Documentation

## 

## Contents

[Contents](#_pgbz5el4uf7h)

[Overall](#_aubchhp3afnh)

[Front-end](#_6uc7dupoi3vj)

[ConnectHandler](#_mt03zqxyhp36)

[get](#_l76ugdcvdzx9)

[getInfo](#_xyz1ygsmotqo)

[postInfo](#_uk3p77u0yti1)

[putInfo](#_sxhl8g8tz6xf)

[Pages](#_7skc4arf4fbi)

[MainPage](#_10gynjp886l5)

[Reload products and catalogues button](#_rwtq7y8wg4c8)

[Create directory button](#_p1ld9p3j814k)

[Create product button](#_d98rdfsnrmy4)

[Other functionalities](#_zdgai3qitspt)

[HomePage](#_jfyqsvxy8l2r)

[AdminPage](#_knz5mftofxgj)

[RegisterPage](#_y6cwmcwfnuhk)

[ProductForm](#_6ln4losxr25i)

[CatalogueFrom](#_rxg1w2a1wih7)

[Back-end](#_og1x8iums9cu)

[UserController](#_zbwyuko10tpn)

[ProductController](#_x5v4klmsrdun)

[CatalogueController](#_9063zvmmq9eo)

[Database](#_stbkq2ke8s4a)

[DTOs](#_fb9dxayss8pc)

[UserDto](#_hxm8qt820xew)

[CatalogueDto](#_bm4sqs4d11g9)

[ProductDto](#_5oyzaafb20oj)

[Diagrams](#_tziv7iqwsth4)

[General picture to illustrate application architecture](#_oi03pa1sutwg)

[General picture to illustrate database structure](#_r4b33kav4hh4)

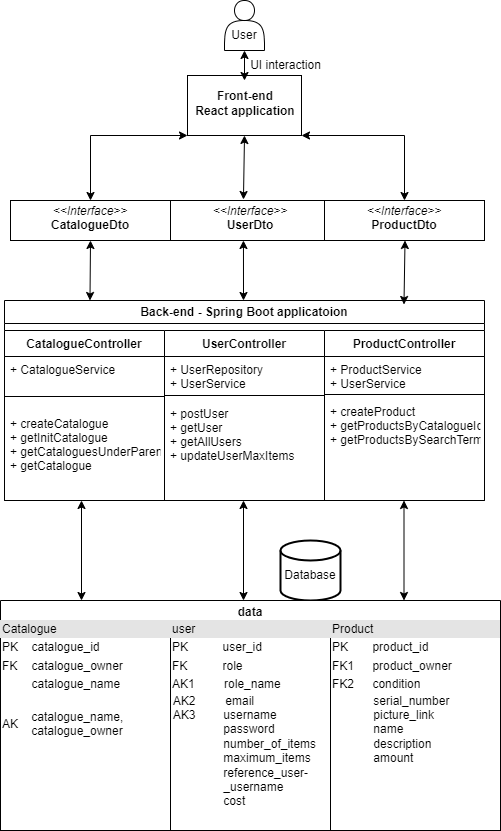
## 

## Overall

The application requeries Java 17, Node 16.18.10 and PostgreSQL 15.1

There is also a docker-compose option for running the application but the back-end needs to be compiled to a jar file so Java 17 is then still needed. The main instructions for running the application are in README.md file which is in the main directory of the application.

Here is a very overgeneralized picture of how the application works, these parts will be discussed in this document.



## Front-end

It uses React, Typescript, Axios and Tailwind. We will go over only the main components of this applications front-end.

### ConnectHandler

is the connection point between front-end and back-end. There are four methods in this file.

#### get

is one of the functions that gets data from the back-end.

The inputs of this function are:

* Item is the request params of the request
* path is the request path
* setComponent is the setState function that will be set when the item is received

The function gets the information using another function getInfo to request info from the backend. After info is gotten then the setState method will be called to set the new state from the info and also the data of the get method is returned.

#### getInfo

is a simpler function then “get” but is used less. This is because get function can get a lot of things done but sometimes more logic is needed to add to the get method so we use this funcion instead.

The inputs of the function are:

* path is the back-end controller path that will be called
* body is the request body that will be sent

If the request goes through then the response from the back-end will be returned.

#### postInfo

is a function for posting to the back-end.

It has 2 parameters:

* path
* body

In returns the returned response from the back-end after the request.

#### putInfo

is the last function for this file and is used for updating values.

It has 2 inputs:

* path
* body

In returns a response from the back-end after the request.

### Pages

#### MainPage

is the users inital page after she/he has logged in.

It has a back button that will navigate the user one page backward.

##### Reload products and catalogues button

that will reload the page, this is useful because sometimes all the products and directories will not load on the front-end, usually after creating a product or catalogue, so the user can use this button to reload the items so that the information is displayed correctly.

##### Create directory button

navigates the user to the directory creation page where they can create a new directory.

##### Create product button

navigates the user to the product creation page where they can create a new directory.

This page also has a grid that displays all the child directories and products in the current directory.

##### Other functionalities

If clicked on a child directory then the page will go to that directory. This will reload the page with the new directories info.

On logging in the page uses useEffect function to set the current user and also if a catalogue isn’t passed to the page then it calles getInitalDirectory() method that get’s the “Home” directory of the user. Otherwise get function from Connection handler is used to make the requests to back-end to get new information about the catalogue.

#### HomePage

is the loggin page of the application.

It has two textfields named “username” and “password” where the user can use their credentials to login to their account by pressing the login button. The login failed if nothing happened. From the class ConnectHandler the function getInfo is used to login to the account.

There is also a button that redirects the user to register page where they can make a new account. The account must have a unique name and email.

#### 

#### AdminPage

is the main page where admin usertype users can look at other accounts statuses.

When the page is initalized get method is used from ConnectionHandler to get all the users in the application.

Each user is represented as a User class in the file User.tsx. If the user is a business user then the admin can change the maximum free items of that user. Each item over the limit is 0.01 cent.

The admin can also see statistics about the user. For instance how many items the user has or how many of those items are in good condition and how many in bad condition.

#### RegisterPage

is the page where a new user can register their account.

Three main parameters that the user has to fill are:

* username, which must be unique
* email, which must be unique
* passwords
* if user is a business user then also:
  + must provide a reference user that exists

After creating the account, the user can click aon the “Login here” button to login to their new account.

#### ProductForm

is the page where the user can create new products.

if has the following values the user has to fill in:

* Name, that has to be unique
* Picture link(I tried to do picture saving but it didn’t work)
* Serial number(can be just a blank)
* Description
* Amount
* Has to choose a condition that the product is in

After filling in all the needed information and clicking on Register button the users new product will be registered. The user is then sent back to the MainPage where they can see their new product. If the product hasn’t loaded the user can click the “Reload products and catalogues button” to reload their current directory and refetch the data.

#### CatalogueFrom

is the page where the user can create a new catalogue.

The only field that the user has to fill in is the “Catalogue name” filed where they have to type a new unique name to their catalogue.

After clicking the “Register” button the user is directed back to the MainPage where they can see their new catalogue. If it dosen’t appear then the user can click the “Reload products and catalogues button” to reload their current directory and refetch the data.

## Back-end

### UserController

is the main controller class that handles users interaction with her/he’s accounts. The path to this class is “/”. It uses UserRepository, which is Users class database connection, and UserService class, that is a service class that handles the main functionalities that apply to users actions, as filed injections.

It has four methods:

* postUser, with PostMapping “/register/user”, that creates a new account for the user using the application.
* getUser, with GetMapping “/user”, that fetches the user by the requests parameters “username” and “password”.
* getAllUsers, with GetMapping “/users”, that fetches all the users for the admin user.
* updateUserMaxItems, with PutMapping “/update”, that updates the users max items status.

### ProductController

is the main controller class that handles product interaction that the user wants to do. The path to this class is “/product”. It users ProductService, which is the main class that handles Products related logic and UserService class, that is a service class that handles the main functionalities that apply to users actions, as filed injections.

It has three methods:

* createProduct, with PostMapping “/create/{catalogueId}/{username}”, where “catalogueId” and “username” are path variables and request body ProductDto as the product that is being created. It creates a new product from the ProductDto class.
* getProductsByCatalogueId, with GetMapping “/get/{parentId}/products” where the “parentId” is the catalogue id that holds the items the user wants to get. This method returns all the products under the catalogue whit the id “parentId”.
* getProductsBySearchTerm, with GetMapping “/get/like” where the request parameters are “username and “searchTerm”. The method returns the users products that match that search term the user requested from the database. This is a basic search function.

### 

### CatalogueController

is the main controller class that handles catalogue interactions that the user wants to do. The path to this class is “/catalogue”. It uses CatalogueService that is the main class that handles Catalogue re;ated logic.

It has four methods:

* createCatalogue, with PostMapping “/create” and request body of CatalogueDto.

If the user wants to create a catalogue this metod will be called and the CatalogueDto has the catalogue parameters that will be created.

* getInitialCatalogue, with GetMapping “/init” and request parameter “username”.

This metod will get the users “Home” catalogue that is the initial catalogue that the user is in when he/she logs in to the account. If user is new and dosen’t have a catalogue then it will be created for the user in this method and the newly created catalogue will be returned.

* getCataloguesUnderParent, with GetMapping “/get/{parentId}/catalogues”, where the path variable is the catalogue id we want to get the child catalogues for.
* getCatalogue, with GetMapping “/get” and request parameters “username” and “catalogueName”. This is a metod for getting a specific catalogue that the user owns with the catalogue name that the user requested.

## 

## Database

Is on PostgreSQL

The schema is called data in this application

It has seven tables:

* roles: Holds the user roles of the applcation
* condition: Holds available product condition statuses
* user: Holds user objects
* catalogue: holds catalogue object data
* catalogue\_catalogue\_reference: holds the child parent references of catalogues
* catalogue\_product\_reference: holds the references of which products are under which catalogue
* product: Holds the product object

## 

## DTOs

### UserDto

Is the object that is passed from the back-end to the front-end and other way as well so that user information can be sent.

Has parameters:

* username
* password
* email
* role
* referenceUserUsername
* maximum\_items
* number\_of\_items
* cost
* items\_status

### CatalogueDto

Is the object that is passed from the back-end to the front-end and other way as well so that catalogue information can be sent.

* username
* catalogueName
* parent
* catalogueId

### ProductDto

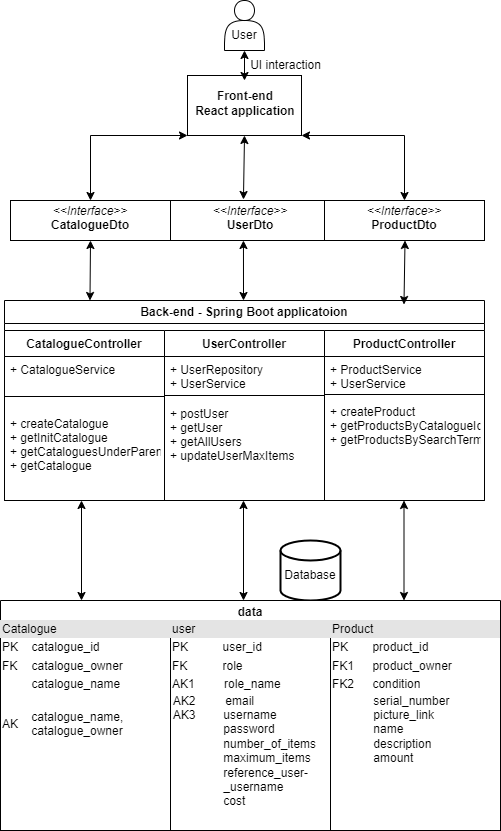
Is the object that is passed from the back-end to the front-end and other way as well so that product information can be sent.

* name
* serialNumber
* pictureLink
* productOwner
* condition
* description
* amount

## 

## Diagrams

### General picture to illustrate application architecture



### General picture to illustrate database structure

