OpenICT - CheckMeIn

Release 0.1

OpenICT

Feb 06, 2023

CONTENTS

OpenICT checkin is a website made by the student to put their own hosted "check in". Together with the OpenICT way of learning, the whole project is managed by the SCRUM process and will be maintained by the students in this semester or internship.

Check out the *Installation* page to set up the program for yourself. Or check out the api if you already have everything set up locally.

Note: This project is under active development and will have frequent changes to the project and this documentation. Make sure to check back often is something is not working as it should.

CHAPTER

ONE

CONTENTS

1.1 Environment

During development, two main sites have been set up to automatically reflect all the changes in the corresponding github branches. These two websites are always in sync with the github repository and will reflect changes within 5 minutes of additions or removals.

1.1.1 Main production website

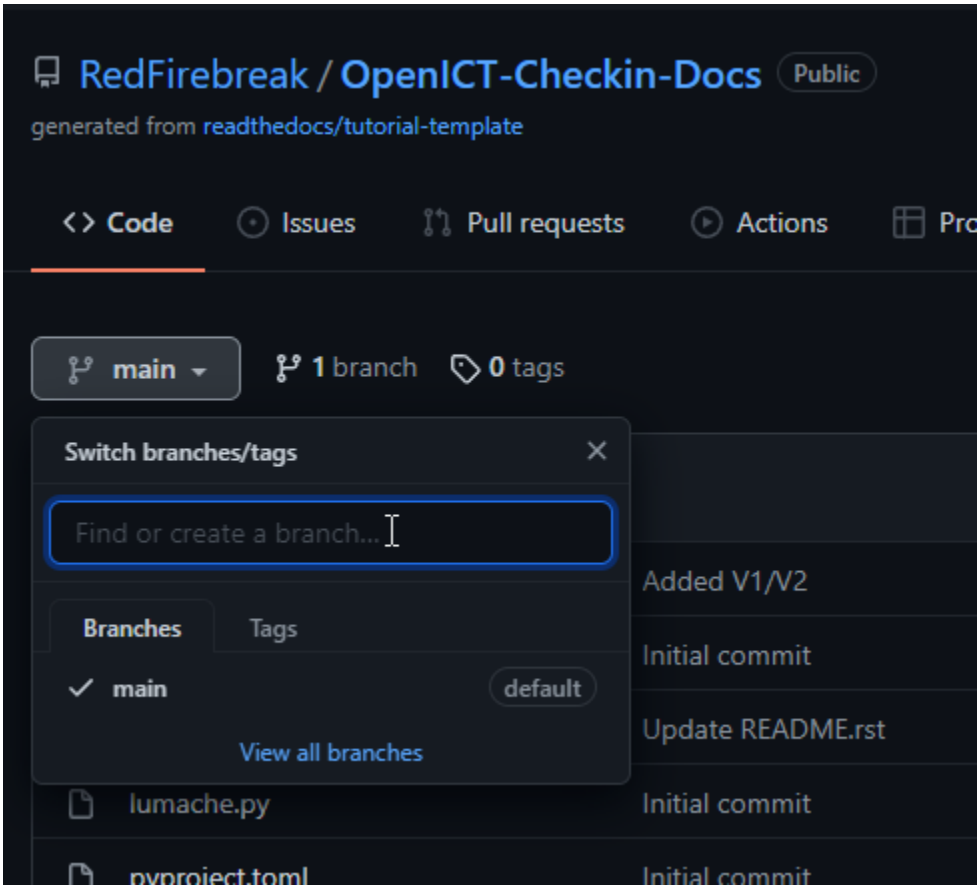
The latest main branch, this version of the website will only get updated on the release of a new version, often at the end of the sprint period.

1.1.2 Main development website

The latest version of the development branch. This version continuesly updates with every addition to the development branch. This website has been set up to spot any extra deployment issues that may arise while updating the main website.

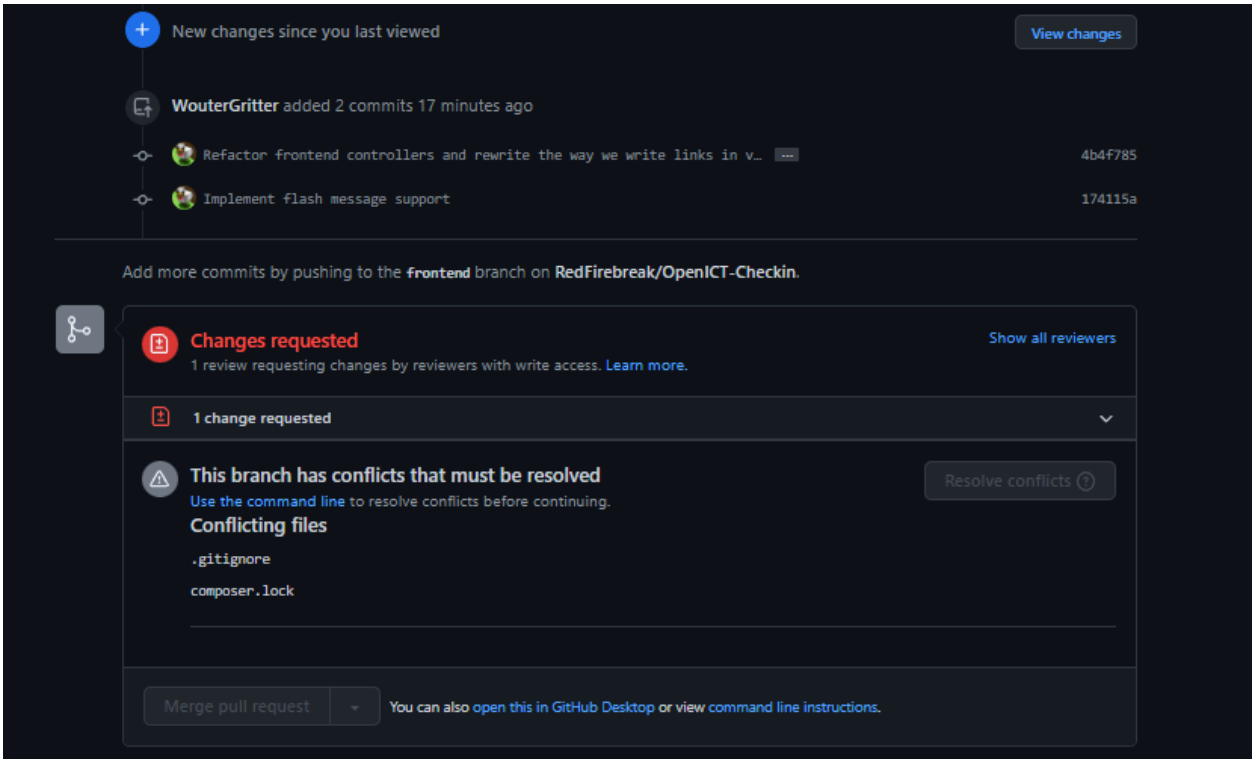
1.1.3 Repository (Github) flow

The Github repository has protected branches. This means that direct commiting/pushing to the master and dev branch is not allowed by default. Instead, the user/programmer is encouraged to create a new branch from the development branch. There are many tutorials to do so, or you can simply ask your team to assist you.

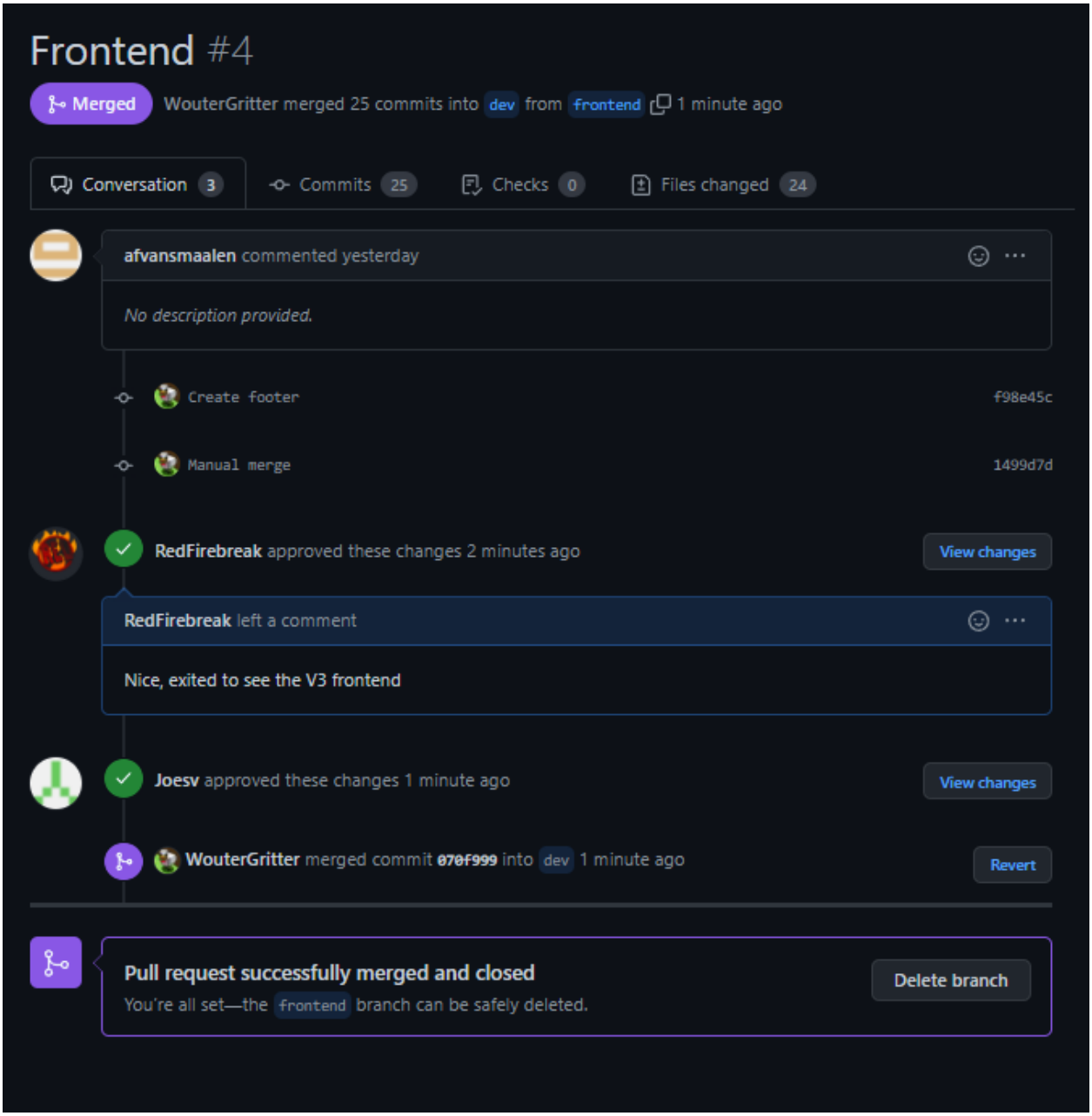


When a new branch is created, a copy is made of the current state of the project. Further on, you can easily work on your changes without interruptions from other commits or pushed. This decreases the amount of merge conflicts.

When the user/programmer is done with their part of code and want to upload it to the development (or even master) branch. They should commit and push their changes to their own made branch. When their branch has the changes included, the user/programmer can create a pull request to merge their state of the project, with the development or master version of the project.



To approve and merge the pull request, another member of the team is required to review the code and approve the changes. This is to prevent "bad" code to enter the project and to make sure multiple team members are aware of the changes. A pull request to the master branch requires two reviews.



1.1.4 Documentation

The documentation of this project will be in continued development and is subject to (many) changes.

1.2 Installation

1.2.1 Requirements

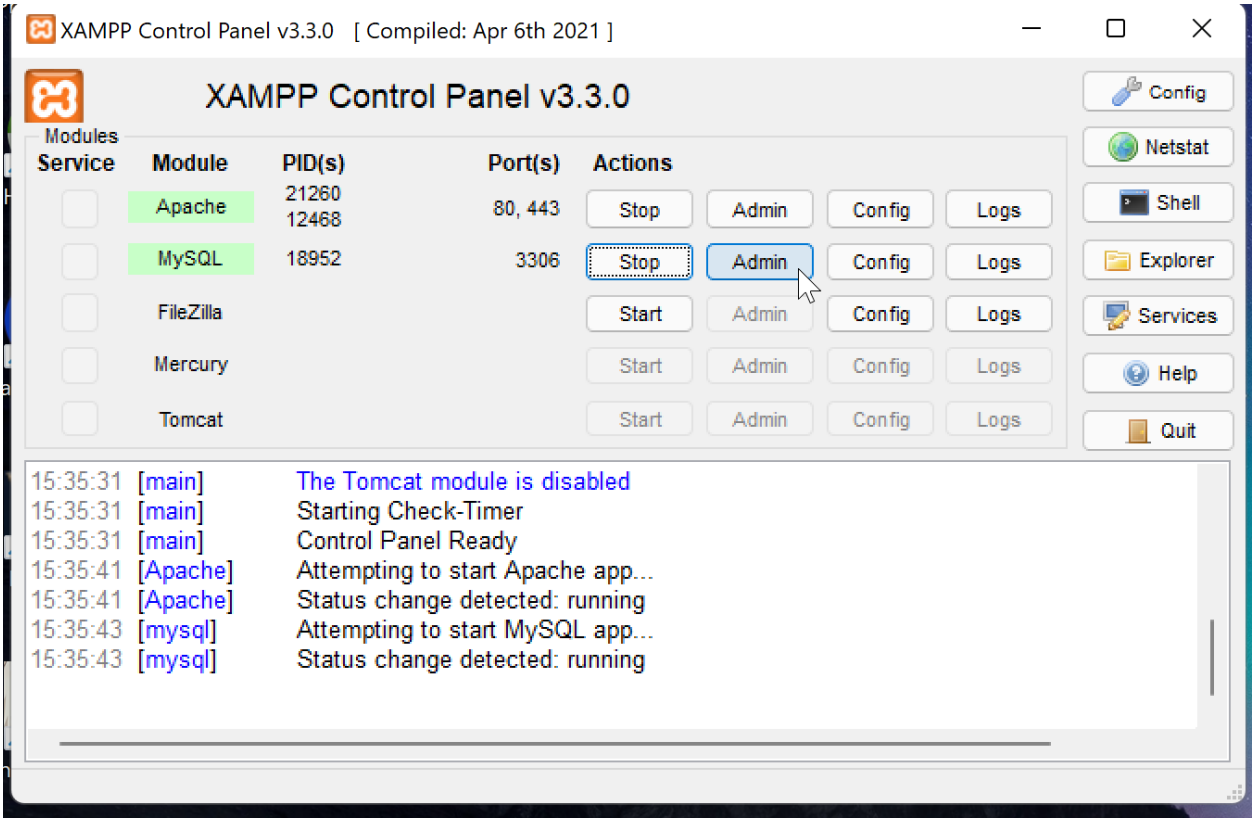
- ``GIT available from commandline <https://git-scm.com/>``_
- ``XAMPP with PHP 8.0 or higher <https://www.apachefriends.org/index.html>``_
- ``Composer <https://getcomposer.org/>``_
- – Syntax error while installing? Go to the specified file and place the path within “” marks.
- PHP installed in PATH variables (Available as a checkmark in the composer installer)
- Access to the restricted github page (Ask RedFirebreak or any collaborator for access)

After making sure that all the requirements are met on your system, you can continue to the next section.

1.2.2 Installation

Note: This is only required if you want to `develop` for the project, want to see or use the project? Go to the development website or the production website. Further, this installation sets up a XAMPP server on your local machine to start developing.

After installing all of the requirements, start your XAMPP control panel, then start the Apache and MySQL. When both of the processes are running and green, click on the *admin* in the MySQL line. This will open up PHPMyAdmin in your default browser.



With this, we are going to create a new, empty database for our project. This is nessecary to make sure the website can store and retrieve data. On this window, click on “nieuw”.

Then, give your database a simple name. For example: `laravel` will do just fine. After filling in a name, click the ‘aanmaken’ button.

The database has been created! You can now close this browser window. After that, to test and develop this project locally, we need to clone the base code from the main repository. To get started **open a powershell or cmd console and peform the following commands below**:

```
(cmd) $ cd {path/to/xamp/htdocs}
(cmd) $ git clone https://github.com/RedFirebreak/OpenICT-Checkin.git
```

Now, preferring on your installation choice, move the insides of the now downloaded folder to your preferred web-space. You do not have to keep the path specified in this tutorial, as long as you know where you store the pulled github files

In the project folder, copy the `.env.example` to a new file called `.env` and **fill in the config file**. By default, you can use `root` as the database password with an empty password.

Note: Make sure to prepare the required database in advance. If no database is present, the application will **NOT** load and you will see a `500: error` message.

After cloning, go into path of the cloned folder and keep the terminal open

```
(cmd) $ cd {path/to/xamp/htdocs/project}
(project/) $
```

And switch to the dev branch for your developing

```
(project/) $ git checkout dev
```

With the still opened terminal, peform the following commands:

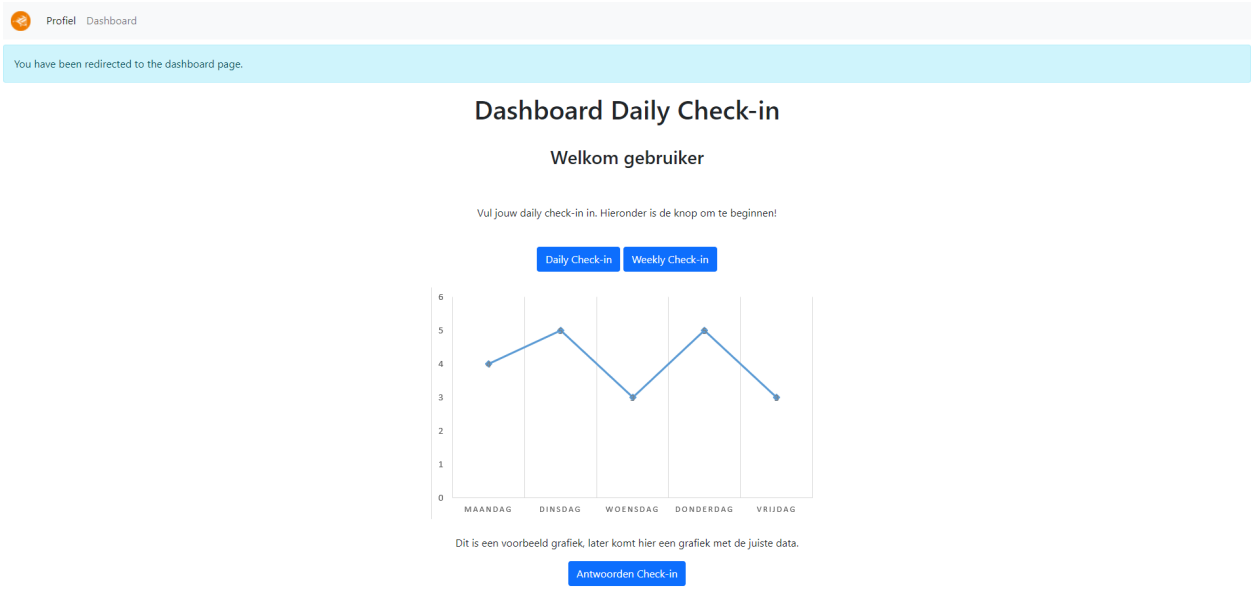
```
(project/) $ composer install
(project/) $ composer update

(project/) $ php artisan key:generate
```

This will install all the required filed, make sure they are updated and set them up for auto-loading. After this, you will generate your own security key for the application. After this, you can begin to prepare the database. In a development situation, you are required to set up your own database, including some dummy data to get started. If you have set up your database connection in the *.env* file, you can start the next commands:

```
(project/) $ php artisan migrate
(project/) $ php artisan db:seed
```

```
(base) PS E:\Program Files (x86)\xampp\htdocs\OpenICT-Checkin> php artisan migrate
v1
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (6.60ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (3.68ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (4.42ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (5.17ms)
Migrating: 2022_03_03_114802_create_daycheckup_table
Migrated:  2022_03_03_114802_create_daycheckup_table (2.74ms)
(base) PS E:\Program Files (x86)\xampp\htdocs\OpenICT-Checkin> php artisan db:seed
v1
Database seeding completed successfully.
(base) PS E:\Program Files (x86)\xampp\htdocs\OpenICT-Checkin>
```

If all goes well, your databse should now be created and filled with some dummy data. Technically, you should now see the project as below! You can also click this link to go to the localhost page.



Note: Error 500 page instead of the project? Or a different error? Make sure to restart the apache server and run `composer update` again to make sure the application can gather all the packages. Otherwise, google the error given for a quick fix, or call for your team! :)

1.2.3 Updating

To update the application, move a cmd to the git cloned project directory and peform the following commands:

```
(cmd) $ cd {path/to/xamp/htdocs/project}
(project/) $
```

Then, with the same cmd screen open:

```
(project/) $ git pull
(project/) $ composer update
```

To deploy the latest database, use the following commands:

```
(project/) $ php artisan migrate
(project/) $ php artisan db:seed
```

The application is now up to date with the dev branch as you should see on the development website

1.2.4 Usage

INCOMPLETE

1.3 Database

The database is automatically generated by Laravel. Here are some usefull commands to keep in mind while interacting with the automatically generated database. All following command should be executed at the root of the project directory.

Note: While trying to make a new table for data, Please do not create any extra tables yourself and use the built in *database->migration* folder.

Note: Make sure the database connection settings in your `.env` file are set accordingly. For a XAMPP install, go to *Installation* and make sure to have followed all the steps.

1.3.1 Usefull command list

- `php artisan migrate` -> Creates the database according to the *database -> migrations* files
- `php artisan migrate:install` -> Only installs the migrate-manager and doesn't run any files.
- `php artisan migrate:status` -> Shows the status of the database migrations
- `php artisan migrate:reset` -> Resets the entire database back to 0.
- ``php artisan db:seed`v -> Fills the database with dummy data (found in *database -> seeders*)
- `php artisan make:migration DESCRIPTION` -> Creates a new migrate job, in the created file you can make your changes to the database
- `php artisan make:model NAME` -> Creates a new model, which can be hooked to the database

To view all the ways you can interact with the database, go to the api page. Here you can find all the available endpoints that can interact with the database to get and store data.

1.4 Creating Tables

- *$table->foreignId('user_id')->constrained();* -> Creates a column called *user_id* that has a foreign key constrain with the column *id* in the user table

1.5 Functionaliteiten

De volgende link redirect naar de aanwezige documentatie over de functionaliteiten van de website / applicatie:

- https://docs.google.com/document/d/1_zgtPjhC6f7ch_OGNXdO3XTsh9qHOM5TyB4ze2sh29A/edit?usp=sharing

---

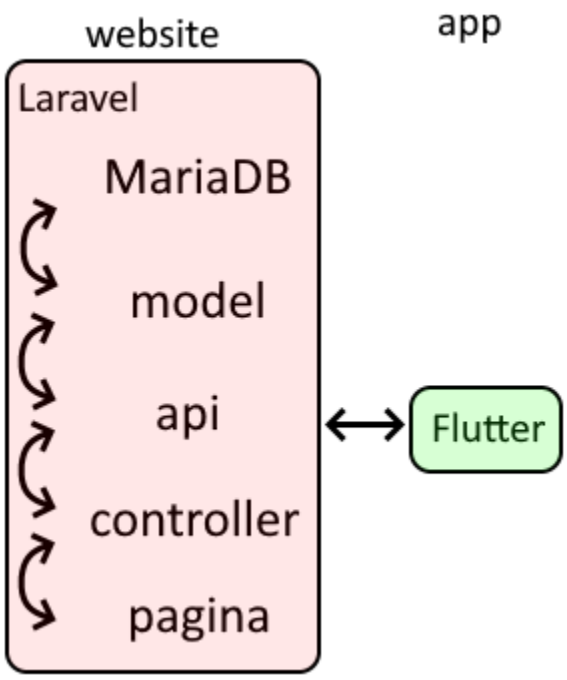
Note: De volgende documentatie is puur geschreven vanwege administratieve redenen. Hierin wordt globaal uitgelegd wat, wat doet en hoe het in elkaar steekt.

1.5.1 Laravel

Het Laravel gedeelte is opgezet in vijf delen:

- Database (MariaDB)
- models
- Api
- controller
- pagina

Elke laag communiceert alleen met onderdelen op dezelfde laag of maximaal 1 laag erboven. Het enige wat er met de laag beneden wordt gedaan is het feliciteren van data aan die laag.



1.5.2 Pagina

Dit is de pagina, Hier wordt de HTML van de pagina gegenereerd. Hier staan ook de loops om bijv. tabellen van users weer te geven.

OpenICT-Checkin/resources/views/pages

1.5.3 Controller

Hier wordt de data via de api's opgehaald en verwerkt om de data vervolgens aan de pagina's te geven. De manier waarop pagina's aan controllers wordt gekoppeld gebeurt in de file web.php

OpenICT-Checkin/app/Http/Controllers/Frontend/

1.5.4 API

In de Applicatie worden alle getters en setters van informatie op de database op de api laag gedaan. De api laag roept de models aan via Eloquent ORM om query's te doen op de database.

Note: Op dit moment zijn alle getters en setters op één api file: OpenICT-Checkin/routes/apiv1.php

1.5.5 Model

Eloquent ORM is een gedeelte query builder dat gebruikt kan worden op de models zodat er op een makkelijke manier query's gemaakt kan worden zonder dat er veel SQL-code geschreven hoeft te worden, afbeelding 2. De mogelijkheid is er om pure SQL-code te gebruiken samen met models, afbeelding 3.

OpenICT-Checkin/app/Models/

```
Answer::where('form_id', $form_id)->where('user_id', $user_id)->get();
```

```
User::select('users.*')->whereRaw('users.id not in (select student_id from `connected_users` where docent_id = ?)', [$docent_id
```

1.5.6 Database

De database is een MariaDB database. Laravel forceert om tabellen te gebruiken die eindigen op s, om zo de tabellen in meervoud te hebben. De database heeft kort voor oplevering een update gehad. Er wordt geen gebruik gemaakt van views of procedures.

De database wordt opgebouwd en gevuld door seeders en migrations. Migrations maken de tabellen en kolommen en seeders stoppen data in de database.

1.5.7 Notifications

Wanneer sommige criteria aanwezig zijn al er een notificatie in deze tabel komen. Op dit moment wordt het gebruikt om de docent te notificeren als de gemiddelde blijheid, van gekoppelde studenten, onder een bepaald niveau komt. De connectie naar Users is ervoor om te kunnen zeggen dat het om een bepaalde user gaat.

1.5.8 Form

De form wordt gebruikt om verschillende vragen te groeperen. De vragen worden opgeslagen in de tabel questions. De koppeltabel naar Users is een futureproof om zo de mogelijkheid te geven om studenten aan verschillende formulieren te koppelen. Omdat er de mogelijkheid is om deze tool te gebruiken op meerdere opleidingen die andere dagelijkse check-ins kunnen hebben.

Note: Een form kan dus meerdere vragen bevatten en kan aan meerdere users gekoppeld zijn.

1.5.9 Answersheets

De tabel om antwoorden te groeperen en te koppelen aan een formulier. In de tabel answers komt ook de vraag te staan hoe de vraag op het moment van schrijven staat. Om zo de mogelijkheid te bieden om vragen aan te passen, maar om zo ook de vraag te hebben waar de student op heeft beantwoord.

1.5.10 Security Profile

In deze tabel komen de default toegangsrechten per account soort. Als er een nieuwe recht gemaakt moet worden, bijv. een nieuwe vragen formulier aanmaken, dan is het de bedoeling dat het een nieuwe kolom in de tabel komt. In de users komen dezelfde kolommen om persoonlijke rechten mogelijk te maken.

1.5.11 Projects

Hier komen projecten in te staan waar users aan vast gemaakt kunnen worden. Er is een tussen tabel omdat sommige users aan meerdere projecten kunnen werken.

Connected Users In deze tabel worden de docent/student relaties opgeslagen.

1.5.12 Jobroles

In deze tabel worden de jobroles opgeslagen die de student kan kiezen om als doel te hebben. De tussentabel met levels is er omdat de jobroles bepaalde niveaus hebben van competenties.

1.5.13 Comptencies, Levels, Architectures en Activities

Deze tabellen samen vormen de HBO-I competenties. De users kunnen een bepaald niveau van competenties selecteren om als doel te hebben of om de niveaus van die user op te slaan. https://hbo-i.nl/domeinbeschrijving/

1.5.14 Submissions

Hier worden de beroepsproducten opgeslagen. Bij elk product moet staan bij welk competentie het hoort.

1.5.15 Competency Meeting

Hier worden de weekelijkse gesprekken die over de hbo-i competenties gaan opgeslagen. In de tabel wordt ook het weeknummer opgeslagen met connectie naar competenties en criterias

1.5.16 Criteria

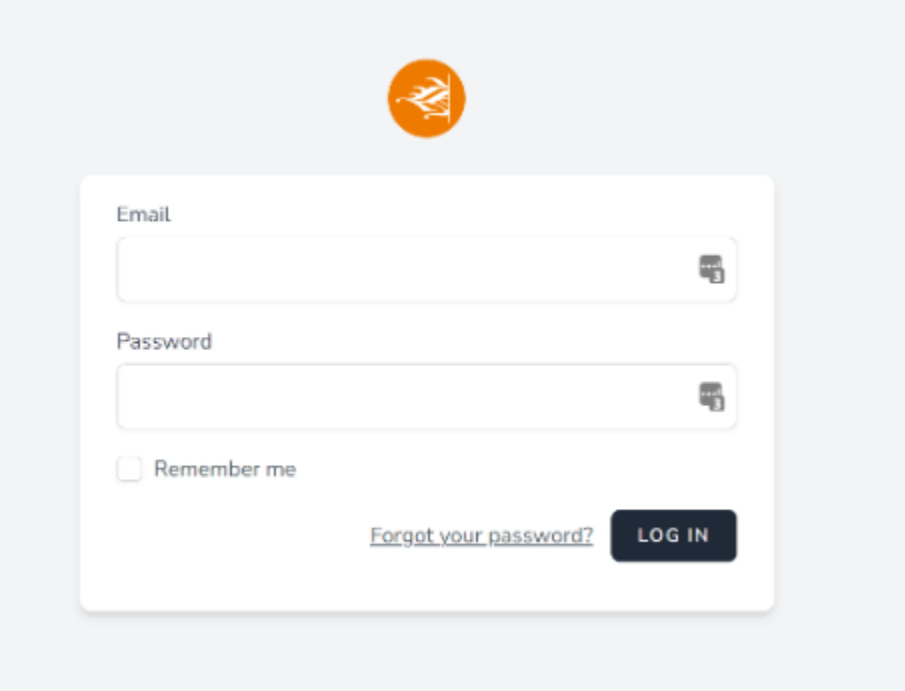
Hier worden de criteria, die in de wekelijkse gesprekken staan, opgeslagen

1.5.17 Aanbevelingen

1.5.18 Schermuitleg:

1.5.19 Inlogscherm

Dit scherm wordt gebruikt om in te loggen, tevens is het vanuit dit scherm mogelijk om naar het wachtwoord vergeten scherm te gaan. In dit scherm worden de inloggegevens ingevuld om doorgewezen te worden naar het dashboard.



1.5.20 Bestanden

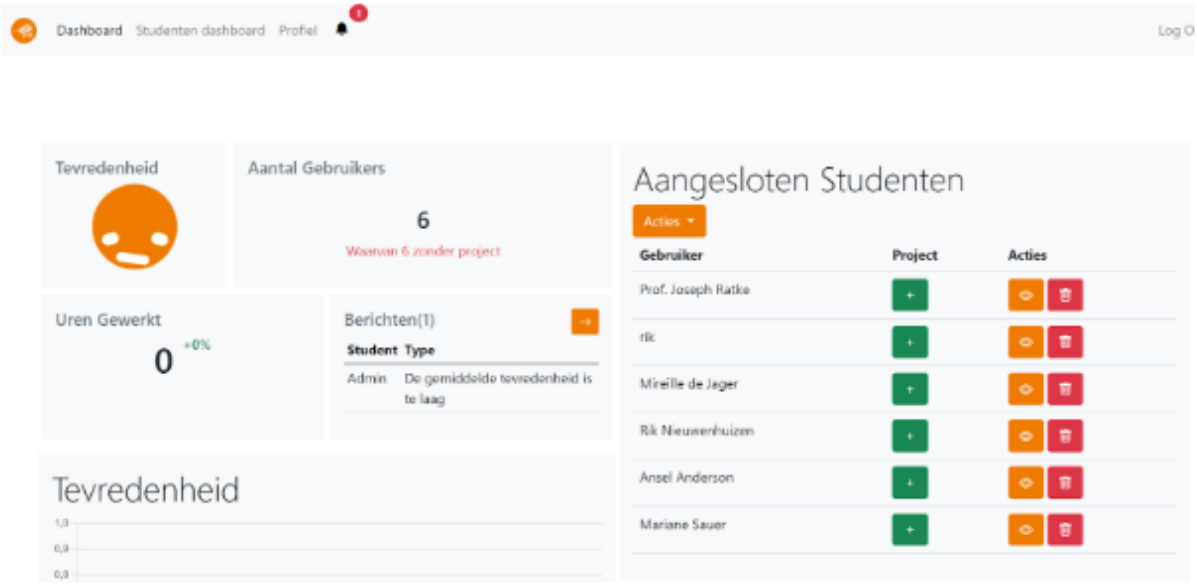
De bestanden die horen bij dit scherm, en dus de functionaliteit beschreven hebben zijn:

- AuthenticatedSessionController.php
- login.blade.php

Note: Binnen het project zijn er geen belangrijke aanpassingen gedaan naast het aanpassen van het design. Design is terug te vinden in login.blade.php en is geschreven in bootstrap 5. Voor uitleg van bootstrap refereer ik naar de documentatie.

1.5.21 Dashboard(docent)

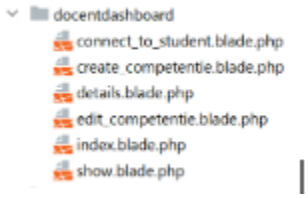
Het dashboard van de docent heeft als doel gemakkelijk een inkijk te geven in de status van de studenten. Het is op het dashboard mogelijk om de gemiddelde tevredenheid van de gebruikers te zien, het aantal gebruikers dat er zijn (met een opmerking met hoeveel studenten nog een project zoeken), er kan gezien worden hoeveel uren de studenten hebben gewerkt vandaag aan de projecten (met een percentueel verschil erbij met de dag van gister), het bovenste bericht kan worden bekeken, de tevredenheid van de studenten per student in grafiekvorm (filterbaar) en tot slot kunnen de aangesloten(docenten kunnen studenten selecteren die zij kiezen te volgen te begeleiden.



1.5.22 Bestanden

De functionaliteiten en variabelen die nodig zijn om dit dashboard tot stand te laten komen zijn terug te vinden in de DocentenDashboardController.php. Binnen de controller zijn een aantal functies beschreven. Het php-bestand is voorzien van documentatie voor een nadere verklaring van wat de functies precies doen.

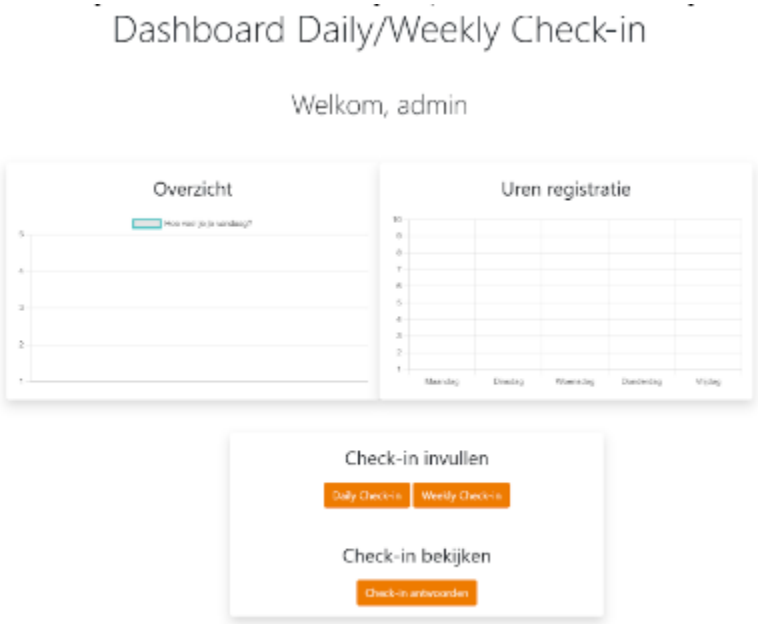
Het design van het dashboard het dashboard en de functionaliteiten die het dashboard heeft zijn te vinden in het mapje 'views->pages->docentdashboard'. De volgende views zijn hier terug te vinden:



De functionaliteiten die gekoppeld zijn aan de views zijn terug te vinden in de controller zoals bovenstaand staat beschreven.

1.5.23 Dashboard (Studentendashboard)

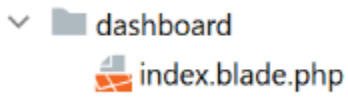
Het dashboard wordt gebruikt om studenten een snel overzicht te geven van de tevredenheid over een periode van de afgelopen week. Daarnaast geeft dit dashboard weer op welke dagen hoeveel uur gewerkt is aan een project. Tot slot is het mogelijk om vanuit dit dashboard je antwoorden te bekijken, bewerken en verwijderen.



1.5.24 Bestanden

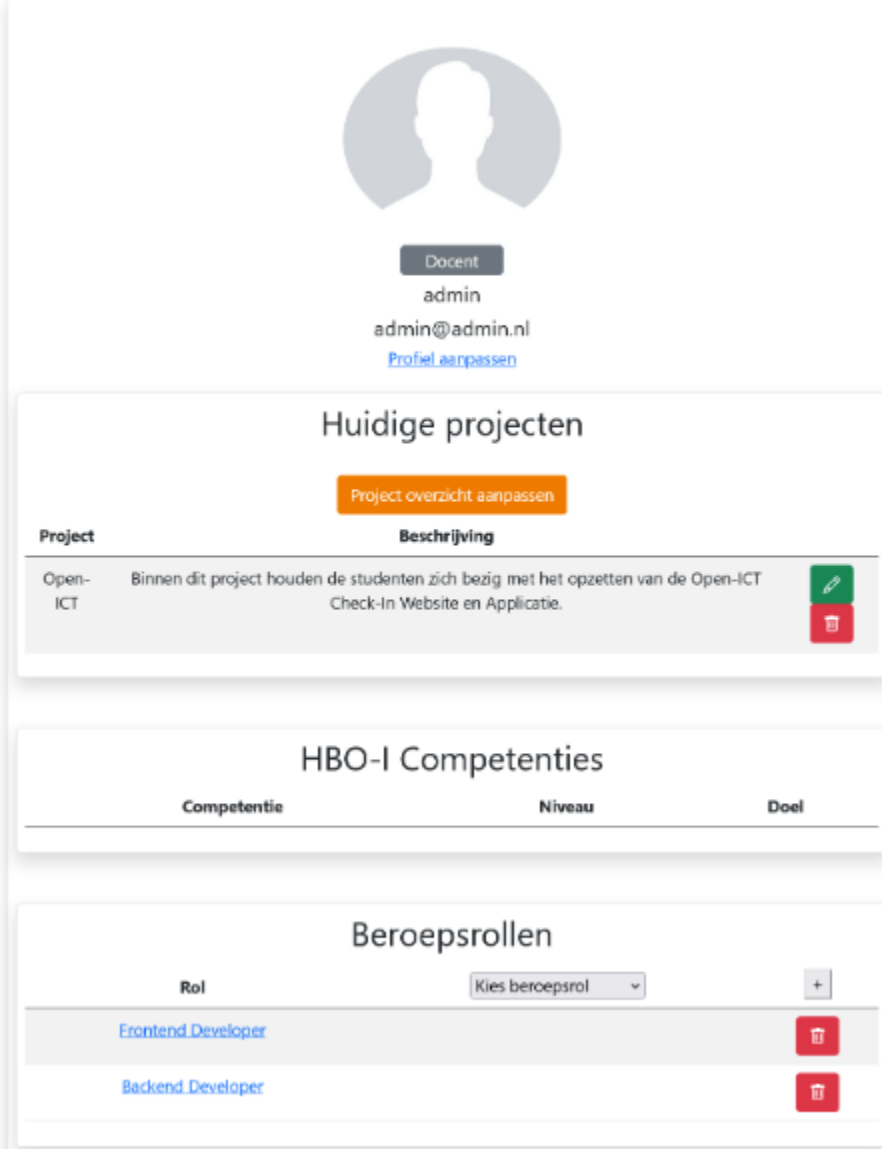
De functionaliteiten en variabelen die nodig zijn om dit dashboard tot stand te laten komen zijn terug te vinden in de DashboardController.php. Binnen de controller zijn een aantal functies beschreven. Het php-bestand is voorzien van documentatie voor een nadere verklaring van wat de functies precies doen.

Het design van het dashboard het dashboard en de functionaliteiten die het dashboard heeft zijn te vinden in het mapje 'views->pages->dashboard'. De volgende views zijn hier terug te vinden:



1.5.25 Profiel

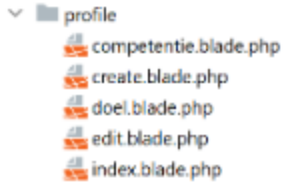
De profiel pagina geeft een overzicht van de gegevens van een gebruiker. Belangrijke gegevens hier zijn de projecten, de competenties en niveau's en tot slot de beroepsrollen. Ook is het mogelijk om hier gegevens aan te passen.



1.5.26 Bestanden

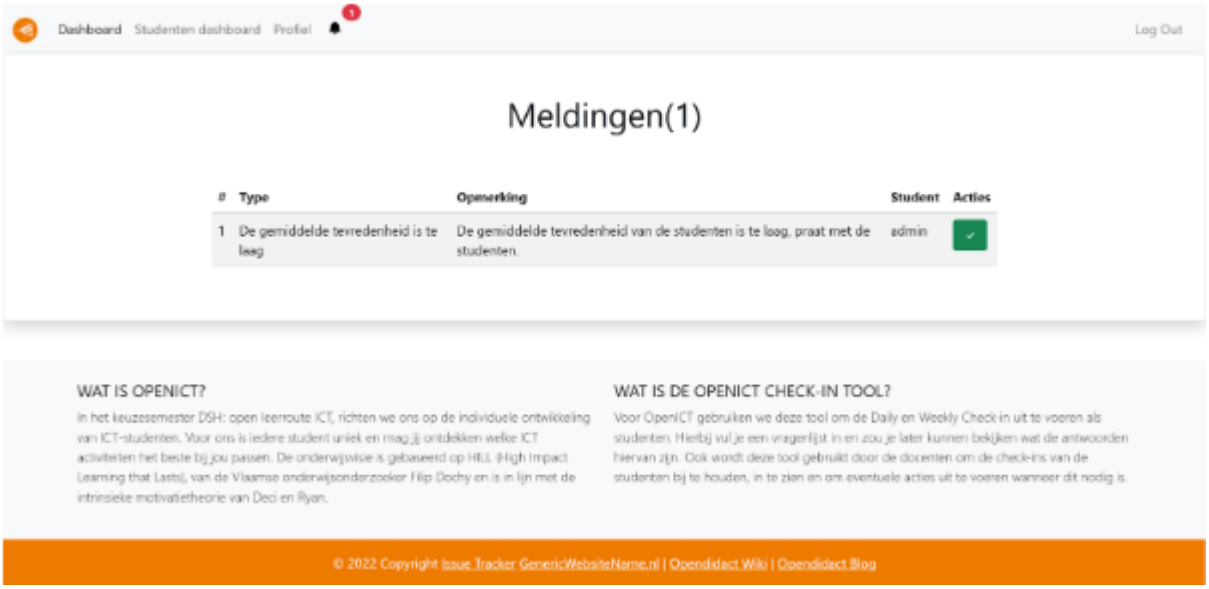
De functionaliteiten en variabelen die nodig zijn om dit dashboard tot stand te laten komen zijn terug te vinden in de ProfileController.php. Binnen de controller zijn een aantal functies beschreven. Het php-bestand is voorzien van documentatie voor een nadere verklaring van wat de functies precies doen.

Het design van het dashboard het dashboard en de functionaliteiten die het dashboard heeft zijn te vinden in het mapje 'views->pages->profile'. De volgende views zijn hier terug te vinden de uitleg van de functionaliteiten zijn terug te vinden in de controller.



1.5.27 Meldingen

De meldingen pagina geeft een tabel weer waarin kan worden bekeken of er openstaande meldingen zijn. Deze meldingen worden gemaakt wanneer de gemiddelde tevredenheid 2 of lager is en verdwijnen als deze weer boven de 2 is. Daarnaast zijn er meldingen wanneer een student invult dat hij ontevreden is, aangeraden is dan om contact op te nemen met de betreffende student.



1.5.28 Bestanden

De functionaliteiten en variabelen die nodig zijn om dit dashboard tot stand te laten komen zijn terug te vinden in de NotificationController.php. Binnen de controller zijn een aantal functies beschreven. Het php-bestand is voorzien van documentatie voor een nadere verklaring van wat de functies precies doen.

Het design van het dashboard het dashboard en de functionaliteiten die het dashboard heeft zijn te vinden in het mapje 'views->pages->notification'. De volgende views zijn hier terug te vinden de uitleg van de functionaliteiten zijn terug te vinden in de controller.

Het genereren van de meldingen wordt gedaan in DocentDashboardController.php op regel 139 en FormInteraction-Controller.php op regel 68.

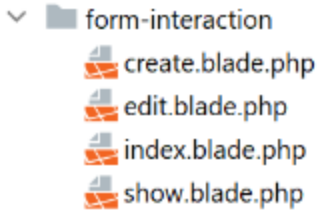
1.5.29 Check-in pagina's

De check-in pagina is waar de student aangeeft hoe het met de student gaat, deze bestaat uit 5 vragen die worden opgeslagen in de database en vervolgens worden weergegeven in het dashboard van de student en dashboard (uitleg hierboven te vinden). De wekelijkse check-in wordt niet besproken omdat het concept anders wordt.

1.5.30 Bestanden

De functionaliteiten en variabelen die nodig zijn om dit dashboard tot stand te laten komen zijn terug te vinden in de FormInteractionController.php. Binnen de controller zijn een aantal functies beschreven. Het php-bestand is voorzien van documentatie voor een nadere verklaring van wat de functies precies doen.

Het design van het dashboard het dashboard en de functionaliteiten die het dashboard heeft zijn te vinden in het mapje 'views->pages->form-interaction'. De volgende views zijn hier terug te vinden de uitleg van de functionaliteiten zijn terug te vinden in de controller.



1.6 API

There are 3 API versions available of which 2 are deprecated and no longer in use. API V1 is still being used for authentication.

1.6.1 V1 (Deprecated, use V3 instead)

/api/v1/sanctum/token:
Functionality that checks if the user filled in the right login credentials, if that's the case, a token will be created and added to the User table. Requires:

E-mail Password Device_name

/api/v1/sanctum/logout:
Functionality that logs out an user.

/api/v1/user:
Returns the current logged in user.

/api/v1/postFormApi:
Functionality that posts the filled in Form data to the database. Requires:

user_id form_id questions_array questions_text array_answers

/api/v1/getUserFormsRecent/{user_id}:
Returns the latest answer sheet of a filled in form of a specific user.

/api/v1/getDailyWeekBetween/{user_id}/{from}/{to}:
from and to need to be a date written as: yyyy-mm-dd Returns an answer sheet that has been submitted between the given timestamps.

/api/v1/getUserFormsMonth/{user_id}: <<<<<< Geen idee wat dit precies moet doen, returned ook geen values op postman bij mij >>>>>>>>>

/api/v1/updateAnswer:
Allows an user to update their answer sheet. Requires:

id question_array question_text array_answers

/api/v1/getTotalDaily/{user_id}:
Returns the total amount of daily check-ins of a specific user.

/api/v1/getTotalWeekly/{user_id}:
Returns the total amount of weekly check-ins of a specific user.

/api/v1/getTotalHoursByUser/{user_id}:
Returns the total amount of hours filled in by a specific user.

/api/v1/getTotalHoursBetweenByUser/{user_id}/{from}/{to}:
Returns the total amount of hours filled in by a specific user between two dates.

/api/v1/getHoursPerDayWeek/{user_id}/{from}/{to}
Returns the amount of hours filled in by a specific user sorted per answersheet between two dates.

/api/v1/getHoursPerDay{user_id}
returns the amount of hours filled in by a specific user per answersheet

/api/v1/getTotalHoursBetweenTotal/{from}/{to}
returns total amount of hours filled in between two dates

/api/v1/user
returns all users

/api/v1/getVersion:
Returns a JSON with the used version.

/api/v1/testAdmin:
Returns a 'works' value.

/api/v1/getUserAll
Returns a JSON with all users.

/api/v1/getUserAdmins:
Returns a JSON with all users who have their role set as 1.

/api/v1/getUserStudents:
Returns a JSON with all users who have their role set as 0.

/api/v1/getConnectedUsers/{docent_id}:
Returns a JSON with all users/students connected to a specific docent.

/api/v1/getAllProjectsOfMyStudents/{docent_id}:
Returns a JSON with all projects of students connected to a specific docent.

/api/v1/getAllAnswersOfMyStudents/{docent_id}:
Returns all answers of students connected to a specific docent.

/api/v1/getAllStudentsNotMine/{docent_id}:
Returns all users that are not connected to a specific docent.

/api/v1/getFirstAnswersOfMyStudents/{docent_id}
returns all first answers of answersheets made by students of {docent_id}

/api/v1/removeLinkToStudent/{student_id}:
Allows an user to remove a student from their group/ unlink

/api/v1/connectToStudent/{docent_id}/{student_id}:
Allows a docent to connect a student to himself.

/api/v1/getUser/{id}:
Returns a JSON with the user associated with id.

POST `/api/v1/editAccount`

Changes the user account based on the given information. Requires all values, empty values will not be changed "". The following example will change the email address of the user id 1.

```
>>> class WordCounter(Document):
```

… … {

… "id":1, … "name": "", … "email": "admin@admin.admin", … "password": "" … }

/api/v1/addUser:
Expects : name, email, password and role Example:

{

"name": "John Doe",

"email": "a@a.a",

"password": "password",

"role": 1

}

/api/v1/editAnswer:
Allows an user to edit their Answer sheet. Requires

id question_array question_text question_answersgetUserAnswersheets

/api/v1/getUserForms/{user_id}:
Returns a JSON with all answers of user user_id.

/api/v1/getUserAnswersheets/{user_id}:
Returns a JSON with the id's of all answersheets filled in by {user_id}

/api/v1/getUserAnswers/{user_id}:
Returns all answers of {user_id}

/api/v1/getForm/{id}:
Returns a JSON with the form form_id and its associated questions.

/api/v1/getDaily:
Returns a JSON with the daily check-in form (form_id 1).

/api/v1/getWeekly:
Returns a JSON with the weekly check-in form (form_id 2)

/api/v1/getAnswersById/{id}:
Returns a JSON of the answers entry of answer.id {id}

/api/v1/getAnswersByAnswersheet/{id}:
Returns a JSON of the answers of answersheet {id}

/api/v1/saveFormAnswers:
Saves the answers in the database. Expects:

user_id -> The id of the user who answered the form. form_id -> The id of the form that is filled in. array_answers -> An array of the answers in JSON format.

Example:
{

"user_id": "1",

"form_id": "1",

"array_answers" : {"boe": "hallo"}

}

/api/v1/deleteAnswer/{id}:
Deletes all answers and ansersheeets connected to answersheet {id}

/api/v1/getDailyCreatedAtBetweenUser/{from}/{to}/{user_id}

Returns a JSON with the created_at date that's between two provided dates by a specific user.

Example : In postman create a request, get the following raw data in JSON format : {"user_id" : "1",

"form_id" : "1", "date1" : "2022-03-14 10:22:00", "date2" : "2022-03-14 10:37:13"}

/api/v1/getTotalUsers:

Returns a JSON with all users.

/api/v1/editQuestion:

Allows the user to edit a question title in the questions table of database. Only allowed by admin user. Expects:

id > The id of the question. title > Title of the question. data > Data of the question.

Example: {

"id": "2",

"title": "Question 2 test",

"data": "1-5",

}

/api/v1/createQuestion

Allows the user to create a new question in the database. Only allowed by admin user. expects:

form-id -> The id of the form (Daily or weekly) qdata -> Data of question title -> Title of the question type -> Type of the question (Text, radio or slider)

Example: {

"form_id": "1",

"qdata": "test data",

"title": "test title",

"type": "text"

}

//alle competentie routes zijn achterhaald en moetten opnieuw geschreven worden /api/v1/editCompetentieNiveau:

Allows an user to edit a competentie niveau. Requires:

user_id competentie_id niveau

/api/v1/editCompetentieDoel:

Allows an user to edit their competentie doel. Requires:

user_id competentie_id doel

Creates a new competentie, Only allowed by admin. expects:

name: the name of the competentie

Example:

{

"name": "backend developer"

}

/api/v1/editCompetentie
Edits an existing competentie, Only allowed by admin. expects:

name: the new name of the competentie id: of the competentie

Example:

{

"id": 1,

"name": "backend deloper"

}

/api/v1/delCompetentie
Removes an existing competentie, Only allowed by admin. expects:

id: of the competentie

Example:

{

"id": 1

}

/api/v1/getAllCompetenties
Returns all competenties

/api/v1/getCompetentieById/{competentie_id}
Returns the specific competentie

/api/v1/addCompetentieToUser
Adds a competentie to a User, Only allowed by admin user. Expects:

user_id, competentie_id

Example:

{

"user_id": 1,

"competentie_id": 3

}

/api/v1/delCompetentieToUser
Removes a competentie from a user, Only allowed by admin user. Expects:

id

Example:

{

"id": 1

}

/api/v1/getAllCompetentiesOfAllUsers
Returns arrays of competenties connected to users, Only allowed by admin user. Example:

{

"1": [

```
        {
          “id”: 3,
          “competentie_id”: 3,
          “user_id”: 1,
          “created_at”: “2022-03-17T11:26:41.000000Z”,
          “updated_at”: “2022-03-17T11:26:41.000000Z”,
          “name”: “backend developer”
        },
        {
          “id”: 2,
          “competentie_id”: 2,
          “user_id”: 1,
          “created_at”: “2022-03-17T11:09:51.000000Z”,
          “updated_at”: “2022-03-17T11:09:51.000000Z”,
          “name”: “frontend developer”
        }
    ],
    “186”: [
        {
          “id”: 3,
          “competentie_id”: 3,
          “user_id”: 186,
          “created_at”: “2022-03-17T11:26:41.000000Z”,
          “updated_at”: “2022-03-17T11:26:41.000000Z”,
          “name”: “backend developer”
        },
        {
          “id”: 2,
          “competentie_id”: 2,
          “user_id”: 186,
          “created_at”: “2022-03-17T11:09:51.000000Z”,
          “updated_at”: “2022-03-17T11:09:51.000000Z”,
          “name”: “frontend developer”
        }
    ]
}
```

/api/v1/getCompetentiesByUser/{comp_id}/{user_id}:
returns a list of competenties that are connected to the user

/api/v1/getAllCompetentieByUser/{comp_id}/{user_id}:
returns all competenties by user.

/api/v1/checkFilledIn/{user_id}/{form_id}
Checks the database if a daily check-in has been filled in already or not. The 'ProfileController' handles this API and returns a warning message if the check-in has been filled in.

/api/v1/getProjectsByUser/{user_id}
Returns a list of projects connected to a specific user.

/api/v1/newProject
Allows an admin user to create a new Project. Requires:

name: the name of the project. description: a small description of the project.

Example:

{

"name": "Check-In Website & Applicatie", "description": "Hier komt een algemene beschrijving"

}

/api/v1/newUserProject
Allows an admin user to connect an user to an existing project. Requires:

project_id: The id of the project user_id: The id of the user

/api/v1/editProject
Allows an admin user to edit an existing project name and description. Requires:

name: the name of the project. description: a small description of the project. id: the id of the project you want to edit.

/api/v1/getProjectByID/{id}
Returns an array of the values of the relevant project.

/api/v1/getProjectIdByUserId/{user_id}
Returns an array of information of the project connected to a specific user.

Example: If admin is connected to project 1 (Check-In) this function will return this project.

/api/v1/getAllProjects
Returns array values of all present projects.

/api/v1/deleteProject/{id}
Allows an admin user to delete a certain project, which is selected by ID.

/api/v1/getAllJobroles
Returns an array of all existing jobroles

/api/v1/getJobRolesByUser/{user_id}
Returns an array value of all jobroles connected to a specific user.

/api/v1/deleteUser/{id}
Allows an user to COMPLETELY delete an existing user from the database.

/api/v1/deleteJobRole/{id}
Allows an user to delete a specific Jobrole from the database.

/api/v1/addJobrole
Allows an user to connect an user to a jobrole Requires:

user_id: ID of the user u want to add the jobrole to. jobrole_id: the ID of the specific jobrole you want to add to the user.

/api/v1/newNotification
Allows the application to create a new notification. Requires:

user_id: ID of the user. type: Type of notification data: Data/description of the notification

/api/v1/getAllNotifications
Returns an array of values of all existing notifications.

/api/v1/getNotificationDetails/{id}
Returns an array of details of a specific notification. Requires:

ID: ID of the specific notification.

/api/v1/getNotificationType/{id}
Returns an array with the 'type' value of a specific notification. Requires:

ID: ID of the specific notification.

/api/v1/getAmountOfNotifications
Returns the total amount of existing notifications.

/api/v1/delNotification/{id}
Allows an user to delete an existing notification.

/api/v1/getJobRolesByUser/{user_id}
Returns an array of all jobroles connected to a specific user. Requires:

user_id: ID of the specific user.

1.6.2 V2 (Deprecated, use V3 instead)

/api/v2/getVersion:
Returns a JSON with the used version.

1.6.3 V3

1.6.4 Authenticatie & Autorisatie

Om je te authenticeren voor de API, vraag je eerst een token op via `/api/v1/sanctum/token`. De token die je van de API krijgt, zet je in een header als `Authorization: Bearer {token}`.

Voor het gebruik van de API moet je geauthenticeerd zijn. Als dit niet het geval is, krijg je een `401` terug. Ook moet je de juiste rechten hebben om bepaalde API endpoints te kunnen gebruiken. Een student kan bijvoorbeeld geen andere student aanmaken, maar een docent of admin kan dit wel.

1.6.5 API V3 Endpoints

Hieronder een opsomming van de API endpoints die gebruikt kunnen worden in combinatie met de geïmplementeerde *Entiteiten & Relaties*. Voor elke API call geldt de prefix `/api/v3`.

GET `/entity` of POST `/entity/search`
Vraag een collectie entiteiten op, eventueel in combinatie met *Search parameters*.

Note: Voorbeeld

Zoek alle users met een email gelijk aan s.tudent@st.hanze.nl:

GET `/users`

```
{
        "email": "s.tudent@st.hanze.nl"
}
```

Note: Voor deze endpoint kun je twee verschillende URLs gebruiken. De reden hiervoor is dat je in sommige gevallen geen request body mee kan sturen met een **GET** request. Hiervoor is een alternatieve **POST** request gemaakt.

POST `/entity`
Voeg een nieuwe entiteit toe.

Note: Voorbeeld

Voeg een nieuwe user toe:

POST `/users`

```
{
        "name": "Student",
        "email": "s.tudent@st.hanze.nl",
        "password": "geheimstudentenwachtwoord"
}
```

GET `/entity/id`
Haal een specifieke entiteit op.

Note: Voorbeeld

Haal de user met id 1 op:

GET `/users/1`

PUT `/entity/id`
Pas een entiteit aan.

Note: Voorbeeld

Verander de email van een user naar d.ocent@pl.hanze.nl:

PUT `/users/1`

```
{
        "email": "d.ocent@pl.hanze.nl"
}
```

DELETE `/entity/id`

Verwijder een entiteit.

Note: Voorbeeld

Verwijder de user met id 1:

DELETE `/users/1`

GET `/entity/id/relation` **of POST** `/entity/id/relation/search`

Haal de relaties van een specifiek model op, eventueel met *Search parameters*.

Note: Voorbeeld

Bekijk de checkins van de user 1, waarvan de mood_score 5 is:

GET `/users/1/daily-checkins`

```
{
        "mood_score": 5
}
```

Bekijk alle projecten waaraan user 1 gekoppeld is:

GET `/users/1/projects`

Note: Voor deze endpoint kun je twee verschillende URLs gebruiken. De reden hiervoor is dat je in sommige gevallen geen request body mee kan sturen met een **GET** request. Hiervoor is een alternatieve **POST** request gemaakt.

POST `/entity/id/relation`

Voeg een nieuwe relatie toe aan een entiteit.

Note: Voorbeeld

Maak een nieuwe daily checkin aan voor user 1:

POST `/users/1/daily-checkins`

```
{
        "mood_score": 5,
        "mood_description": "Toppie",
        "hours_worked": 6,
        "comment": "Heb je een scooter?"
}
```

> **Warning:** Deze API call werkt enkel voor `hasMany` relaties (*Entiteiten & Relaties*)

GET `/entity/id/relation/id`
Haal een specifieke relatie bij een entiteit op.

> **Note: Voorbeeld**
>
> Haal het project met id 1 op, als deze bij user 1 hoort:
>
> **GET** `/users/1/projects/1`

PUT `/entity/id/relation/id`
Koppel twee entiteiten aan elkaar.

> **Note: Voorbeeld**
>
> Koppel project 1 aan user 1:
>
> **PUT** `/users/1/projects/1`

> **Warning:** Deze API call werkt enkel voor `belongsToMany` relaties (*Entiteiten & Relaties*)

DELETE `/entity/id/relation/id`
Verwijder een relatie of koppel twee entiteiten los.

> **Note: Voorbeeld**
>
> Verwijder de daily checkin met id 1 van user 1:
>
> **DELETE** `/users/1/daily-checkins/1`
>
> Koppel project 1 los van user 1:
>
> **DELETE** `/users/1/projects/1`

1.6.6 Search parameters

Voor de API endpoints

GET `/entity` of **POST** `/entity/search`

en

GET `/entity/id/relation` of **POST** `/entity/id/relation/search`

en gedeeltelijk

GET `/entity/id` en **GET** `/entity/id/relation/id`

kun je search parameters meegeven in de vorm van een JSON request body. Dit is handig om te kunnen bepalen hoeveel informatie je terug krijgt wanneer je data opvraagt.

Er zijn een aantal mogelijkheden voor het beïnvloeden van de response data. Deze methodes worden hieronder met voorbeelden uitgelegd:

Velden

Je kan de inkomende data filteren op basis van de database velden. Stel je doet de volgende request:

GET `/api/v3/projects`

Dan krijg je als response het volgende terug:

```
[
        {
                "id": 1,
                "name": "Check-in",
                "description": "Het ontwikkelen van een feedback en check-in tool"
        },
        {
                "id": 2,
                "name": "Powerchainger",
                "description": "Apparaten herkennen adhv het verbruiksprofiel"
        },
        {
                "id": 3,
                "name": "Colloquium app",
                "description": "Maken van een applicatie waarin voordrachten kunnen␣
↪worden bijgehouden"
        },
        {
                "id": 4,
                "name": "ACS",
                "description": "Opruimen van ACS coderepo"
        }
]
```

Wil je alleen de Colloquium app vinden? Dan kan je het volgende doen:

Request:

```
{
        "name": "Colloquium app"
}
```

Response:

```
[
        {
                "id": 3,
                "name": "Colloquium app",
                "description": "Maken van een applicatie waarin voordrachten kunnen␣
↪worden bijgehouden"
        }
]
```

Maar wat nou als je de naam niet precies weet? Dan is er de volgende mogelijkheid:

Request:

```
{
        "name": {
                "like": "collo"
        }
}
```

Response:

```
[
        {
                "id": 3,
                "name": "Colloquium app",
                "description": "Maken van een applicatie waarin voordrachten kunnen
↪worden bijgehouden"
        }
]
```

Naast de *operator* `like`, kun je voor numerieke velden ook < (kleiner dan) en > (groter dan) gebruiken.

Relaties

Je kan ook aangeven welke relaties je terug wil zien in de response door middel van een `with` array. Stel je bekijkt de user met ID 47:

GET `/api/v3/users/47`

```
{
        "id": 47,
        "name": "Student",
        "email": "s.tudent@st.hanze.nl",
        "roles": [
                "Student"
        ],
        "permissions": [
                "use studentendashboard"
        ]
}
```

Misschien is dit niet genoeg informatie en wil je ook kunnen zien aan welke groepen en projecten deze gebruiker gekoppeld is. Dan kan je natuurlijk twee requests doen naar **GET** `/api/v3/users/47/projects` en **GET** `/api/v3/users/47/groups`, maar je kan het ook in één request doen.

GET `/api/v3/users/47`

Request:

```
{
        "with": [
                "groups",
                "projects"
        ]
}
```

Response:

```
{
        "id": 47,
        "name": "Student",
        "email": "s.tudent@st.hanze.nl",
        "roles": [
                "Student"
        ],
        "permissions": [
                "use studentendashboard"
        ],
        "projects": [],
        "groups": [
                {
                        "id": 1,
                        "name": "Coachingsgroep Ronald (Checkin)",
                        "description": "Studenten die werken aan de check-in tool."
                }
        ]
}
```


De relaties die in de `with` array gezet kunnen worden, komen redelijkerwijs overeen met de relaties in *Entiteiten & Relaties*.

Er zijn ook een aantal extra opties die via de `with` array kunnen worden opgevraagd:

- **Groups**
  - `user_mood_avg` (Gemiddelde mood_score van alle gebruikers van een groep van de huidige week)
  - `hours_worked_today` (Totaal aantal uur dat alle gebruikers van een groep vandaag hebben gewerkt)
  - `hours_worked_yesterday` (Totaal aantal uur dat alle gebruikers van een groep gisteren hebben gewerkt)
- **Users**
  - `happiness` (Overzicht van de mood_scores van een gebruiker van het begin van dit jaar tot nu)
  - `hoursWorkedThisWeek` (Totaal aantal uur dat de gebruiker deze week heeft gewerkt)
  - `canDoDaily` (Of de gebruiker vandaag nog een daily checkin kan invullen)
  - `canDoWeekly` (Of de gebruiker deze week nog een weekly checkin kan invullen)

1.6.7 API responses

De API geeft voor elke client request een response terug. Hieronder staat aangegeven in welke situatie welke response van de API verwacht kan worden.

`HTTP 200` `HTTP 201` `HTTP 204`

Bij een succesvolle request zal een 200, 201 of 204 response terug worden gegeven.

`HTTP 404`

Als een bepaalde resource wordt opgevraagd die niet bestaat, zal een 404 response worden teruggegeven.

Note: Voorbeeld

`/users/1`, waarbij geen user met id 1 bestaat.

`/users/1/daily-checkins/1`, waarbij geen user **en/of** daily checkin met id 1 bestaat, **of** dat beide wel bestaan, maar de daily checkin bij een andere user hoort.

`/users/1/projects/1`, waarbij geen user **en/of** project met id 1 bestaat, **of** dat beide wel bestaan, maar het project niet aan de user is gekoppeld.

HTTP 403
Als de gebruiker een token meestuurt, maar de token incorrect is of de gebruiker onvoldoende rechten heeft, zal een 403 response terug worden gegeven.

HTTP 401
Als de gebruiker de API probeert te gebruiken zonder een token mee te sturen, kan een 401 response verwacht worden.

HTTP 400
Bij een API call in combinatie met een request body met onvoldoende of ongeldige parameters, zal een 400 response worden teruggestuurd.

1.6.8 Entiteiten & Relaties

Op dit moment geeft de API V3 toegang tot de volgende entiteiten en relaties.

- **users**
  - `belongsToMany` projects
  - `belongsToMany` groups
  - `hasMany` cubes
  - `hasMany` daily-checkins
  - `hasMany` weekly-checkins
  - `belongsToMany` jobroles
  - `hasMany` notifications
  - `hasMany` portfolio-elements
- **projects**
  - `belongsToMany` users
- **groups**
  - `belongsToMany` users
- **jobroles**
  - `belongsToMany` users
  - `hasMany` descriptions
- **cubes**
  - `hasMany` matrix-fields
- **matrix-field**
  - `belongsToMany` portfolio-elements
- **portfolio-elements**
- **activities**

- **architectures**
- **levels**

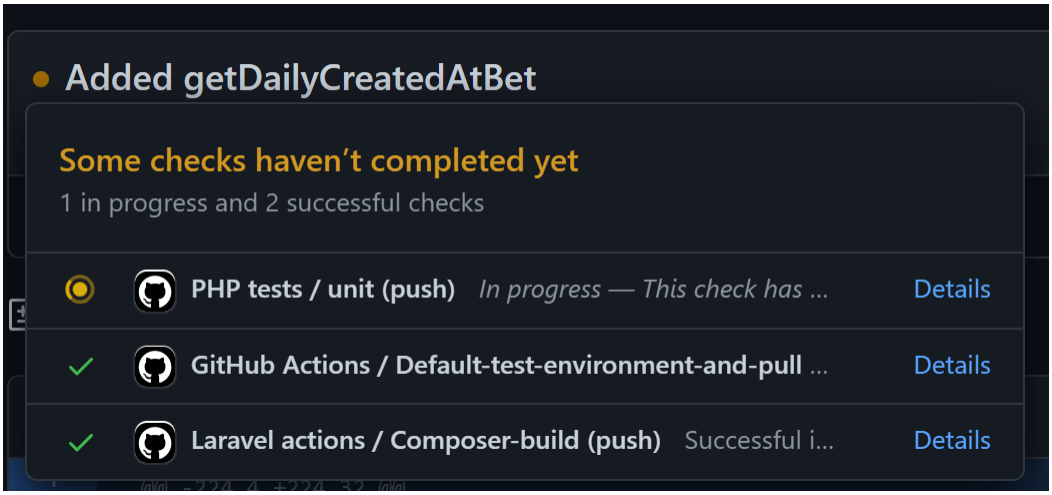
1.7 Tests

To ensure the application works, the team had written many checks that will automatically be tested during github merges, pulls and transfers. For an acurate view of the current tests, one should go to the "tests" folder in the github page.

Also, pulling up a random commit should show the tests being run and (hopefully) succesfully being completed.

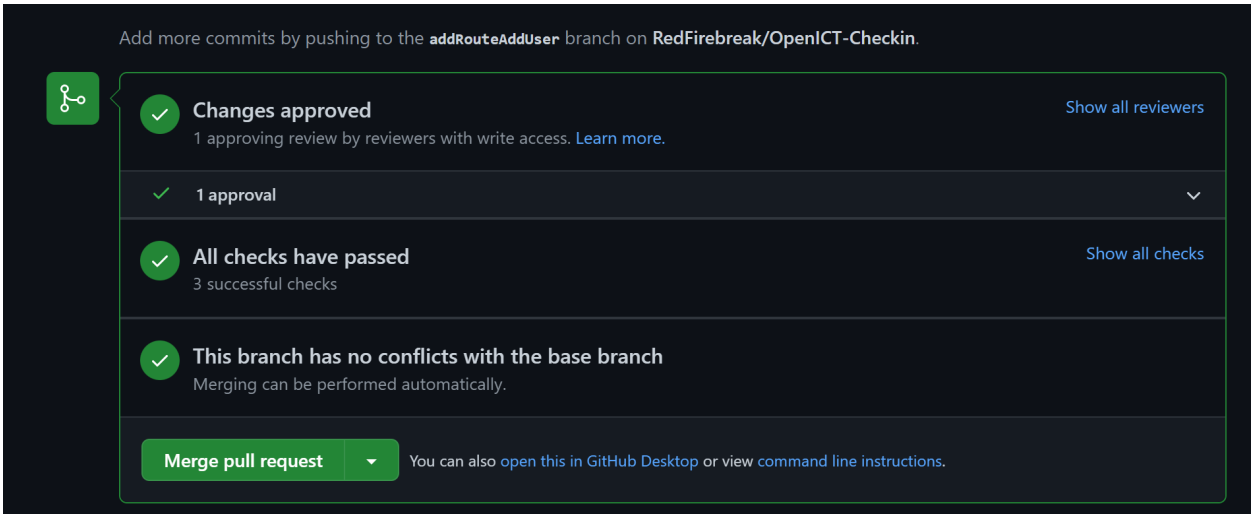
1.7.1 Running tests

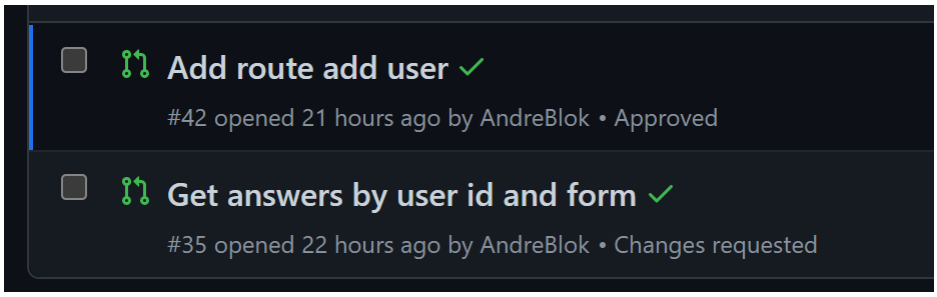
This is how the tests look when they are being run:



1.7.2 Passed tests

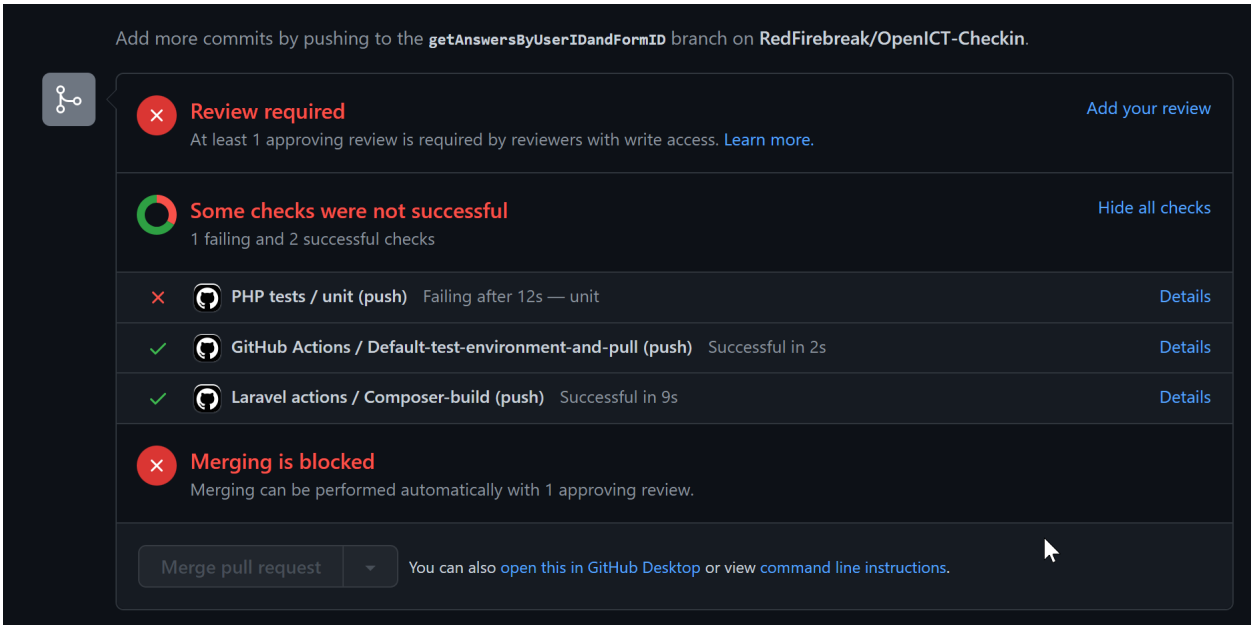
This is how the tests look when they have been succesfully completed:





1.7.3 Failed tests

This is how the tests look when they have been succesfully completed:



1.8 Authentication

The user authentication is done by Laravel. As of writing, the laravel package used is laravel:breeze.

/login

/register

/logout of Auth::logout()

The Authorization bearer token is automatically added when the frontend makes an api call. In case the token hasn't been set, it creates and adds a token generated using an md5 hash of 120 random bytes.

1.9 Release notes

Release notes :

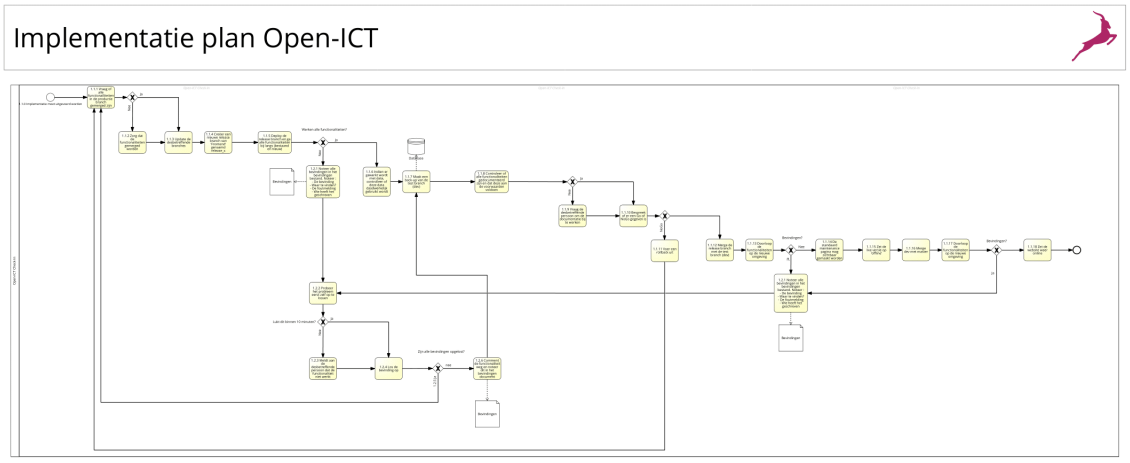
Release-22.6.10:

- **Design changes :**
  - Smileys
  - Dashboard indeling
  - Database
  - Foutmelding pagina's
  - Wachtwoord resetten
- **Bugfixes :**
  - Lege jobroles toevoegen zonder foutmelding
  - Mail-server gefixed (werd niet meer ondersteund door google/werkt alleen met gmail)
- **Theorie :**
  - Implementatie plan geschreven
  - Documentatie bijgewerkt
  - BPMN model gemaakt
- **QoL :**
  - Opmerkingen veld alleen zichtbaar bij ja
  - Check-box laatste vraag niet geselecteerd
  - Radio button voor docent of student keuze tijdens het aanmaken van een gebruiker

1.10 Deliverables

1.10.1 Implementatie plan

Hier kan je een image vinden voor het implementatie plan:



Implementatie plan Alle stappen moeten succesvol uitgevoerd worden. Wanneer dit niet het geval is of wanneer er bevindingen zijn gemaakt, volg dan stappenplan 1.2.

Stappenplan 1.1.0.

- Stap 1.1.1: Vraag of alle functionaliteiten in de productie branch gemerged is.
- Stap 1.1.2: Indien niet alle functionaliteiten toegevoegd zijn, zorg ervoor dat i-edereen deze toevoegd.
- Stap 1.1.3: Update de desbetreffende branch zodat alle nieuwe functionaliteiten mee genomen zijn.
- Stap 1.1.4: Maak een nieuwe release branch aan van de branch 'Frontend' genaamd 'release_X' met X als de volgende versienr.
- **Stap 1.1.5: Checkout de release branch en ga elke functionaliteit (bestaande en nieuwe) bij langs op een lokale omgeving.**
  - Bevindingen gevonden? Volg Stappen plan 1.2 vanaf hier.
- Stap 1.1.6: In de code, als er met data gewerkt wordt moet er gecontrolleerd worden of die data ook daadwerklijk gebruikt en doorgegeven wordt.
- **Stap 1.1.7: Als de bovenstaande stappen succesvol uitgevoerd zijn.**
  - backup van huidige productie
  - Vervolgens kan de live versie 'offline' gehaald worden. De standaard 'Maintenance' pagina zal zichtbaar gemaakt worden voor de gebruikers. Alleen de admin/beheerder kan nog volledig gebruik maken van de website.

- Stap 1.1.8: Vervolgens moet er gecontrolleerd worden of elke functionalitiet gedocumenteerd is en dat dit aan de voorwaarden voldoet. Dit heet: Limited Production Run -> maak hier een haandboekje van op basis van de release notes. Bedenk even welke oude functionaliteiten je wil testen.
- Stap 1.1.9 Indien de documentatie niet compleet is, vraag de desbetreffende persoon of hij of zij deze kan bijwerken.
- Stap 1.1.10: Bespreek met de product owners of er een Go of NoGo is voor het uitrollen van de nieuwe release.
- Stap 1.1.11: Is er een NoGo gegeven? Rollback de voorgaande werkzaamheden en doorloop het implementatie plan opnieuw.
- Stap 1.1.12: Merge de release branche naar de 'dev' branch.
- **Stap 1.1.13 Doorloop de website op de online dev omgeving en noteer de bevindingen, indien nodig doorloop ook stap 1.1.6.**
  - Bevindingen gevonden? Volg Stappen plan 1.2 vanaf hier.
- Stap 1.1.14: Indien de voorgaande stappen succesvol doorgelopen zijn, kan de standaard 'maintenance' pagina zichtbaar gemaakt worden op de live versie.
- Stap 1.1.15: Zet de live versie op 'offline'
- Stap 1.1.16: Indien stap 7 succesvol verloopt zonder problemen kan de 'dev' branch gemerged worden met de 'master' branch.
- **Stap 1.1.17: Doorloop de website op de live omgeving en noteer de bevindingen.**
  - Bevindingen gevonden? Volg Stappen plan 1.2 vanaf hier.
- Stap 1.1.18: Indien stap 10 succesvol is dan kan de live website weer op 'online' gezet worden en mag deze weer volledig gebruikt worden.

Stappenplan 1.2

- **Stap 1.2.1: Noteer de bevindingen in een apart bestand. Hierbij noteer je het volgende:**
  - De bevinding
  - Waar is dit probleem te vinden
  - De foutmelding zelf
  - Wie heeft deze functionaliteit geschreven
- Stap 1.2.2: Probeer eerst zelf 5 tot 10 minuten het probleem op te lossen.
- Stap 1.2.3: Mocht stap 1.2.2 niet lukken meldt dit bij de desbetreffende persoon die het stukje functionaliteit gemaakt heeft en geef aan wat er fout is en vraag of hij of zij dit z.s.m. kan oplossen.
- Stap 1.2.4: Los de bevinding op.
- Stap 1.2.5: Indien er meerdere bevindingen zijn, moet er gewacht worden tot elk van deze bevinding opgelost zijn voordat Stappenplan 1.1 opnieuw gedaan wordt.

Als stap 1.2.4 voltooid is kan Stappenplan 1.1 weer doorlopen worden. - Stap 1.2.6: Mocht een bevinding niet snel genoeg opgelost worden voor een oplevering dan zal deze functionaliteit niet aanwezig mogen zijn op de live omgeving. Dit moet 'uit gecomment' worden. Noteer dit wel in het bevindingen formulier. Denk hierbij wel aan het noteren van :

- De functionaliteit
- Waar is deze te vinden
- Wat is er gecomment

1.11 Bevindingen

1.11.1 Bevinding 1:

- Probleem: (Vermoedelijk) Spamfilter van de Hanze filtert automatisch berichten van een onbekende bron uit, waardoor mails niet binnenkomen, zelfs niet in de ongewenste mail.
- Oplossing/Kanttekening: Het versturen van mails naar een omgeving buiten Outlook om, zoals gmail is wel mogelijk. De huidige instellingen van de mailserver is de vinden in het .env bestandje in het project. Vermoedelijk moet er contact worden opgenomen met de Hanze over het whitelisten van onze applicatie, zodat er ook mailtjes verstuurd kunnen worden naar schoolmail adressen

1.11.2 Bevinding 2:

- Probleem: Er worden te veel API calls gemaakt op het docentendashboard, waardoor als jij de pagina laadt hij soms tot vaak een foutmelding geeft 'Too many requests'. Dit is voornamelijk op de lokale omgeving.
- Oplossing: In de toekomst zou kunnen gekeken worden naar de structuur van API calls, zodat deze beperkt worden. Eventueel kan ook gekeken worden naar de api rate limitter / throttle.

1.11.3 Bevinding 3:

- Probleem: De applicatie is nog niet geheel responsive momenteel, als voorbeeld het docentendashboard wordt incorrect ingeladen op de mobiel waardoor het een onprofessionele look geeft.
- Oplossing: De formatting en css moet aangepast worden zodat dit op alle/meeste apparaten netjes en correct weergeven wordt.