2. App

Our monitoring system sends alerts to caregivers when there are changes in a patient's condition. These alerts can be manually triggered by patients or automatically when someone falls. These alerts can be escalated by a confidant based on how serious the situation is, allowing caregivers to respond accordingly.

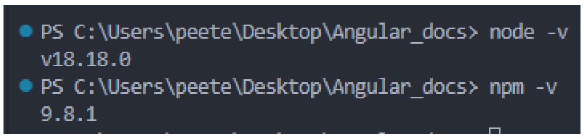
The easy-to-use dashboard shows daily movement data for each patient, giving caregivers a clear picture of their well-being. This helps caregivers make informed decisions and provide personalized care.

Front-End

For the Front-end, there are a few configuration files and pieces of code that we need to describe to give a clear overview of the changes that need to be made for a user to get this project up and running. Let's start by installing the right things to get a beginning app.

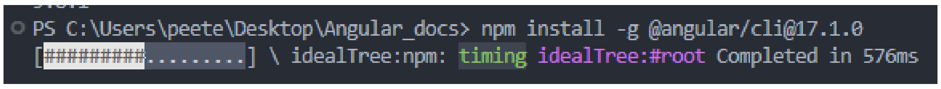
Create Application

The first thing to do is check your node version using the command below. If Node.js is not installed, you can do this on the node.js website itself. You can also check the npm version, but this is automatically installed with node.js.

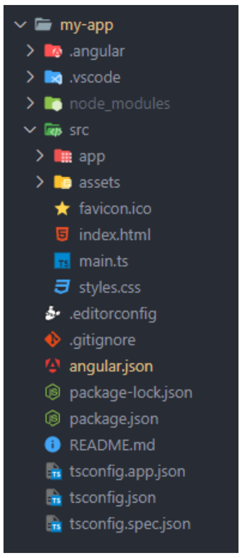


```
● PS C:\Users\peete\Desktop\Angular_docs> node -v
  v18.18.0
● PS C:\Users\peete\Desktop\Angular_docs> npm -v
  9.8.1
```

We are using the Angular CLI. The @ sign allows us to install a specific version of a npm package.

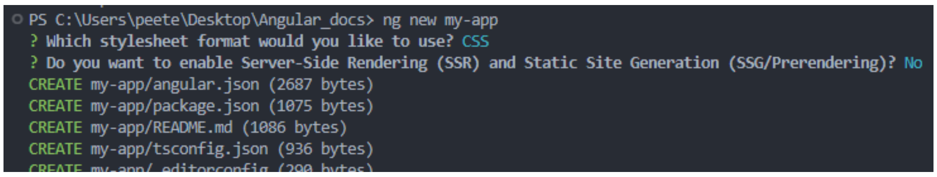


```
○ PS C:\Users\peete\Desktop\Angular_docs> npm install -g @angular/cli@17.1.0
  [##########.........] \ idealTree:npm: timing idealTree:#root Completed in 576ms
```



```
⌄ my-app
  › .angular
  › .vscode
  › node_modules
  ⌄ src
    › app
    › assets
      favicon.ico
      index.html
      main.ts
      styles.css
    .editorconfig
    .gitignore
    angular.json
    package-lock.json
    package.json
    README.md
    tsconfig.app.json
    tsconfig.json
    tsconfig.spec.json
```

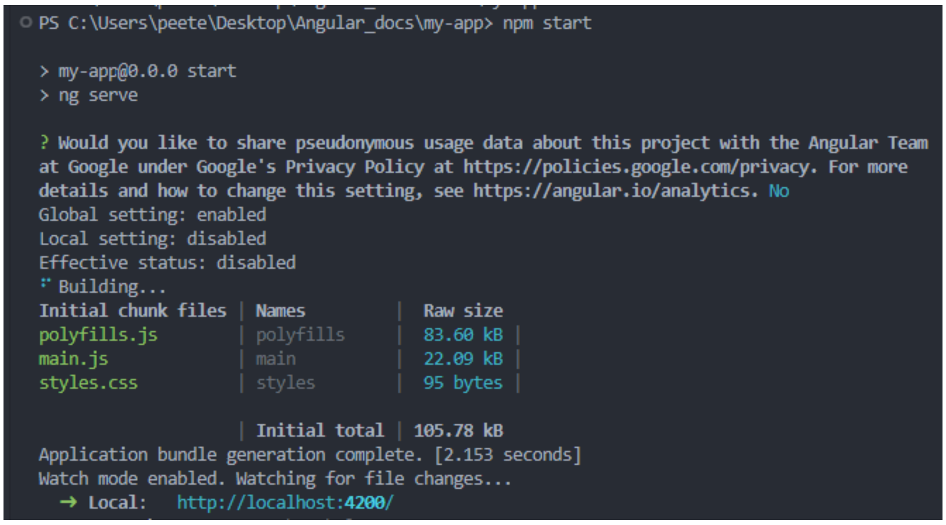
Of course, the next thing we need to do is create the application itself we do this by entering the command "ng new your-app-name". Then you must choose a style type, which in our case is CSS. Then another question is asked and to this you may answer 'no'. If that succeeds, the application is created and then it looks like the folder to the right of this.



```
○ PS C:\Users\peete\Desktop\Angular_docs> ng new my-app
  ? Which stylesheet format would you like to use? CSS
  ? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
  CREATE my-app/angular.json (2687 bytes)
  CREATE my-app/package.json (1075 bytes)
  CREATE my-app/README.md (1086 bytes)
  CREATE my-app/tsconfig.json (936 bytes)
  CREATE my-app/.editorconfig (290 bytes)
```

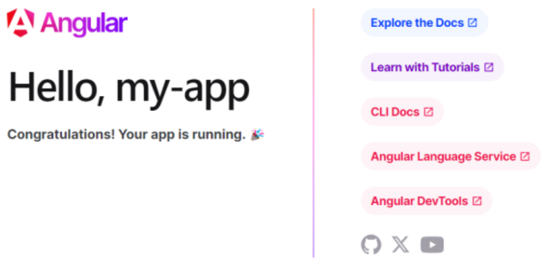
Elder Guardians

To start the application, go to your app's folder and enter 'npm start'. Then you can go to localhost:4200 to see if your application works.



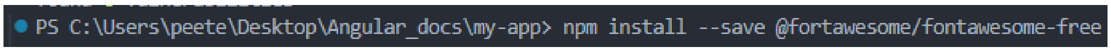
```
PS C:\Users\peete\Desktop\Angular_docs\my-app> npm start

> my-app@0.0.0 start
> ng serve

? Would you like to share pseudonymous usage data about this project with the Angular Team
at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more
details and how to change this setting, see https://angular.io/analytics. No
Global setting: enabled
Local setting: disabled
Effective status: disabled
∵ Building...
Initial chunk files | Names         |  Raw size
polyfills.js        | polyfills     |  83.60 kB |
main.js             | main          |  22.09 kB |
styles.css          | styles        |  95 bytes |

                    | Initial total | 105.78 kB
Application bundle generation complete. [2.153 seconds]
Watch mode enabled. Watching for file changes...
  ➜  Local:   http://localhost:4200/
```

If the application works, you will normally see the following.

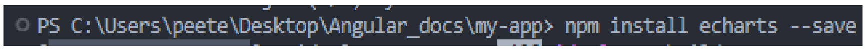


Other installations

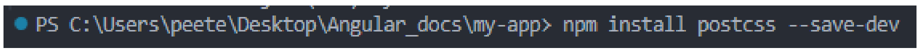
There are some installations that need to happen such as:
- font awesome (for icons)
- echarts (for the dashboard)
- postcss and tailwindcss (for styling)
- tailwind (for the custom colours)



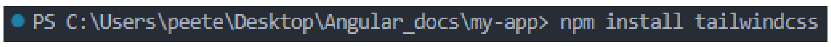
```
PS C:\Users\peete\Desktop\Angular_docs\my-app> npm install --save @fortawesome/fontawesome-free
```



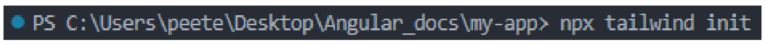
```
PS C:\Users\peete\Desktop\Angular_docs\my-app> npm install echarts --save
```



```
PS C:\Users\peete\Desktop\Angular_docs\my-app> npm install postcss --save-dev
```



```
PS C:\Users\peete\Desktop\Angular_docs\my-app> npm install tailwindcss
```

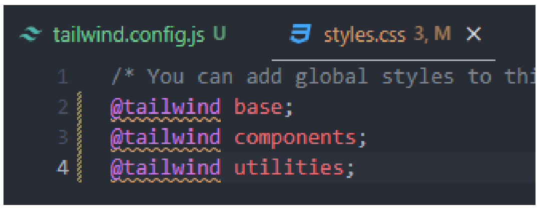


```
PS C:\Users\peete\Desktop\Angular_docs\my-app> npx tailwind init
```

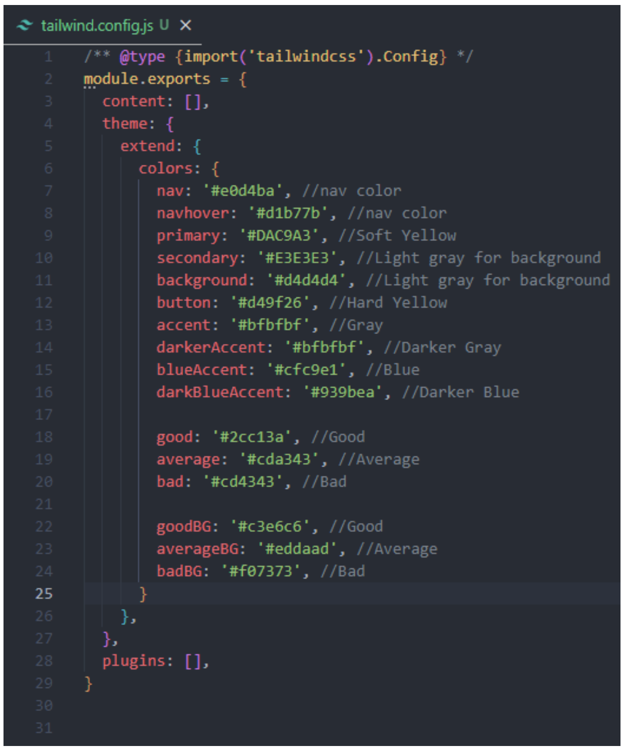
Elder Guardians

The tailwind configuration consists of the default colours and the custom colours we add by using 'extend colours'.

The @tailwind directives will be processed by Tailwind, which will then inject its base, components, and utilities classes.



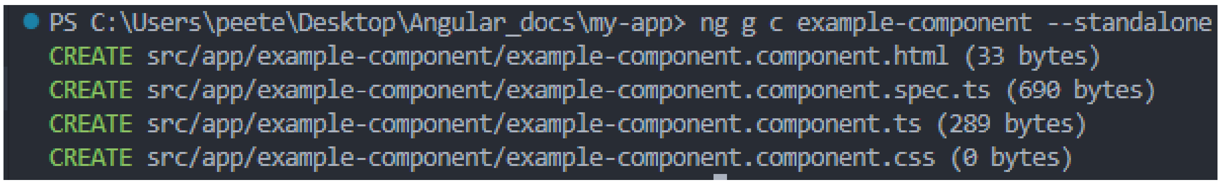
```css
/* You can add global styles to thi
@tailwind base;
@tailwind components;
@tailwind utilities;
```



```js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [],
  theme: {
    extend: {
      colors: {
        nav: '#e0d4ba', //nav color
        navhover: '#d1b77b', //nav color
        primary: '#DAC9A3', //Soft Yellow
        secondary: '#E3E3E3', //Light gray for background
        background: '#d4d4d4', //Light gray for background
        button: '#d49f26', //Hard Yellow
        accent: '#bfbfbf', //Gray
        darkerAccent: '#bfbfbf', //Darker Gray
        blueAccent: '#cfc9e1', //Blue
        darkBlueAccent: '#939bea', //Darker Blue

        good: '#2cc13a', //Good
        average: '#cda343', //Average
        bad: '#cd4343', //Bad

        goodBG: '#c3e6c6', //Good
        averageBG: '#eddaad', //Average
        badBG: '#f07373', //Bad
      }
    },
  },
  plugins: [],
}
```

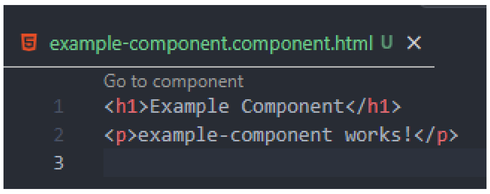
Creating components

Creating components is something very important because the application consists mainly of different components. The command below shows how to create a separate component.



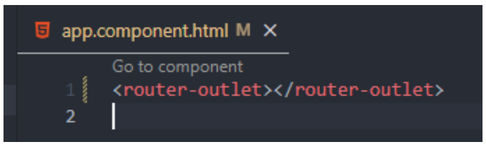
```
PS C:\Users\peete\Desktop\Angular_docs\my-app> ng g c example-component --standalone
CREATE src/app/example-component/example-component.component.html (33 bytes)
CREATE src/app/example-component/example-component.component.spec.ts (690 bytes)
CREATE src/app/example-component/example-component.component.ts (289 bytes)
CREATE src/app/example-component/example-component.component.css (0 bytes)
```

Of course, as an example, we do need to show something on the page. Hence, I put in an H1 and a p.



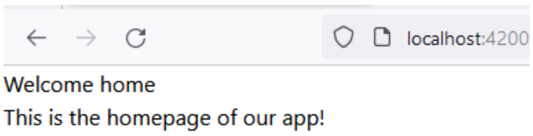
```html
<h1>Example Component</h1>
<p>example-component works!</p>
```

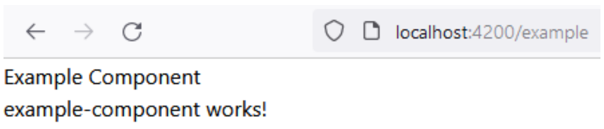
what you need to do is put 'router-outlet' in the app html component. Because then through the router the correct route will be designated in the url, then the correct html will also be deployed in this router-outlet.



```html
<router-outlet></router-outlet>
```

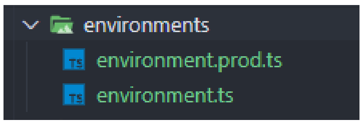
I have created 2 components to demonstrate this. The home with as route ' ', and the example with as route '/example'. You can therefore see this in the examples below.

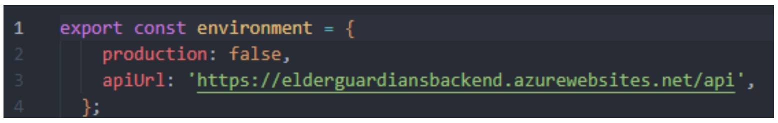

Welcome home
This is the homepage of our app!

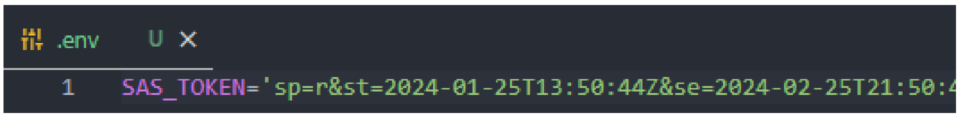

Example Component
example-component works!

Environments

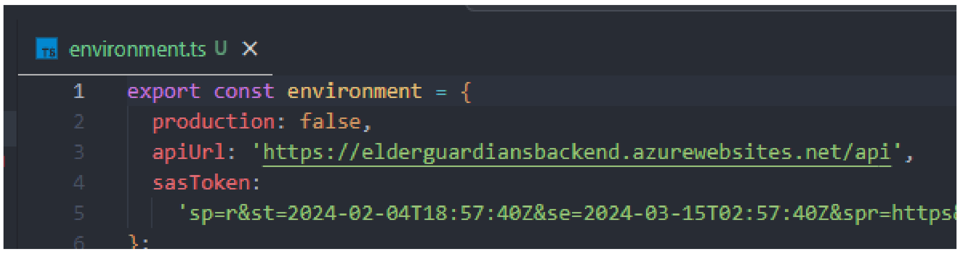
Environments is the last thing that is very important to know to make the project run well. You create a folder in the src folder called 'environments'. In it, you create a production environment and a local one. The sastoken is to extract the video from the blobstorage that ai passes. There is also a special file for this called '.env'. If the sastoken is wrong, you can't just get the video.


```
environments
  environment.prod.ts
  environment.ts
```


```
export const environment = {
    production: false,
    apiUrl: 'https://elderguardiansbackend.azurewebsites.net/api',
};
```



```
SAS_TOKEN='sp=r&st=2024-01-25T13:50:44Z&se=2024-02-25T21:50:4
```



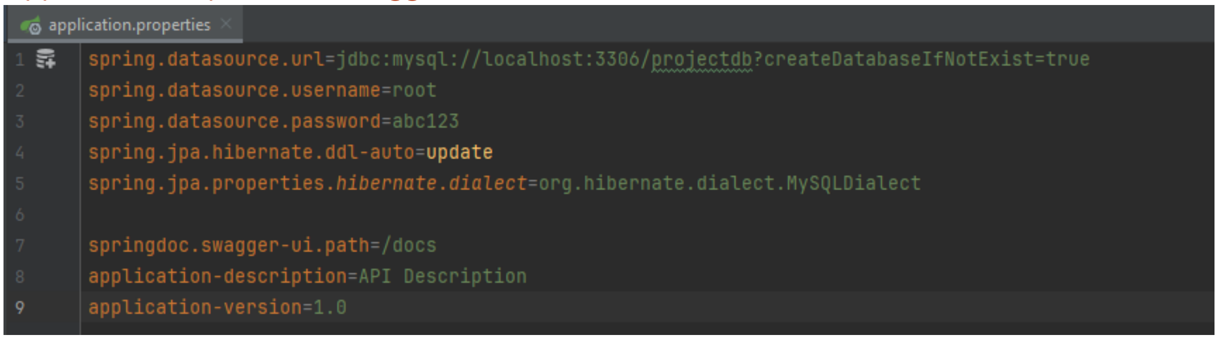
```
export const environment = {
  production: false,
  apiUrl: 'https://elderguardiansbackend.azurewebsites.net/api',
  sasToken:
    'sp=r&st=2024-02-04T18:57:40Z&se=2024-03-15T02:57:40Z&spr=https
};
```

Of course, you don't have to go through all these steps if you just clone our repo. What you do need to do are, of course, the npm installations listed under 'other installations' in this document.

Back-End

For the back-end there are a few configuration files and pieces of code we need to describe to provide a clear overview of changes that need to be made for a user to get this project running. Let's start with the application.properties configuration file we can find under "/API/src/main/resources".

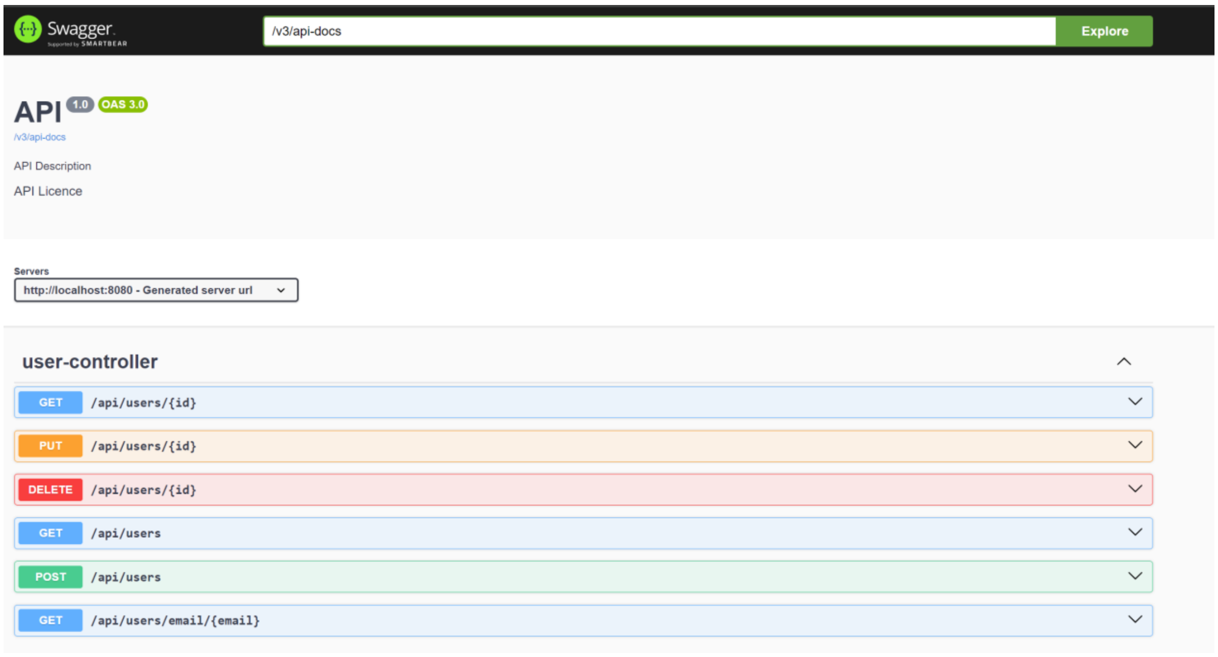
Application Properties & Swagger



```
application.properties
spring.datasource.url=jdbc:mysql://localhost:3306/projectdb?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=abc123
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

springdoc.swagger-ui.path=/docs
application-description=API Description
application-version=1.0
```

In the application.properties file we can find the database configuration properties. Depending on your setup and requirements, you might want to change the name under "spring.datasource.username" and the password under "spring.datasource.password". In the spring datasource url you can also change the port (3306) and the name of the database (projectdb).

We us swagger for our API Documentation. If you want to customize the endpoint for reaching swagger, you can modify the "/docs" under "springdoc.swagger-ui.path".



API 1.0 OAS 3.0

/v3/api-docs

API Description

API Licence

Servers

http://localhost:8080 - Generated server url

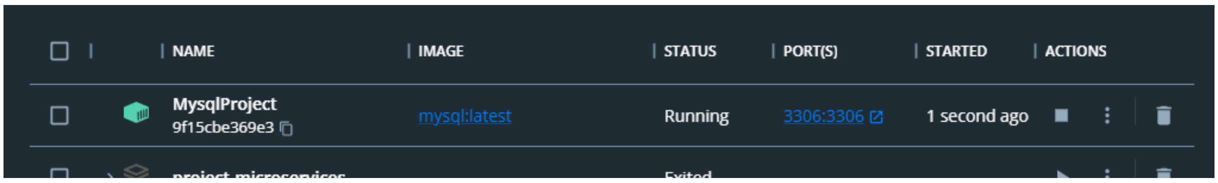
user-controller

```
GET     /api/users/{id}
PUT     /api/users/{id}
DELETE  /api/users/{id}
GET     /api/users
POST    /api/users
GET     /api/users/email/{email}
```

Docker (Required software: Docker Desktop)

```
docker run --name MysqlProject -p
3306:3306 -e
MYSQL_ROOT_PASSWORD=abc123 -d mysql
```

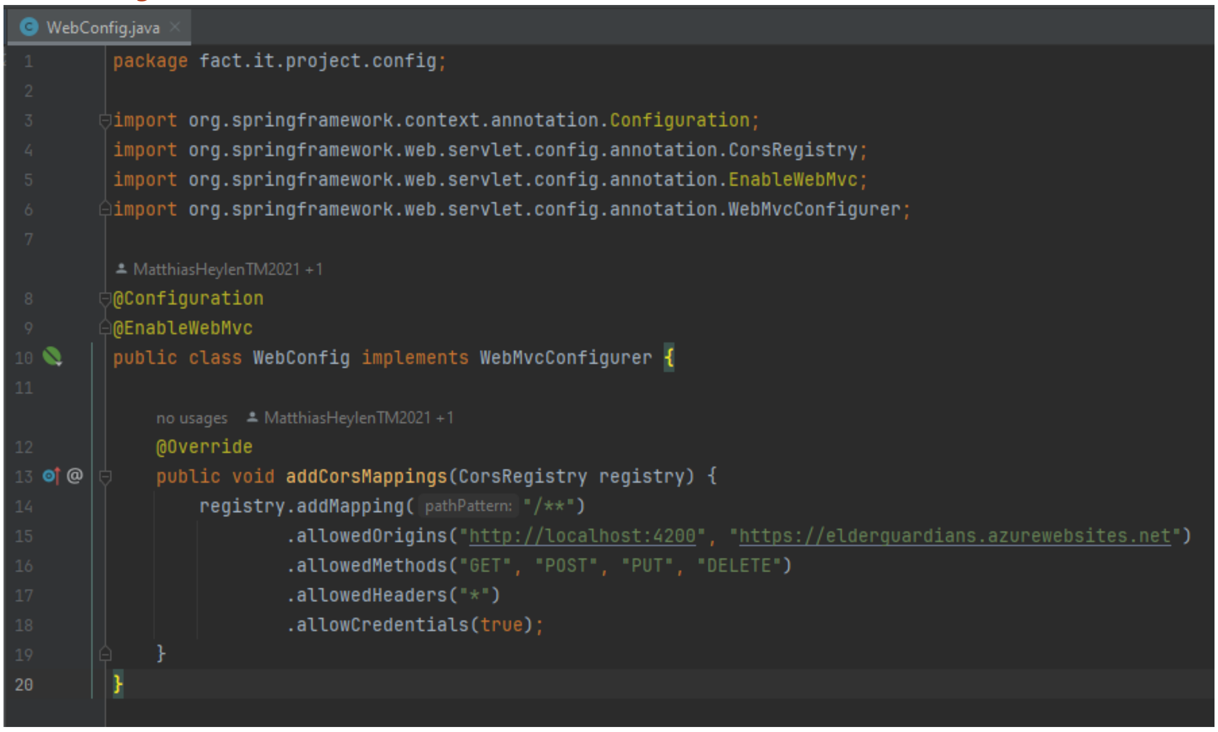
For initializing the database we can use the “docker run –name projectname -p port:port -e MYSQL_ROOT_PASSWORD=password -d mysql” command.



	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
	MysqlProject 9f15cbe369e3	mysql:latest	Running	3306:3306	1 second ago	
	project-microservices		Exited			

Once this command has been executed, we can see that our container is activated.

Cors configuration



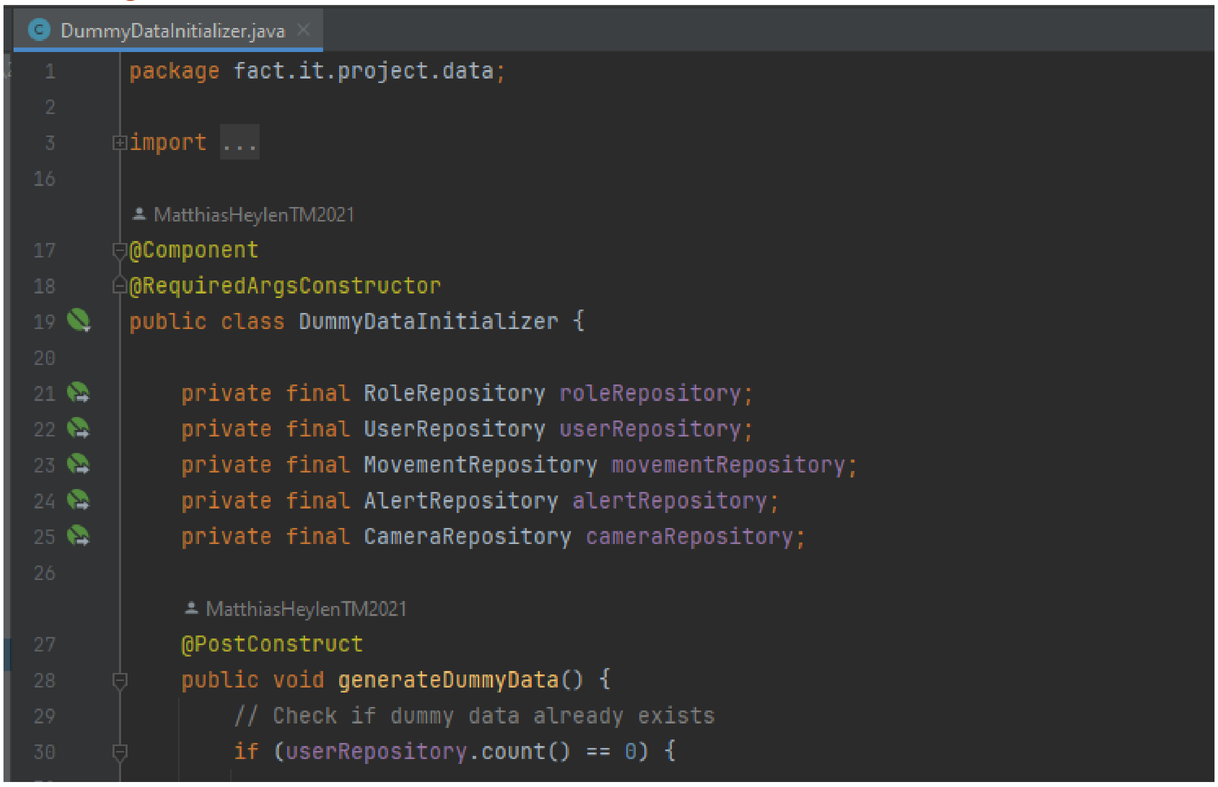
WebConfig.java

```java
package fact.it.project.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping( pathPattern: "/**")
                .allowedOrigins("http://localhost:4200", "https://elderguardians.azurewebsites.net")
                .allowedMethods("GET", "POST", "PUT", "DELETE")
                .allowedHeaders("*")
                .allowCredentials(true);
    }
}
```

When deploying the website, CORS (Cross-Origin Resource Sharing) errors might occur. To prevent these errors from occurring, make sure the correct information is filled in the WebConfig configuration class. Under the “addCorsMappings” method a user can modify the url’s that fit his requirements.

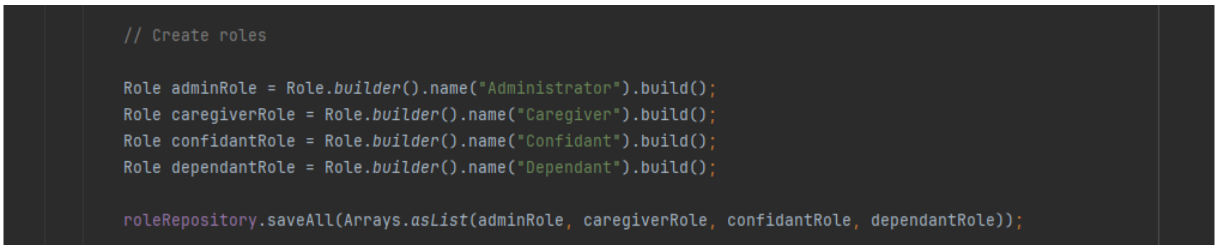
Elder Guardians

Initializing users and data



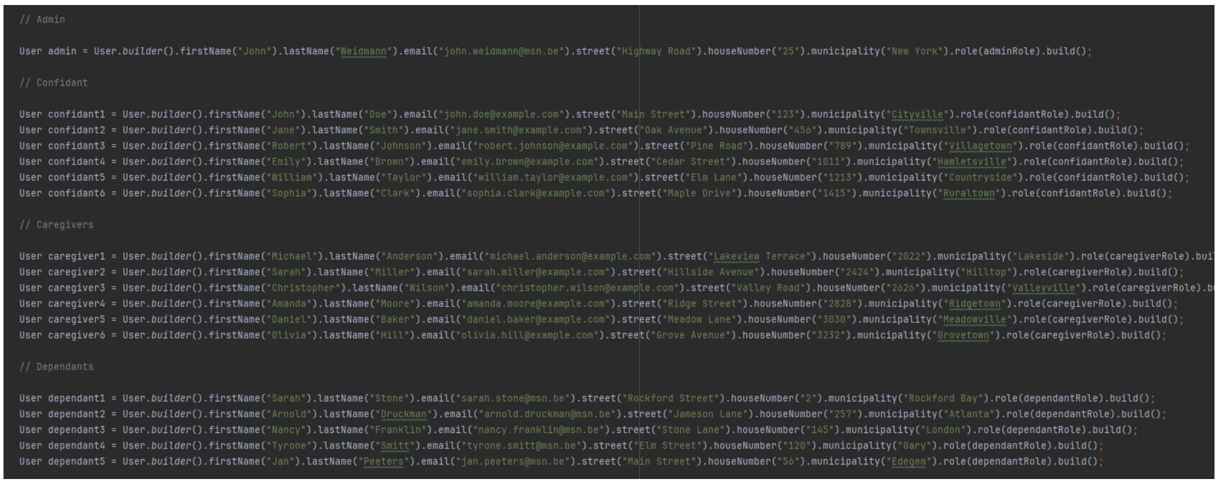
```
package fact.it.project.data;

import ...

MatthiasHeylenTM2021
@Component
@RequiredArgsConstructor
public class DummyDataInitializer {

    private final RoleRepository roleRepository;
    private final UserRepository userRepository;
    private final MovementRepository movementRepository;
    private final AlertRepository alertRepository;
    private final CameraRepository cameraRepository;

    MatthiasHeylenTM2021
    @PostConstruct
    public void generateDummyData() {
        // Check if dummy data already exists
        if (userRepository.count() == 0) {
```

To initialize users in the users, we use a DummyDataInitializer class. In this class we have already provided data that can be removed or modified.



```
// Create roles

Role adminRole = Role.builder().name("Administrator").build();
Role caregiverRole = Role.builder().name("Caregiver").build();
Role confidantRole = Role.builder().name("Confidant").build();
Role dependantRole = Role.builder().name("Dependant").build();

roleRepository.saveAll(Arrays.asList(adminRole, caregiverRole, confidantRole, dependantRole));
```

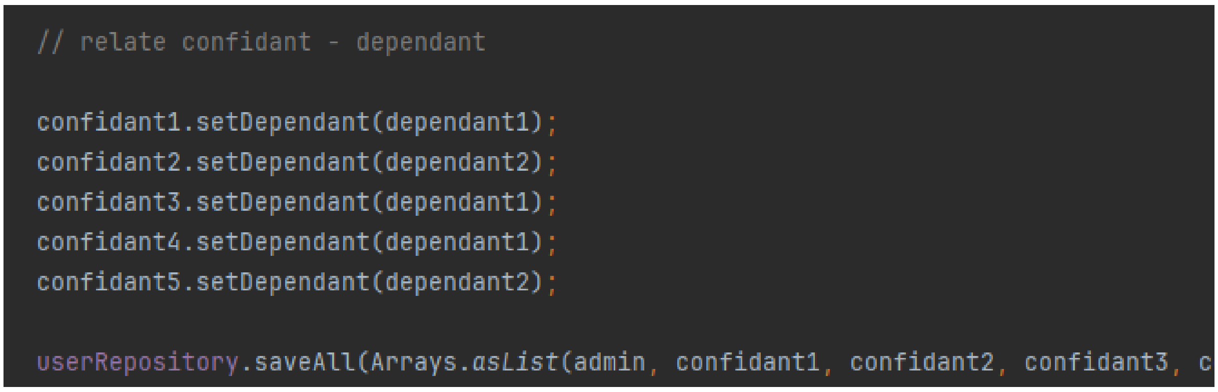
These are the main roles of our application. If you want to remove, modify or add a role to your application, you can do it by modifying this section. Don't forget to modify the saveAll method after removing or adding a Role.



```
// Admin

User admin = User.builder().firstName("John").lastName("Weidmann").email("john.weidmann@msn.be").street("Highway Road").houseNumber("25").municipality("New York").role(adminRole).build();

// Confidant

User confidant1 = User.builder().firstName("John").lastName("Doe").email("john.doe@example.com").street("Main Street").houseNumber("123").municipality("Cityville").role(confidantRole).build();
User confidant2 = User.builder().firstName("Jane").lastName("Smith").email("jane.smith@example.com").street("Oak Avenue").houseNumber("456").municipality("Townsville").role(confidantRole).build();
User confidant3 = User.builder().firstName("Robert").lastName("Johnson").email("robert.johnson@example.com").street("Pine Road").houseNumber("789").municipality("Villagetown").role(confidantRole).build();
User confidant4 = User.builder().firstName("Emily").lastName("Brown").email("emily.brown@example.com").street("Cedar Street").houseNumber("1011").municipality("Hamletsville").role(confidantRole).build();
User confidant5 = User.builder().firstName("William").lastName("Taylor").email("william.taylor@example.com").street("Elm Lane").houseNumber("1213").municipality("Countryside").role(confidantRole).build();
User confidant6 = User.builder().firstName("Sophia").lastName("Clark").email("sophia.clark@example.com").street("Maple Drive").houseNumber("1415").municipality("Ruraltown").role(confidantRole).build();

// Caregivers

User caregiver1 = User.builder().firstName("Michael").lastName("Anderson").email("michael.anderson@example.com").street("Lakeview Terrace").houseNumber("2022").municipality("Lakeside").role(caregiverRole).bui
User caregiver2 = User.builder().firstName("Sarah").lastName("Miller").email("sarah.miller@example.com").street("Hillside Avenue").houseNumber("2424").municipality("Hilltop").role(caregiverRole).build();
User caregiver3 = User.builder().firstName("Christopher").lastName("Wilson").email("christopher.wilson@example.com").street("Valley Road").houseNumber("2626").municipality("Valleyville").role(caregiverRole).b
User caregiver4 = User.builder().firstName("Amanda").lastName("Moore").email("amanda.moore@example.com").street("Ridge Street").houseNumber("2828").municipality("Ridgetown").role(caregiverRole).build();
User caregiver5 = User.builder().firstName("Daniel").lastName("Baker").email("daniel.baker@example.com").street("Meadow Lane").houseNumber("3030").municipality("Meadowville").role(caregiverRole).build();
User caregiver6 = User.builder().firstName("Olivia").lastName("Hill").email("olivia.hill@example.com").street("Grove Avenue").houseNumber("3232").municipality("Grovetown").role(caregiverRole).build();

// Dependants

User dependant1 = User.builder().firstName("Sarah").lastName("Stone").email("sarah.stone@msn.be").street("Rockford Street").houseNumber("2").municipality("Rockford Bay").role(dependantRole).build();
User dependant2 = User.builder().firstName("Arnold").lastName("Druckman").email("arnold.druckman@msn.be").street("Jameson Lane").houseNumber("257").municipality("Atlanta").role(dependantRole).build();
User dependant3 = User.builder().firstName("Nancy").lastName("Franklin").email("nancy.franklin@msn.be").street("Stone Lane").houseNumber("145").municipality("London").role(dependantRole).build();
User dependant4 = User.builder().firstName("Tyrone").lastName("Smitt").email("tyrone.smitt@msn.be").street("Elm Street").houseNumber("120").municipality("Gary").role(dependantRole).build();
User dependant5 = User.builder().firstName("Jan").lastName("Peeters").email("jan.peeters@msn.be").street("Main Street").houseNumber("56").municipality("Edegem").role(dependantRole).build();
```

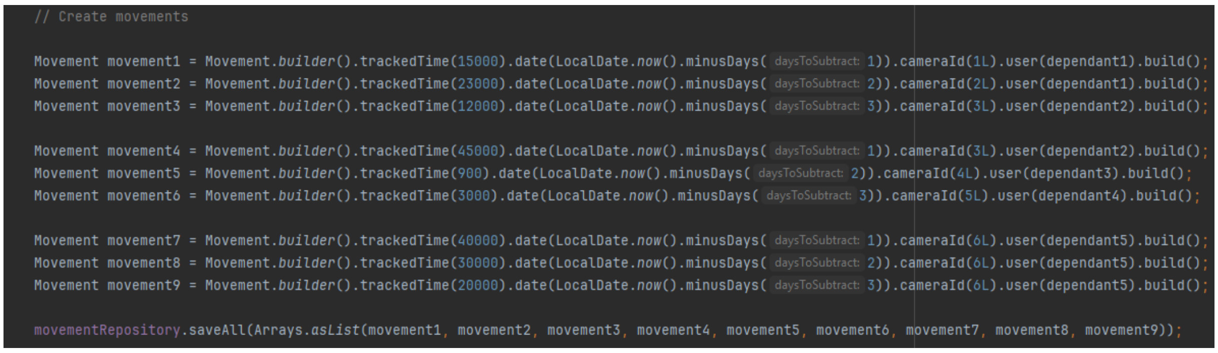
Elder Guardians

These are the current users in our application. For a use you can initialize a firstname, lastname, email, street, housenumber, municipality and a role.



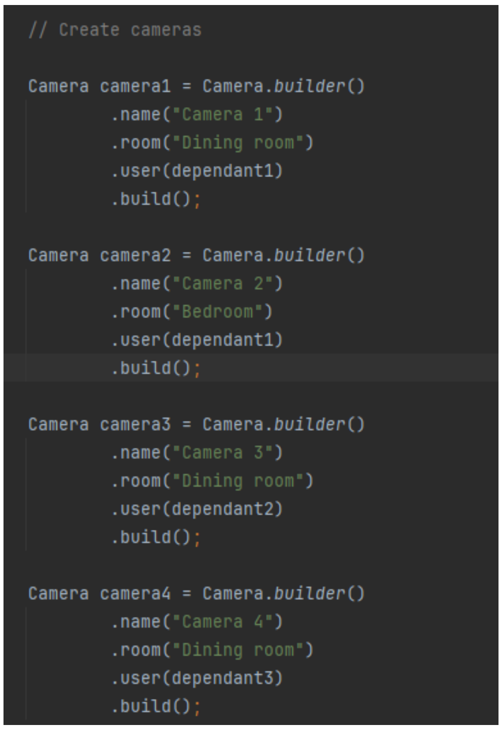
```
// relate confidant - dependant

confidant1.setDependant(dependant1);
confidant2.setDependant(dependant2);
confidant3.setDependant(dependant1);
confidant4.setDependant(dependant1);
confidant5.setDependant(dependant2);

userRepository.saveAll(Arrays.asList(admin, confidant1, confidant2, confidant3, c
```

After creating users, you might also want to relate the dependants and their confidants.



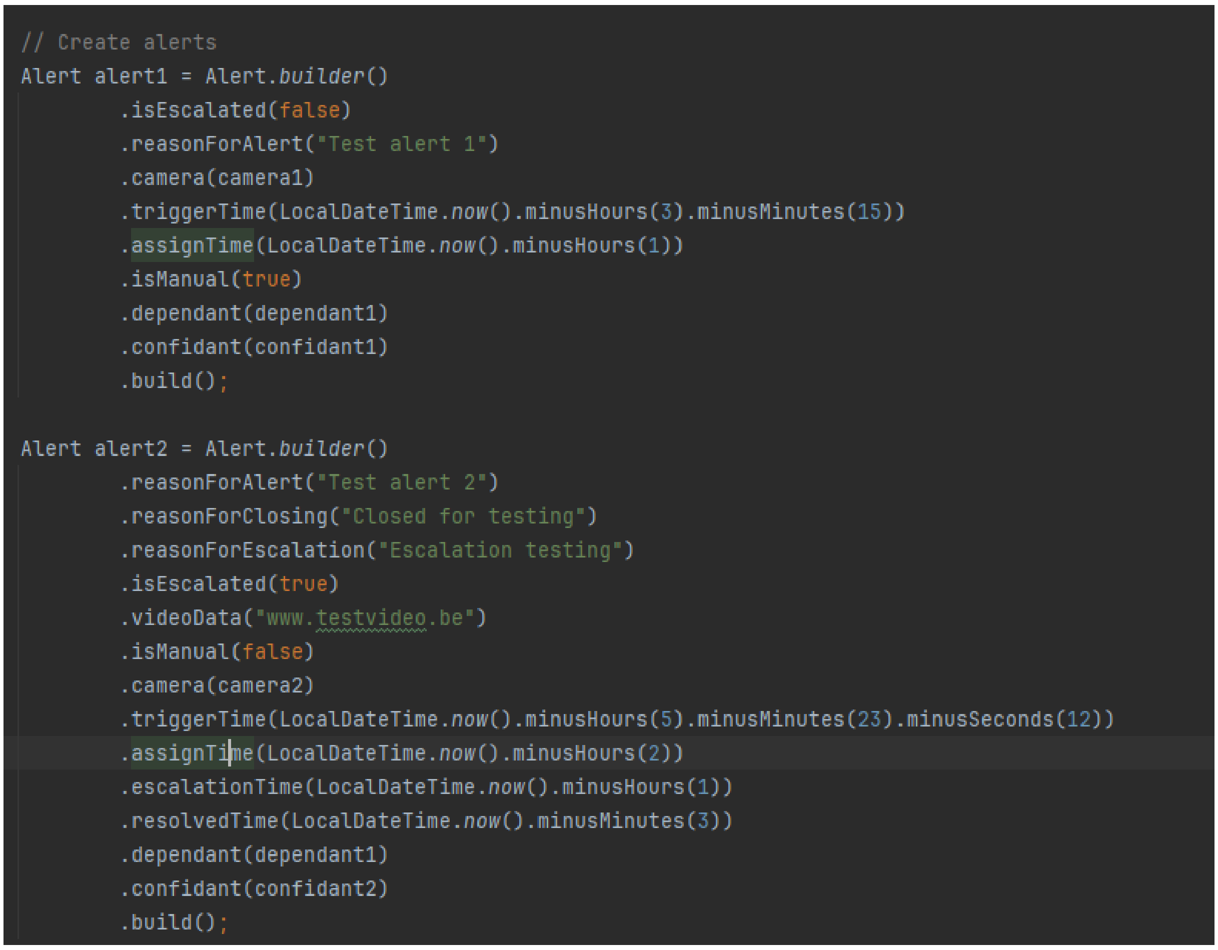
```
// Create movements

Movement movement1 = Movement.builder().trackedTime(15000).date(LocalDate.now().minusDays( daysToSubtract: 1)).cameraId(1L).user(dependant1).build();
Movement movement2 = Movement.builder().trackedTime(23000).date(LocalDate.now().minusDays( daysToSubtract: 2)).cameraId(2L).user(dependant1).build();
Movement movement3 = Movement.builder().trackedTime(12000).date(LocalDate.now().minusDays( daysToSubtract: 3)).cameraId(3L).user(dependant2).build();

Movement movement4 = Movement.builder().trackedTime(45000).date(LocalDate.now().minusDays( daysToSubtract: 1)).cameraId(3L).user(dependant2).build();
Movement movement5 = Movement.builder().trackedTime(900).date(LocalDate.now().minusDays( daysToSubtract: 2)).cameraId(4L).user(dependant3).build();
Movement movement6 = Movement.builder().trackedTime(3000).date(LocalDate.now().minusDays( daysToSubtract: 3)).cameraId(5L).user(dependant4).build();

Movement movement7 = Movement.builder().trackedTime(40000).date(LocalDate.now().minusDays( daysToSubtract: 1)).cameraId(6L).user(dependant5).build();
Movement movement8 = Movement.builder().trackedTime(30000).date(LocalDate.now().minusDays( daysToSubtract: 2)).cameraId(6L).user(dependant5).build();
Movement movement9 = Movement.builder().trackedTime(20000).date(LocalDate.now().minusDays( daysToSubtract: 3)).cameraId(6L).user(dependant5).build();

movementRepository.saveAll(Arrays.asList(movement1, movement2, movement3, movement4, movement5, movement6, movement7, movement8, movement9));
```

You can also create movement data for existing users. Movement data requires the "trackedTime", a date and an attached user.



```
// Create cameras

Camera camera1 = Camera.builder()
        .name("Camera 1")
        .room("Dining room")
        .user(dependant1)
        .build();

Camera camera2 = Camera.builder()
        .name("Camera 2")
        .room("Bedroom")
        .user(dependant1)
        .build();

Camera camera3 = Camera.builder()
        .name("Camera 3")
        .room("Dining room")
        .user(dependant2)
        .build();

Camera camera4 = Camera.builder()
        .name("Camera 4")
        .room("Dining room")
        .user(dependant3)
        .build();
```

You can add camera's here for your users. Cameras require a name, location/room and a user.



```java
// Create alerts
Alert alert1 = Alert.builder()
        .isEscalated(false)
        .reasonForAlert("Test alert 1")
        .camera(camera1)
        .triggerTime(LocalDateTime.now().minusHours(3).minusMinutes(15))
        .assignTime(LocalDateTime.now().minusHours(1))
        .isManual(true)
        .dependant(dependant1)
        .confidant(confidant1)
        .build();

Alert alert2 = Alert.builder()
        .reasonForAlert("Test alert 2")
        .reasonForClosing("Closed for testing")
        .reasonForEscalation("Escalation testing")
        .isEscalated(true)
        .videoData("www.testvideo.be")
        .isManual(false)
        .camera(camera2)
        .triggerTime(LocalDateTime.now().minusHours(5).minusMinutes(23).minusSeconds(12))
        .assignTime(LocalDateTime.now().minusHours(2))
        .escalationTime(LocalDateTime.now().minusHours(1))
        .resolvedTime(LocalDateTime.now().minusMinutes(3))
        .dependant(dependant1)
        .confidant(confidant2)
        .build();
```

Ultimately you can also initialize alerts. As these are quite expansive, not all information is required. For example, a manual alert does not have a camera, while an automatic alert will have video data attached to it (the URL of the video).

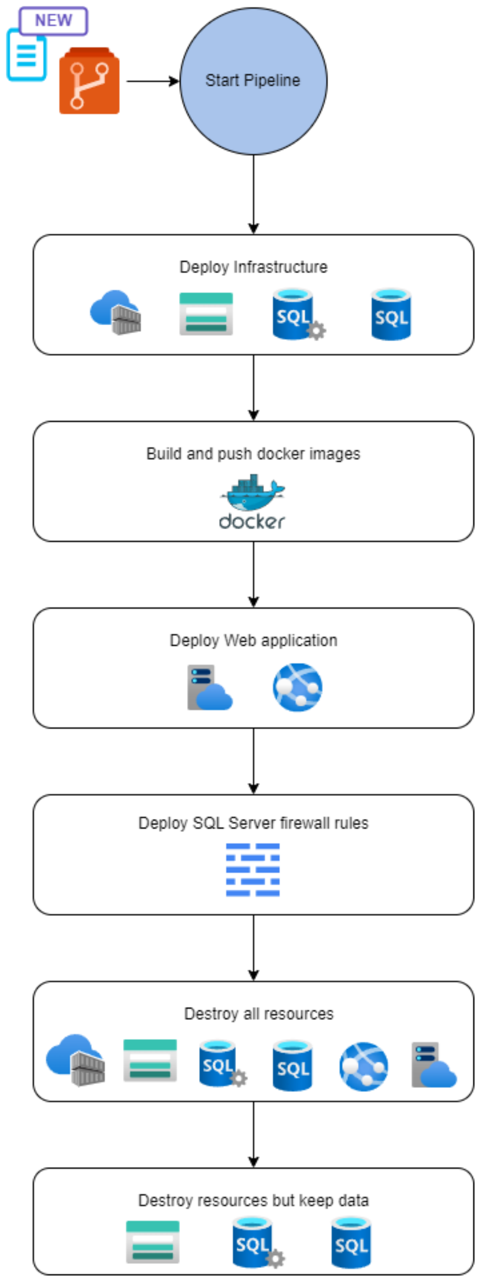
Elder Guardians

3. CCS

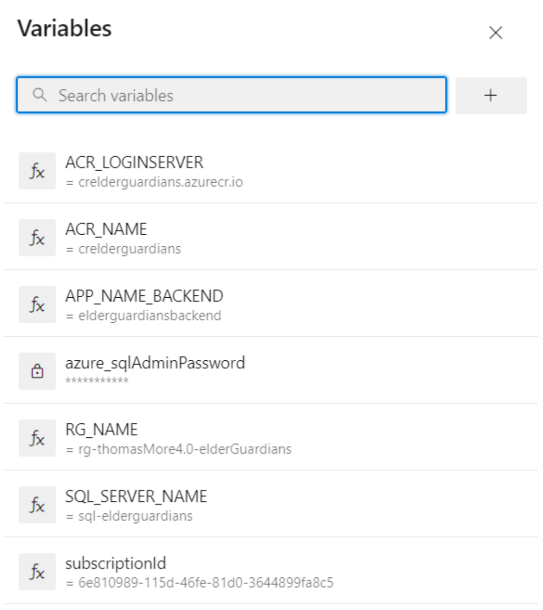
The Cloud and Cyber Security students provided a working CI/CD pipeline utilizing Azure DevOps (Azure Repo and Azure Pipeline) to deploy the needed infrastructure for this project. The pipeline can deploy a fully functional application, utilizing the latest available code without manual actions needed.

Pipeline scheme

The pipeline is designed to deploy the necessary infrastructure to host the front-end and back-end as well as being able to view the video in the web application. It will build and push docker images of our applications to the container registry. The applications will be hosted on app services, this will be done via deploy web application stage. The pipeline is able to destroy all the resources that are inside of the resource group, but there is a destroy that keeps the data: Storage account, SQL server and database.



Pipeline variables


Variables

Search variables

ACR_LOGINSERVER	= crelderguardians.azurecr.io
ACR_NAME	= crelderguardians
APP_NAME_BACKEND	= elderguardiansbackend
azure_sqlAdminPassword	***********
RG_NAME	= rg-thomasMore4.0-elderGuardians
SQL_SERVER_NAME	= sql-elderguardians
subscriptionId	= 6e810989-115d-46fe-81d0-3644899fa8c5

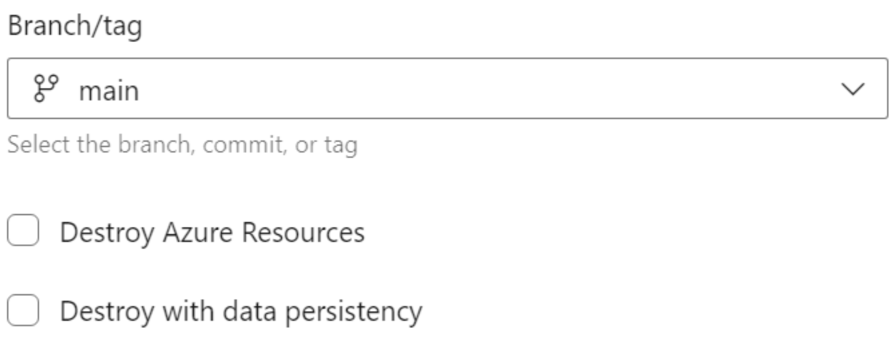
Those variables ensure that the pipeline code is more readable and changes to another environment are easier to perform.

To change environments, only the Azure Resource Group and Subscription ID should be changed.

The Azure container registry URL and name where the frontend and backend containers are stored and the database username and password are stored as variables as well for easier management and usage of these values.

Pipeline Parameters

We included two parameters in the pipeline which are Booleans. With this you are able to destroy all resources or destroy the resources without any data.


Branch/tag

main

Select the branch, commit, or tag

☐ Destroy Azure Resources

☐ Destroy with data persistency

Pipeline Stages and jobs

Deploy Infrastructure

The pipeline will first deploy the “independent” infrastructure. This is the infrastructure that doesn’t need anything else to be built beforehand. In our architecture this is the container registry, the storage account and container and also the SQL server and database.

Build and push docker images

We containerized our applications, these need to be built and pushed to the container registry. So that in the next step our app services can get these docker images and host them. There is a condition, this stage will only run if the parameters are not checked on runtime besides this it also consists of two jobs: build and push front-end and build and push back-end. In each of these jobs a script is called. The script needs some arguments to be able to build and push the docker images.

Deploy web applications

We have our images in the registry and now we can start with deploying the app services to host the front-end and back-end applications. In the job you can see that we override a parameter, this is to keep the password protected and not have it in our own code.

Deploy SQL firewall rules

Only the back-end or API has to be able to access the database. We did this by adding a script to the stage/job. That will get all the outbound IP addresses of the app service which is hosting the API and will make SQL server firewall rules of these outbound IP addresses.

Destroy all resources

We have created a parameter that could be used when the pipeline is manually triggered. If the box is checked it will only run this stage. The job will call a script that will be run to destroy the resources. The resource group name variable in the pipeline is called as an argument so that the script knows on which resource group the destroy needs to happen.

Destroy specific resources

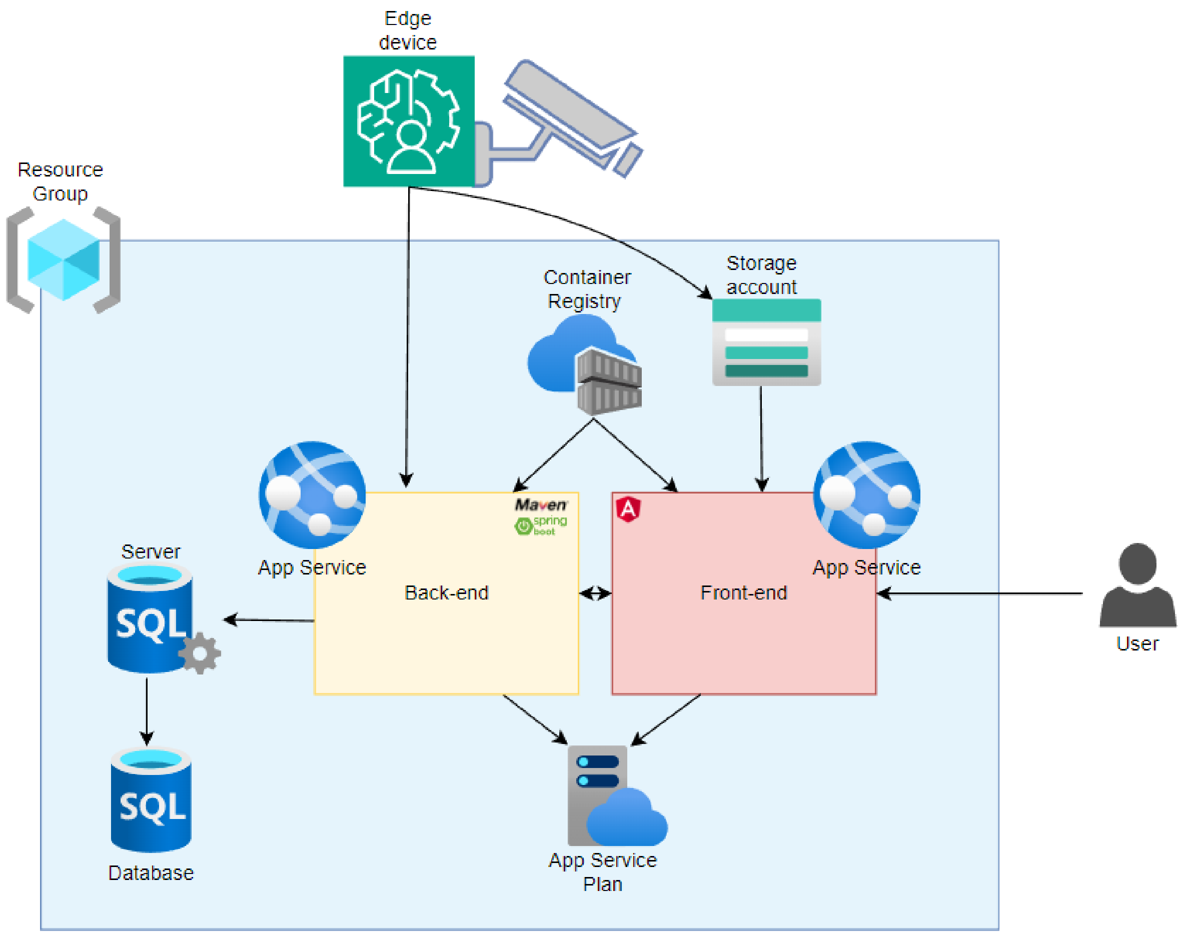
We have created a parameter that could be used when the pipeline is manually triggered. If the box is checked it will only run this stage. The job will call a script that will be ran to destroy the resources but keep the data of the application. The resource group name variable in the pipeline is called as an argument so that the script knows on which resource group the destroy needs to happen.

Pipeline scripts

In our pipeline configuration, we've adopted a practice of organizing our scripts into separate files. This approach not only ensures clean code but also offers us several key advantages in terms of modularity, reusability, and scalability.

Cloud architecture

The whole Azure cloud setup has been drawn out to visualize the used components.



We built our cloud architecture using infrastructure as code, bicep in particular. In our cloud architecture we have a container registry which will hold the images of the applications, front-end and back-end. Two app services which will host the applications. We also have aa SQL server; on this server we have one database which will hold the data of the application. We also added 2 webhooks on the container registry which will allow us to have continuous deployment.

The edge device will send the video – which will be shown in the application - to the storage account and they will also send other data to the backend with a post request.

4. Conclusion

The AI Component uses computer vision and pose estimation to monitor movements, detect falls, and issue alerts. Leveraging MediaPipe and OpenCV, keypoint positions and velocity are calculated for precise movement tracking. Fall detection tiggers alerts based on nose position and velocity, with incident data recorded and asynchronously sent to Azure Blob Storage.

The application developed using Angular for the front-end and Java/Spring Boot for the back-end, serves as a caregiver interface. It delivers real-time alerts and a dashboard displaying daily movement data, allowing informed decisions on personalized care.

Front-end development involves Angular CLI commands, dependency installations and component creation. Configuration files and dependencies like font awesome, echarts, postcss, and tailwindcss are crucial for styling.

In the back-end, Java/Spring Boot configures databases, utilizing Swagger for API documentation. Docker initializes the database, and CORS configuration addresses Cross-Origin Resource Sharing errors. User and data initialization, role management, camera setup, and alert configuration are integral components.

Cloud and Cyber Security are managed through an Azure DevOps CI/CD pipeline, deploying infrastructure efficiently. The pipeline, designed for hosting front-end and back-end components, incorporates Docker image building, hosting through app services, and selective resource destruction.

With variable-driven configurability and parameterized options for resource destruction, the pipeline ensures adaptability across different environments.

The cloud architecture, implemented using Bicep for Infrastructure as code, outlines the components, including a container registry, app services, SQL server, and database. Webhooks on the container registry enable continuous deployment, and the data flow involves the edge device sending video content to the storage account and other data to the back-end via a post request.