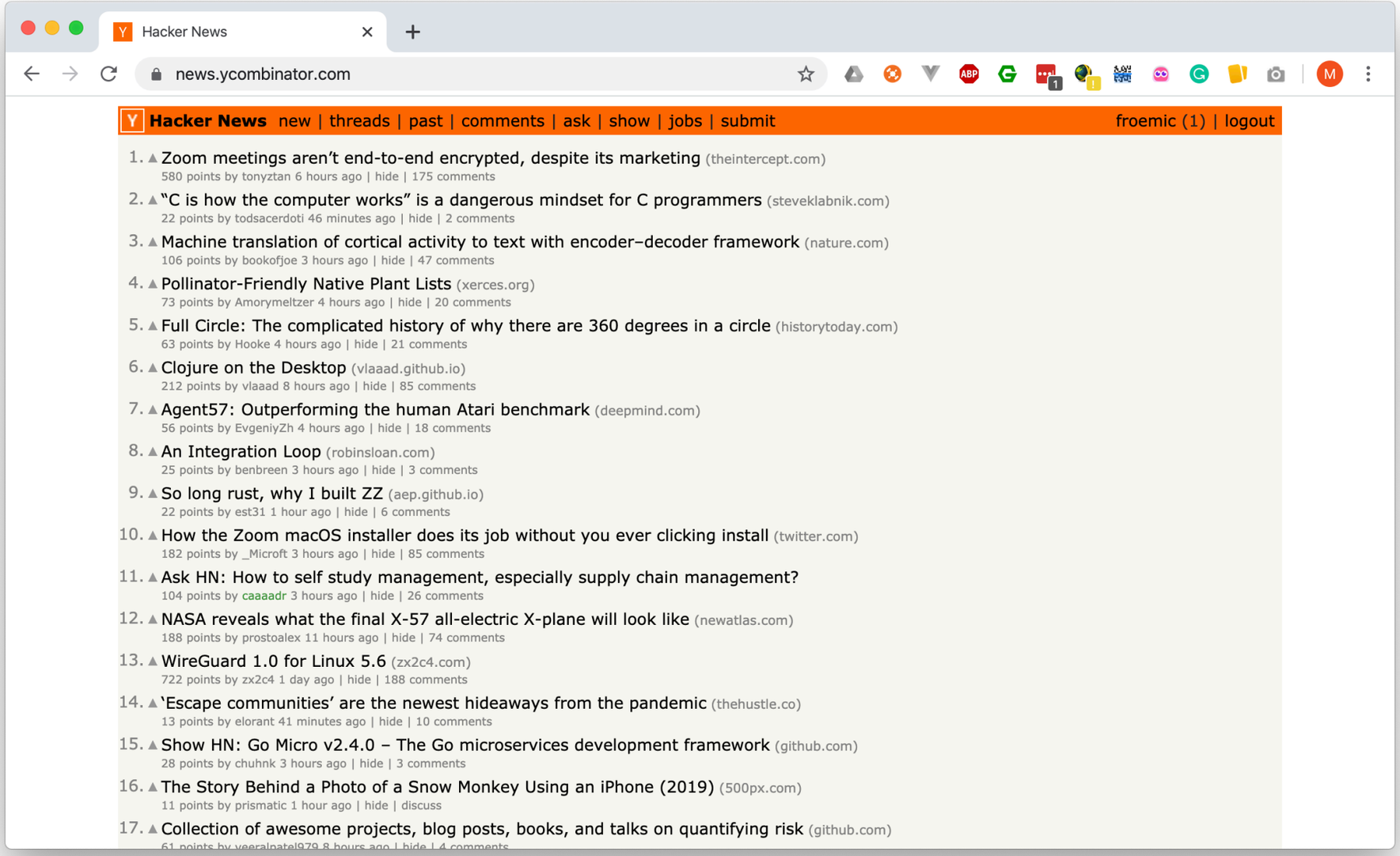
WEB DEVELOPMENT

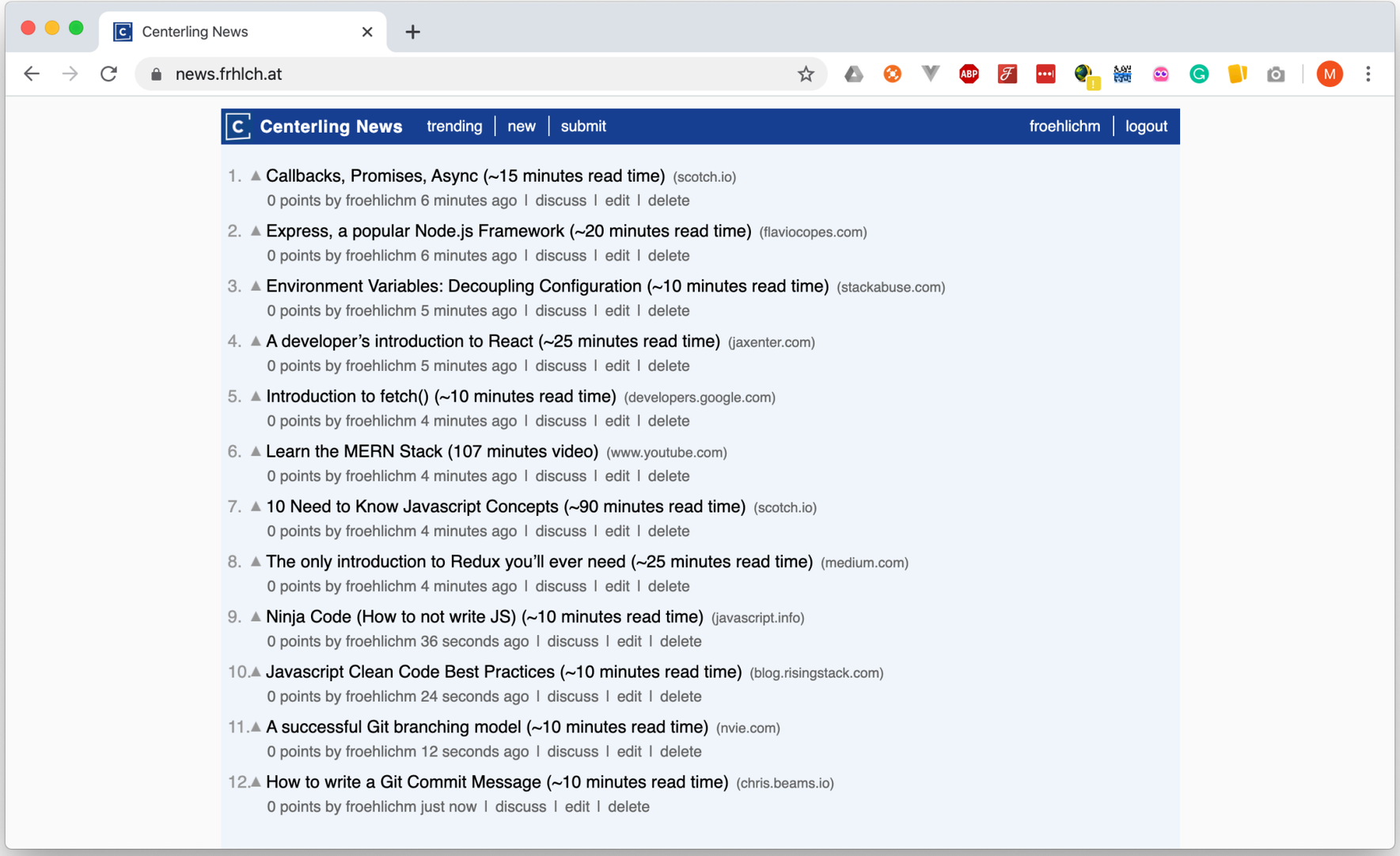
ASSIGNMENT II

ASSIGNMENT II – HACKER NEWS!



https://news.ycombinator.com/

ASSIGNMENT II – CENTERLING NEWS



https://news.frhlch.at/

ASSIGNMENT II (1/2)

👉 You will develop your first web application, a clone of Y-Combinators popular site Hacker News.

INSTRUCTIONS

(1) Complete the following readings (mandatory)

Please take the time to read the following articles, before you jump into development. They will help you to mitigate some of the beginner errors you would otherwise run into.

- Callbacks, Promises, Async(~15 minutes read time) https://scotch.io/courses/10-need-to-know-javascript-concepts/callbacks-promises-and-async
- Express, a popular Node.js Framework (~20 minutes read time) https://flaviocopes.com/express/
- Environment Variables: Decoupling Configuration (~10 minutes read time) https://stackabuse.com/managing-environment-variables-in-node-js-with-dotenv/
- A developer's introduction to React (~25 minutes read time) https://jaxenter.com/introduction-react-147054.html
- Introduction to fetch() (~10 minutes read time) https://developers.google.com/web/updates/2015/03/introduction-to-fetch
  Learn the MERN Stack (107 minutes video) [1] https://www.youtube.com/watch?v=7CqJlxBYj-M

Optional Readings

These readings are optional but will help you understand advanced concepts (such as React-Redux) and *make you a better developer* (really). If you are struggling with Javascript, I would recommend you, to read through the first article.

- 10 Need to Know Javascript Concepts (~90 minutes read time) https://scotch.io/courses/10-need-to-know-javascript-concepts
- The only introduction to Redux you'll ever need (~25 minutes read time) https://medium.com/javascript-in-plain-english/the-only-introduction-to-redux-and-react-redux-youll-ever-need-8ce5da9e53c6
- Ninja Code (How to not write JS) (~10 minutes read time) https://javascript.info/ninja-code
- Javascript Clean Code Best Practices (~10 minutes read time) https://blog.risingstack.com/javascript-clean-coding-best-practices-node-js-at-scale/
- A successful Git branching model (~10 minutes read time) https://nvie.com/posts/a-successful-git-branching-model/
- How to write a Git Commit Message (~10 minutes read time) https://chris.beams.io/posts/git-commit/

1) You don't need to code along this time. However, this video has all the core concepts you will need to implement Assignment II. Watch it and transfer the things he is doing to your user stories.

ASSIGNMENT II (2/2)

👉 You will develop your first web application, a clone of Y-Combinators popular site Hacker News.

INSTRUCTIONS

(2) Implement the "Centerling News" application in the provided repository

- Fork the prepared repository: https://gitlab.com/CDTM/courses/2020-1-ele-web-development/assignment-2
- Implement the open user stories in your forked repository
- Make sure to follow Git Flow
  - For each user story create a feature branch from develop
  - Once implemented merge the feature branch back into develop using a merge request, in which you summarize the changes you made
  - Once all user stories are developed, create a merge request from develop into master and assign your coach as reviewer (latest 27th April)

(3) Decide on an idea for a final project together with another student

- Come up with a project idea with another student from this class.
- Your project should include CRUD functionality and require implementation across React, Express and MongoDB.
- You should have a partner and idea by the next session, since we will develop the idea in a workshop format.

(4) Fill out the survey: https://forms.gle/bay5hHKDii1uwhTG9

HELPFUL RESOURCES

Google and Stackoverflow will be your biggest friends during this exercise. If you encounter a problem, google it. Aside from that here are a few helpful links that should cater for ~80 % of the issues you will be facing.

- https://www.w3schools.com/
- https://reactjs.org/docs/getting-started.html
- https://reactjs.org/docs/thinking-in-react.html
- https://mongoosejs.com/docs/#
- https://expressjs.com/
- https://sass-lang.com/guide

GRADING

Functional Completeness:	70 %
Code Quality:	15 %
Git Usage:	15 %

DUE DATES

April 12th 23:59: complete all mandatory readings
April 27th 23:59: implement all user stories and assign your coach as reviewer
April 27th 23:59: form a team for the final project and fill out the survey
May 4th 17:00: bring your project idea to class

ASSIGNMENT II: COACHING CONCEPT

👉 For this assignment all of you will be supported by an experienced coach.

Ground Rules

- All coaches agreed to one 30 minutes coaching slot per student during the next weeks. They might contribute more time, but this is up to them.
- After you hand in your assignment the coaches will provide a code review, so you also get feedback on your code style and quality (This is why you have to add them as reviewer in your final merge request).
- Coaches are not there to write code for you – it is you who should do the programming.

How to take the most out of the coaching?

- Schedule your coaching session after you completed the readings and after you have started to implement the first user stories.
- Use the Slack channel to ask the group, if you get stuck. Helping each other will benefit your learning process.

ASSIGNMENT II: COACHING ALLOCATIONS

👉 Contact your assigned coach today via email and introduce yourself. You should schedule a coaching session before April 27th.

COACH	STUDENT	
Marcello Schreiber (Fall 2019) marcello.schreiber@cdtm.de marcellosch	Esteban Vindas Prado	esteban.prado@cdtm.de
	Lucas Grabmaier	lucas.grabmaier@cdtm.de
Simon Zachau (Fall 2019) simon.zachau@cdtm.de ytfrcyuli	Valentin Kellner	valentin.kellner@cdtm.de
	Andreas Felderer	andreas.felderer@cdtm.de
Felix Eckert (Fall 2019) felix.eckert@cdtm.de felixeckerttum	Tim Engelmann	tim.engelmann@cdtm.de
	Borja Clemente	borja.sanchez@cdtm.de
Mohammad Alamleh (Fall 2019) mohammad.alamleh@cdtm.de malamleh93	Max Knicker	max.knicker@cdtm.de
	Till Wiechmann	till.wiechmann@cdtm.de
Nico Bentenrieder (Fall 2017) nico.bentenrieder@cdtm.de bentocin	Luis Onuma Okamoto	luis.onuma@cdtm.de
	Sabrina Mohr	sabrina.mohr@cdtm.de
Johannes Stanggassinger (Spring 2016) johannes.stanggassinger@cdtm.de -	Jannik Wiedenhaupt	jannik.wiedenhaupt@cdtm.de
	Omar Dahroug	omar.dahroug@cdtm.de
Sebastian Schlecht (Fall 2014) sebastian.schlecht@cdtm.de sebastian-schlecht	Amadea Pély	amadea.pely@cdtm.de
	José Vega	jose.vega@cdtm.de

(VOLUNTARY) HELP USE IMPROVE THIS ELECTIVE

👉 How much time does it take to complete this elective? We would like to know to ensure a fair workload. For this we would like to ask you to track the time it takes you to complete it. How long it takes you to finish any assignment will not affect grading in any way.

https://toggl.com/

https://clockify.me/

USER STORIES

To implement Centerling News we have split the project into 12 user stories. The first 6 user stories are already implemented. Your task is to implement the remaining 6.

- The *Definition of Done* for each user story defines the functionality that has to be implemented for it to be complete.
- The final site does not have to resemble the style of Hacker News. Feel free to add your personal touch, if you want.
- Please avoid using component frameworks for React. Build your components with jsx and css.

01 REGISTER A NEW ACCOUNT

As a ‹user›, I'd like to ‹create a new account›, in order to ‹create, upvote and comment on interesting posts›.

Definition of Done

- A user can create a new account with my email, username, firstname, lastname and password.
- Account registration fails, if not all of the parameters specified above are entered.
- Account registration fails, if the user is already registered.
- If the account registration fails, the user is presented with a descriptive error message.
- After successful account creation the user data is persisted in the database.

02 VERIFY MY ACCOUNT

As a ‹user›, I'd like to ‹verify my email address›, in order to ‹get access to my account›.

Definition of Done

- A user is sent a verification code to the email address he registered with.
- A user can enter this verification code in order to verify the ownership of this email address.
- If the verification fails, the user is presented with a descriptive error message.
- Accounts that are not verified cannot log in.
- The verification status is persisted permanently.
- Users can request a new verification code.

03 SIGN INTO MY ACCOUNT

As a ‹user›, I'd like to ‹sign in my account›, in order to ‹create, upvote and comment on interesting posts›.

Definition of Done

- A user can sign in their account with their email and password.
- The sign in process fails, if the email-password combination is not known or not correct.
- If the sign in process fails, the user is presented with a descriptive error message.
- If the sign in process succeeds, the user's name is shown as long as they are logged in.
- If the sign in process succeeds, the user does not need to sign in for 24 hours.

04 SIGN OUT OF MY ACCOUNT

As a ‹user›, I'd like to ‹sign out of my account›, in order to ‹avoid other users creating, upvoting or commenting posts on my behalf›.

Definition of Done

- A user can sign out of their account by pressing a "Logout" button.
- Logging out of an account triggers a logout on all devices where the user was logged in.

05 EDIT MY ACCOUNT

As a ‹user›, I'd like to ‹edit my account data›, in order to ‹keep it up to date and secure›.

Definition of Done

- A user can edit their Firstname, Lastname and password.
- Updating the password requires to enter the correct current password.
- Updating the account fails, if either of the submitted fields contains invalid data.

06 RESET PASSWORD

As a ‹user›, I'd like to ‹reset my password›, in order to ‹regain access after I forgot my current password›.

Definition of Done

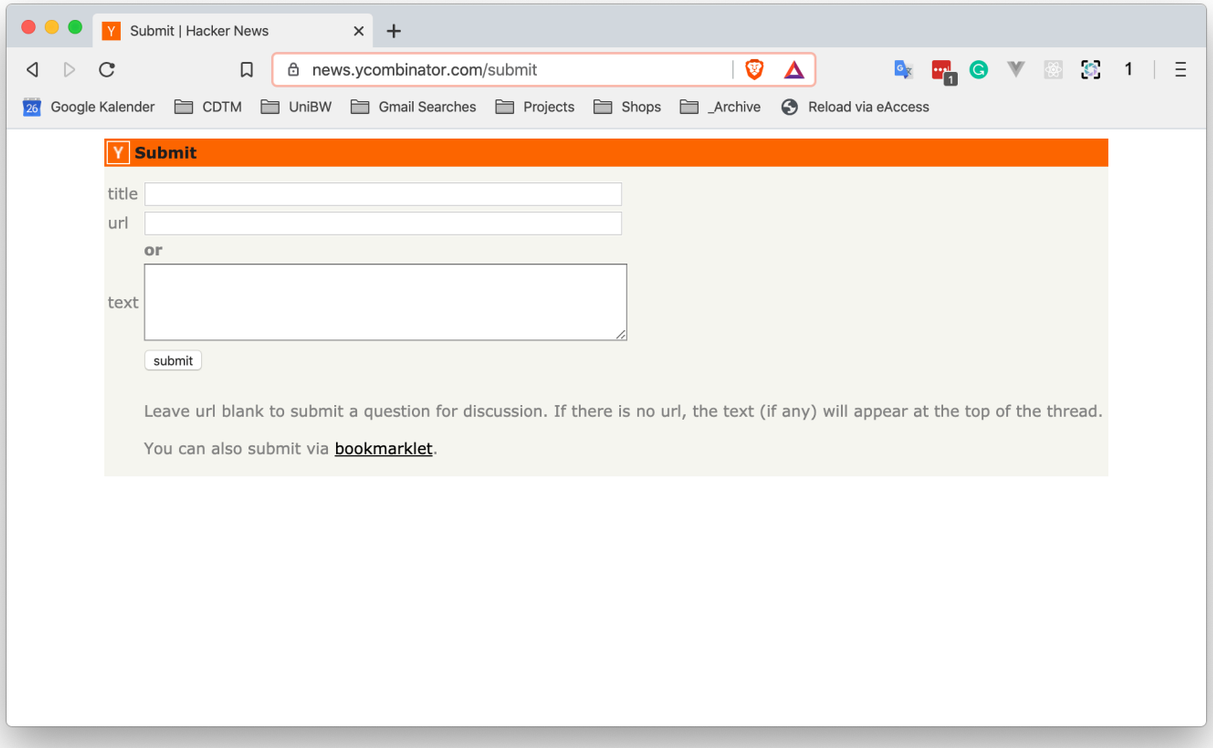
- A user can request a password reset link.
- A user who requested a reset link will be able to choose a new password for their account.

07 SUBMIT A NEW POST

As an ‹author›, I'd like to ‹create a new post›, in order to ‹share interesting computer science and entrepreneurship resources and stories with the world›.

Definition of Done

- An author can create a new post with a title and an url.
- The author must be signed in to create a new post.
- The post creation fails, if the title is longer than 100 characters.
- The post creation fails, if the url is longer than 500 characters.
- The post creation routes adhere to RESTful principles.
- If the post creation fails, the author is presented with a descriptive error message.
- If the post creation succeeds, the post is permanently persisted in the database.



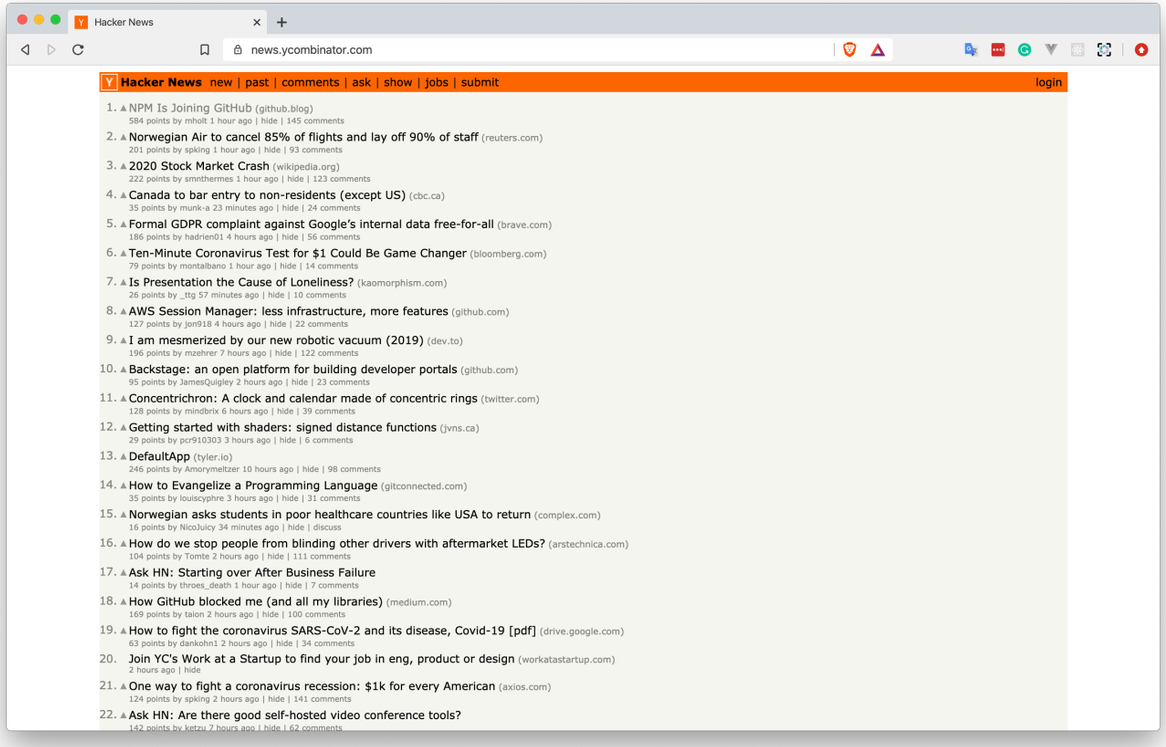
User Stories

08 VIEW ALL POSTS

As a ‹user›, I'd like to ‹see all posts›, in order to ‹discover interesting computer science and entrepreneurship resources and stories with the world›.

Definition of Done

- A user can see all posts on the site.
- For each post, the title and the name of the author a clearly readable.
- Posts are ordered by the time they were created on. The newest posts are shown on the top.
- Clicking on the title of a post opens a new tab with the url the post it pointing to.

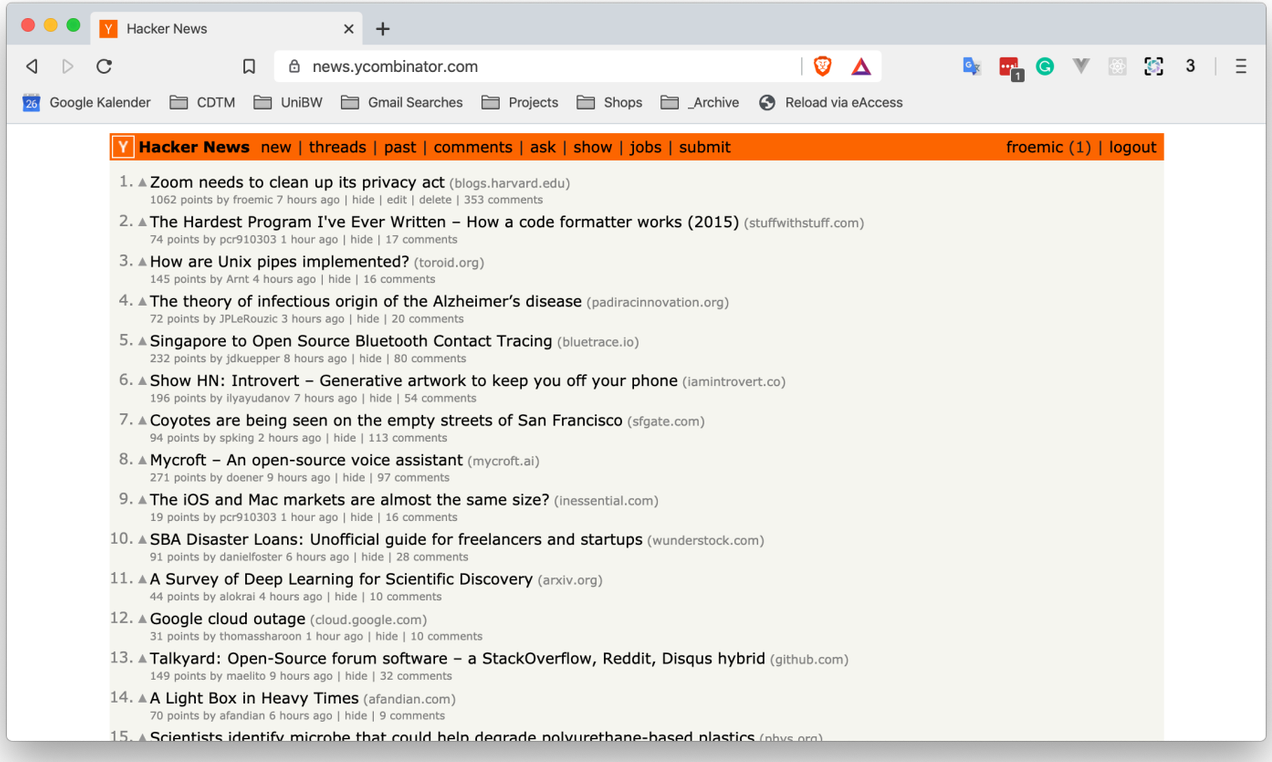


09 DELETE MY OWN POST

As a ‹author›, I'd like to ‹delete a post I created earlier›, in order to ‹remove uninteresting content›.

Definition of Done

- An author can remove a post they created earlier.
- The author must be signed in to delete on of their posts.
- Authors of a post can see a delete button or icon next to their own posts in the list.
- After clicking the delete button authors have to confirm their decision.
- After deleting a post, it is automatically removed from the user interface.
- After deleting a post, it is permanently removed from the database.
- The post deletion routes adhere to RESTful principles.

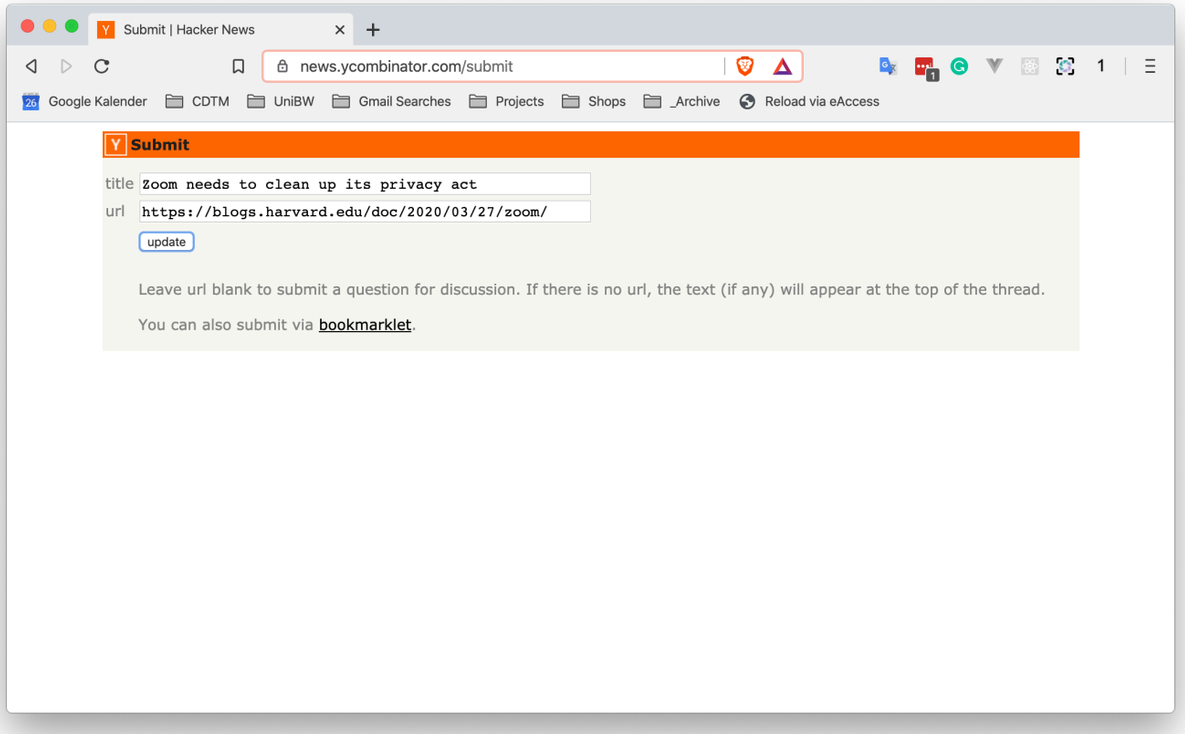


10 EDIT MY OWN POST

As a ‹author›, I'd like to ‹change a post I created earlier›, in order to ‹remove mistakes I had made.›.

Definition of Done

- An author can edit a post they created earlier.
- The author must be signed in to edit one of their posts.
- Authors of a post can see an edit button or icon next to their own posts in the list.
- After editing a post, it is automatically removed from the user interface.
- After editing a post, it is permanently updated in the database.
- The post deletion routes adhere to RESTful principles.



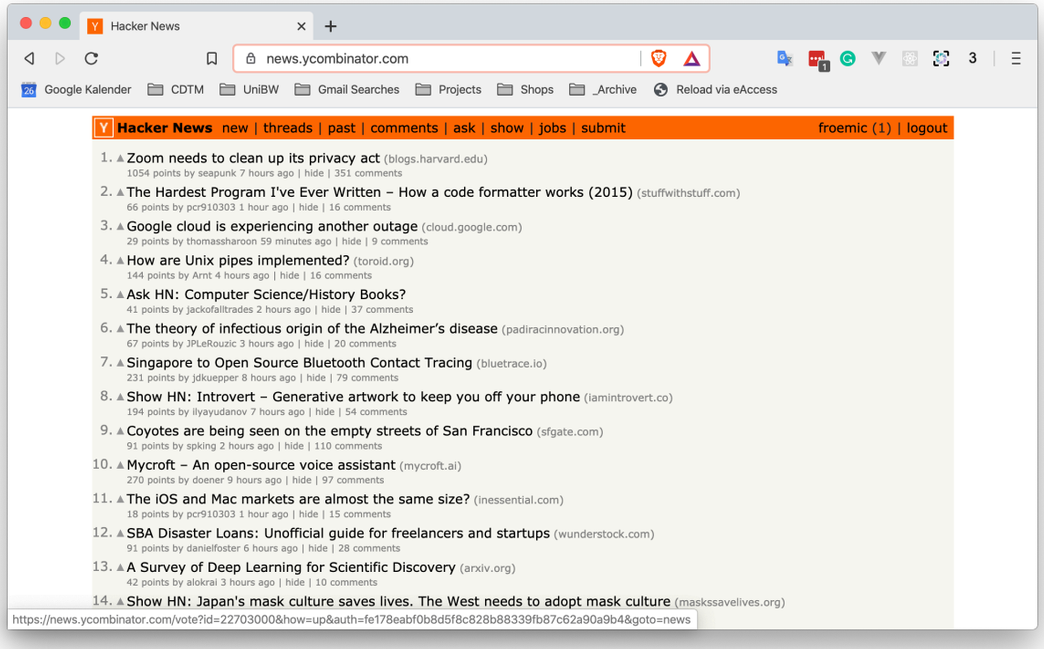
User Stories

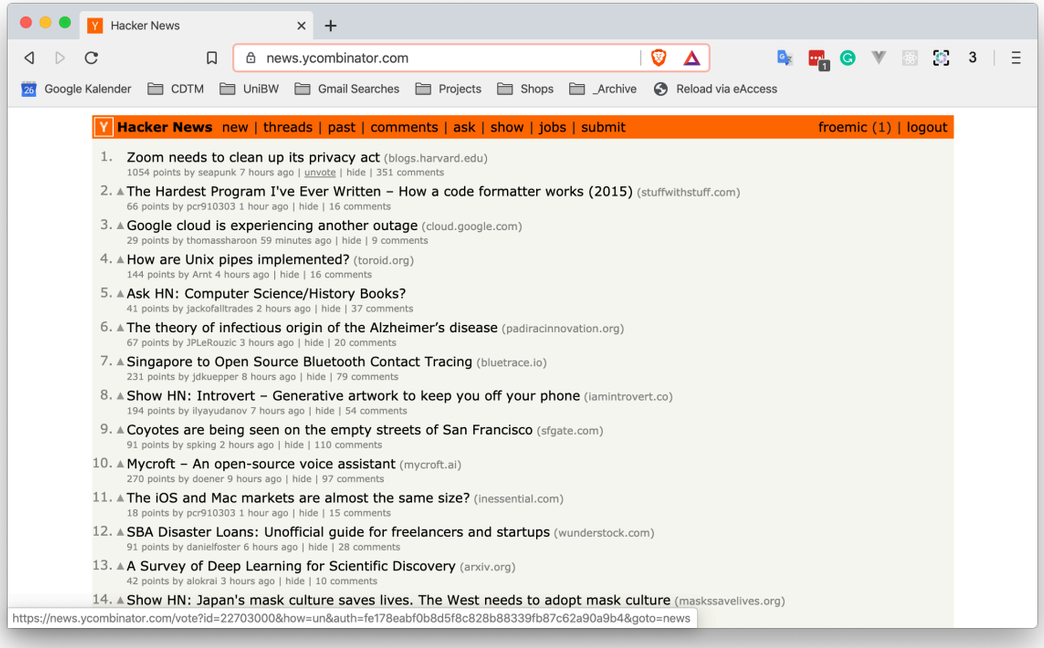
11 UPVOTE/ UNVOTE A POST

As a ‹user›, I'd like to ‹upvote a post›, in order to ‹show the community that I think a post is interesting›.

Definition of Done

- A user can upvote a post.
- Each post in the list has an upvote button.
- Each post shows the number of upvotes it has received.
- The user must be signed in to upvote any post.
- If a user presses the upvote button without being signed in, they are redirected to the login page.
- Each user can only upvote each post once.
- If a user has upvoted a post, the upvote button for this specific post turns into a remove-upvote button.
- After clicking the upvote button, the upvote is persisted in the database.
- After clicking the remove-upvote button, the upvote is permanently removed from the database.
- The list of posts is ordered by the ratio between the number of upvotes and the hours since its creation. The highest scoring post is on the top.
- The post deletion routes adhere to RESTful principles.



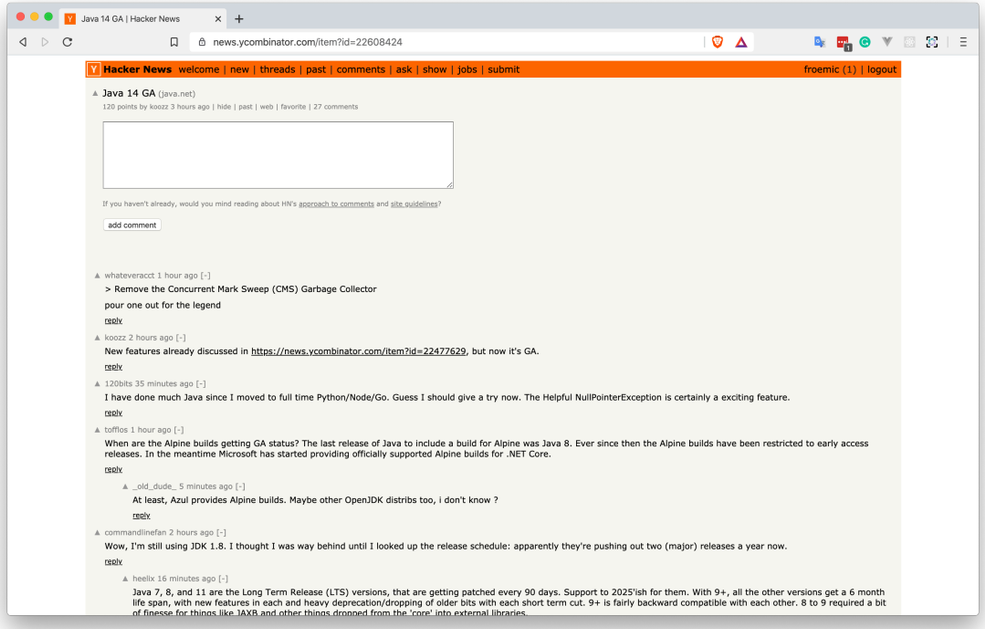


12 ADD A COMMENT TO A POST

As a ‹user›, I'd like to ‹upvote a post›, in order to ‹show the community that I think a post is interesting›.

Definition of Done

- Each post can have many comments.
- Each post shows the number of comments it received.
- Each post has a detail page, which shows all the associated comments, their author and time of creation.
- Comments are ordered by their creation date. The newest comment is on the top.
- Signed in users can add comments on the detail page.
- The user must be signed in to comment on a post.
- After creating a comment, the comment is persisted in the database.
- After creating a comment, the comment is added to the user interface.



INITIAL SETUP

This guide walks you through the initial setup of the repository. You will only have to do this once, but it is important that you closely follow this tutorial.

01 GITLAB SETUP

02 CLONE THE REPOSITORY

03 REGISTER AND SET UP MONGODB ATLAS

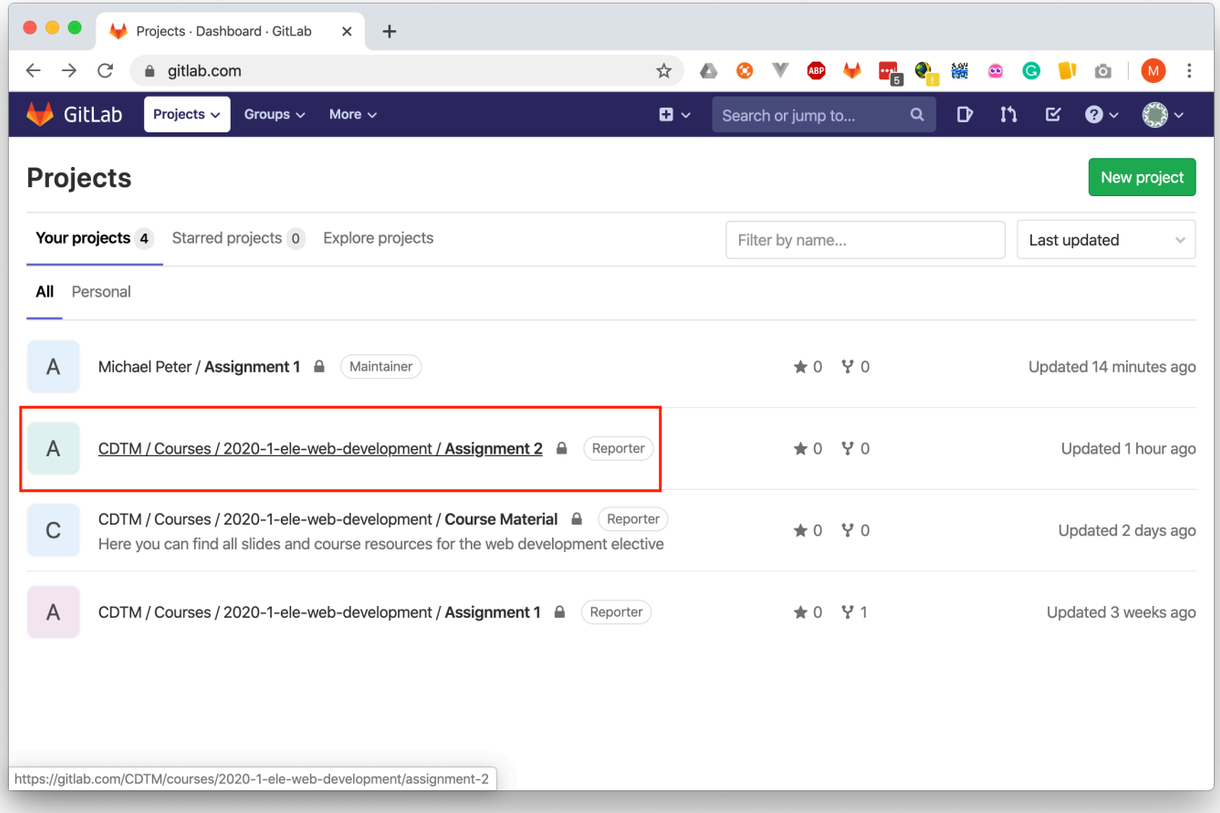
04 SET UP A SERVER

05 SET UP THE REACT CLIENT

1. GITLAB SETUP

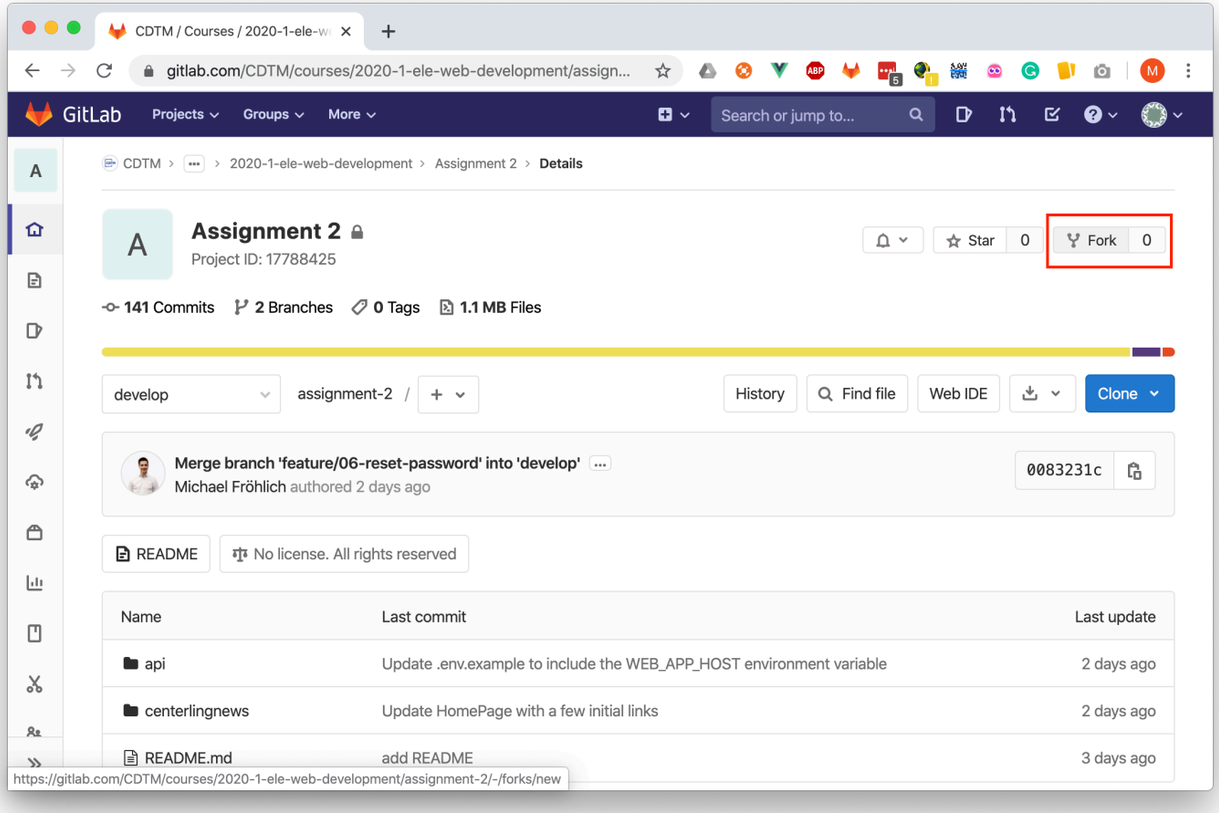
FORK THE REPOSITORY (1/2)

1. Go to https://gitlab.com
2. Select the CDTM/Courses/2020-1-ele-web-development/Assignment 2 repository
3. Click on **Fork**
4. Select your personal namespace



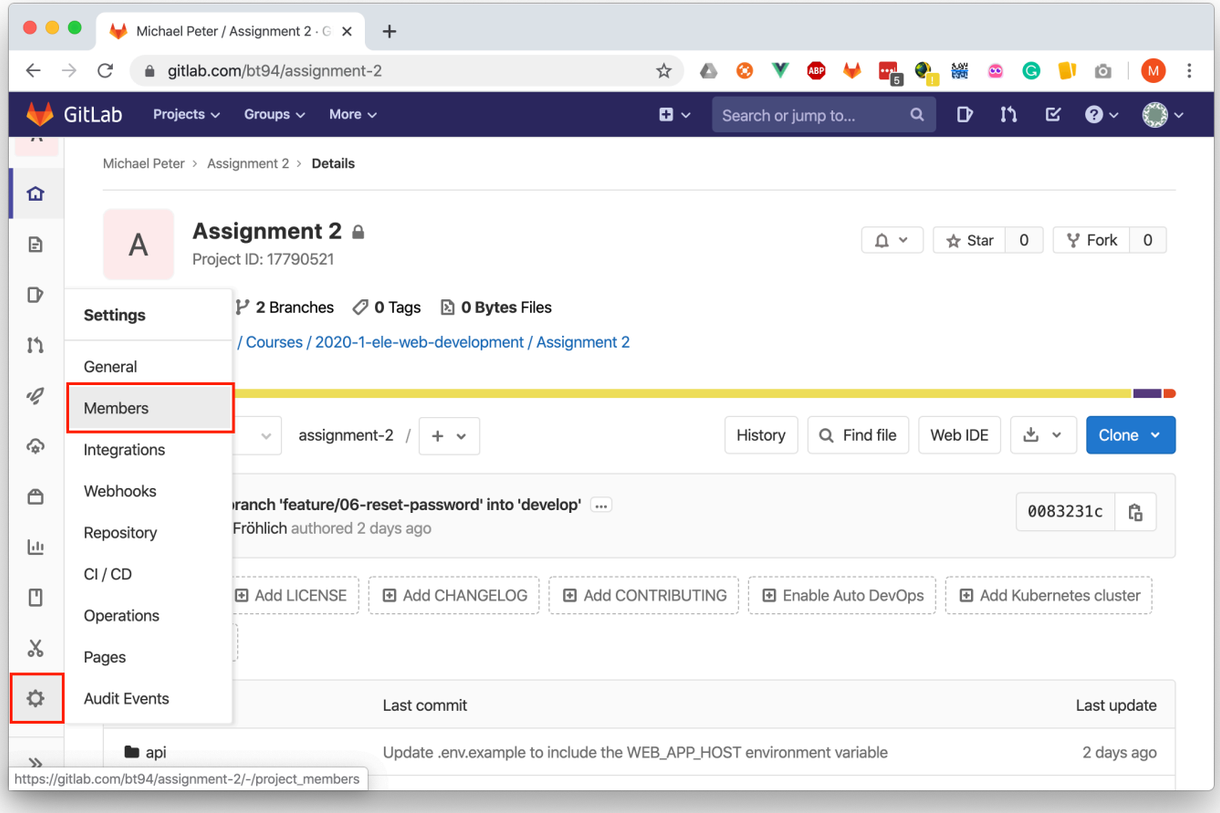
FORK THE REPOSITORY (2/2)

1. Go to https://gitlab.com
2. Select the CDTM/Courses/2020-1-ele-web-development/Assignment 2 repository
3. Click on **Fork**
4. Select your personal namespace



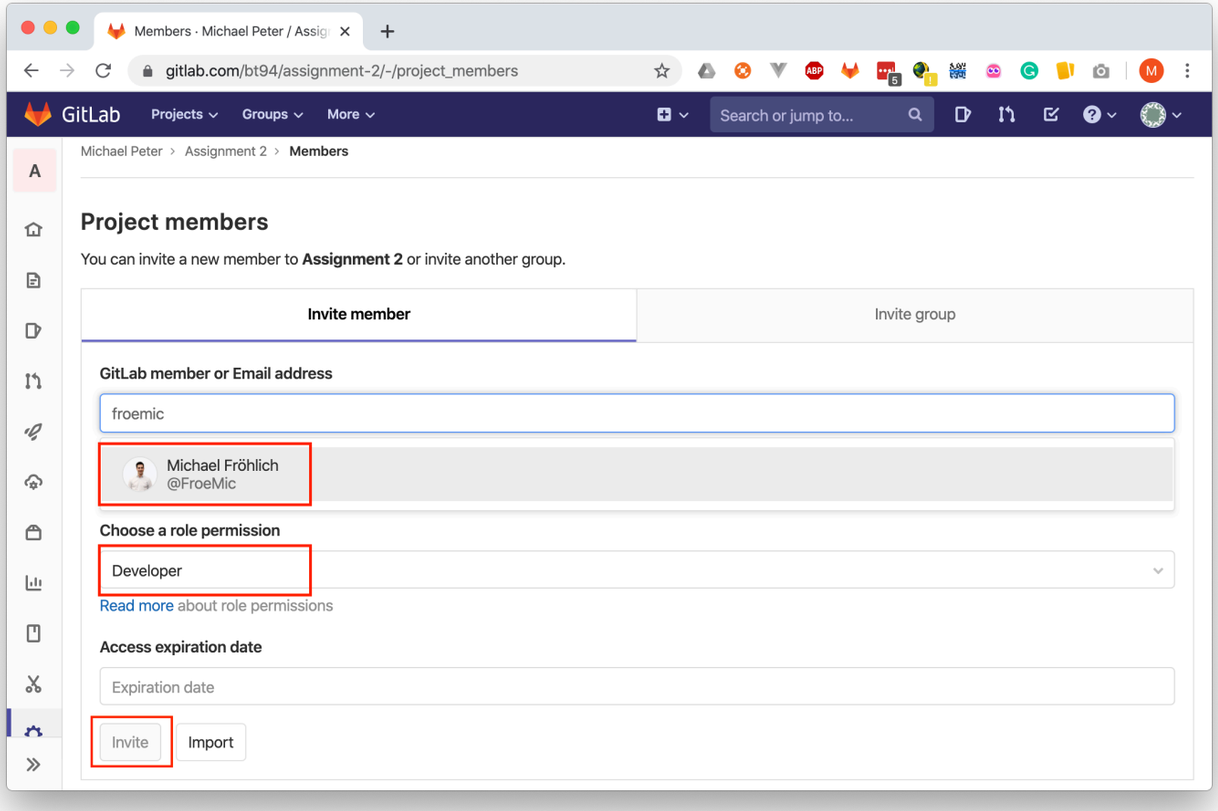
Add me and your coach to your repository (1/2)

1. In the side bar on the left select **Settings** and then **Members**
2. Add **froemic** as Developer to the repository
3. Add your coach to the repository as well. (If you don't know their username, write them an email right away)
4. Click **Invite**
5. **Important:** In the thread we have opened in the slack channel for exactly this step, post your Gitlab username to signal that you have finished this step. I can then run a script that will copy the User Stories into your project's issues section.



Add me (and your coach) to your repository (2/2)

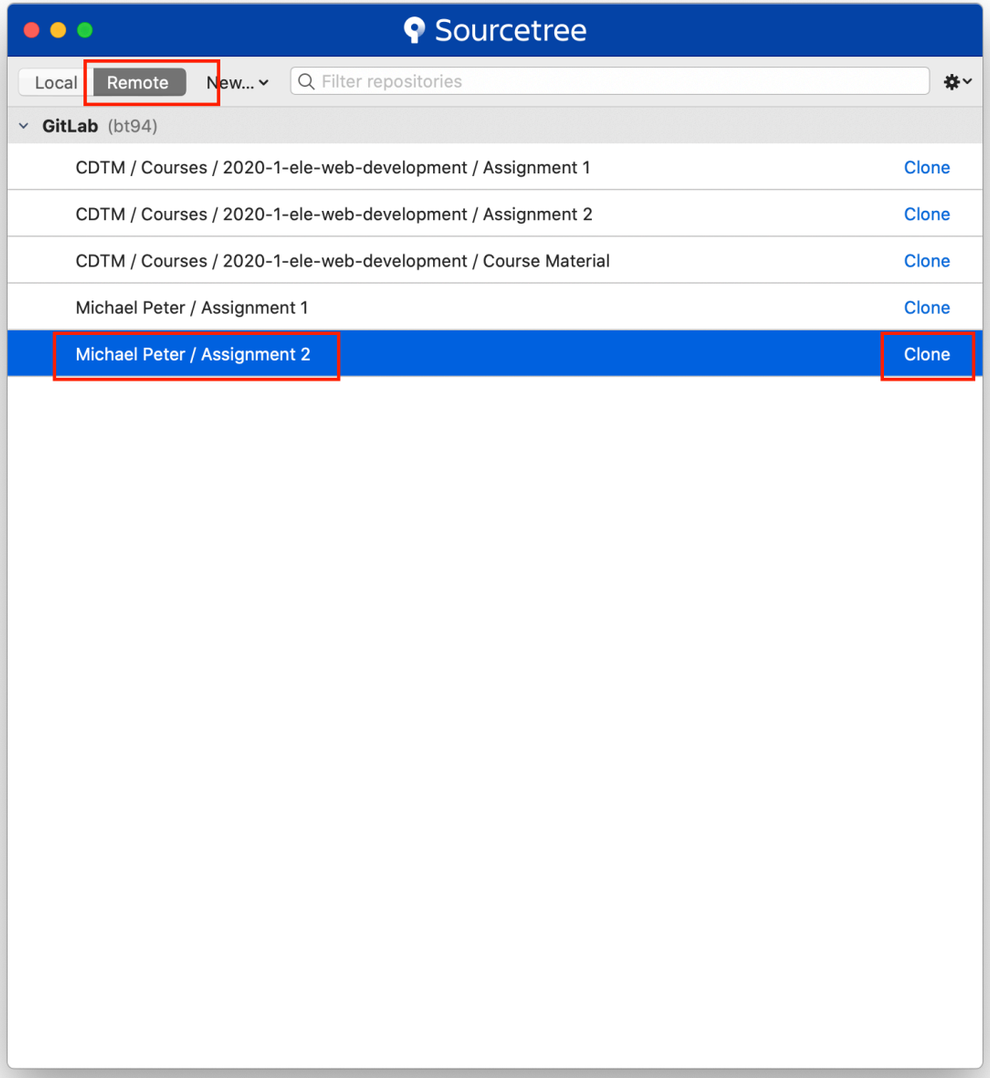
1. In the side bar on the left select **Settings** and then **Members**
2. Add **froemic** as Developer to the repository
3. Add your coach to the repository as well. (If you don't know their username, write them an email right away)
4. Click **Invite**
5. **Important:** In the thread we have opened in the slack channel for exactly this step, post your Gitlab username to signal that you have finished this step. I can then run a script that will copy the User Stories into your project's issues section.



2. CLONE THE REPOSITORY

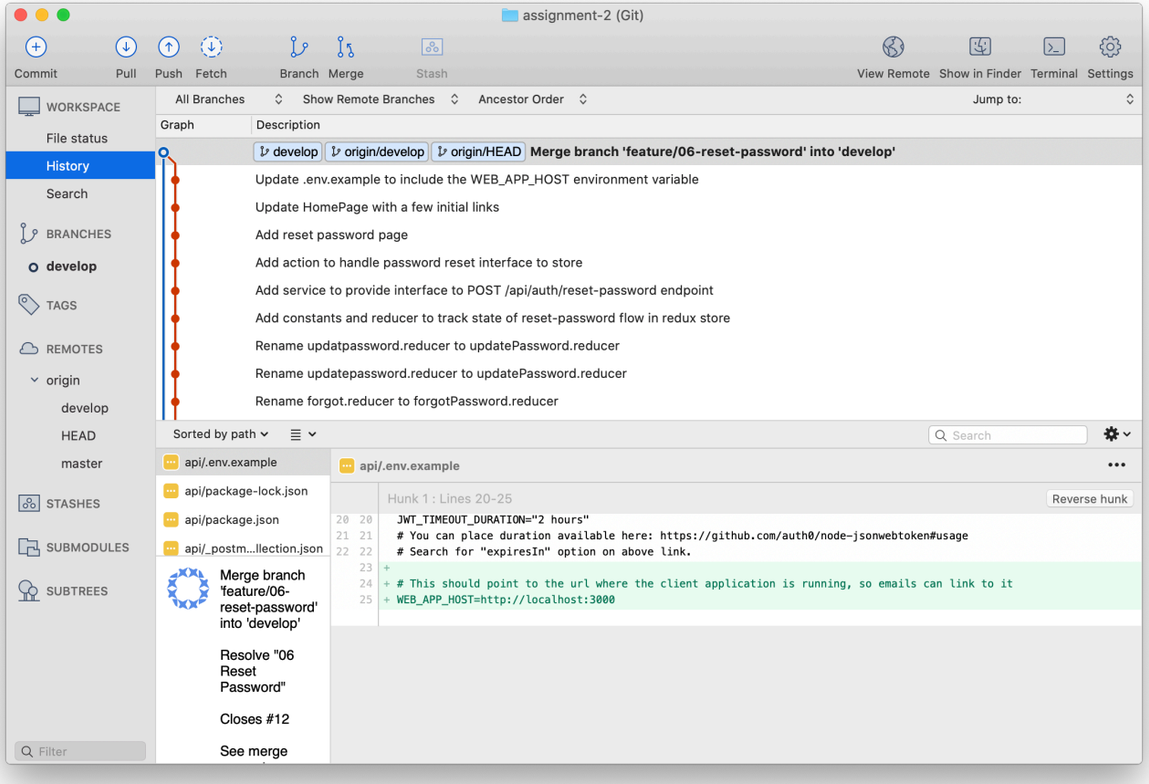
In Sourcetree: Clone the repository (1/2)

1. In Sourcetree under the **Remote** tab you should now be able to see the **Assignment 2** repository under your namespace.
   (If you cannot see repository 2, click on the Settings Wheel and select Refresh)
2. Clone the repository (make sure to set the **Destination Path** so you can find it)
3. Once you open the repository you will already see that this repository already has some history.
4. Make sure you are on the **develop** branch



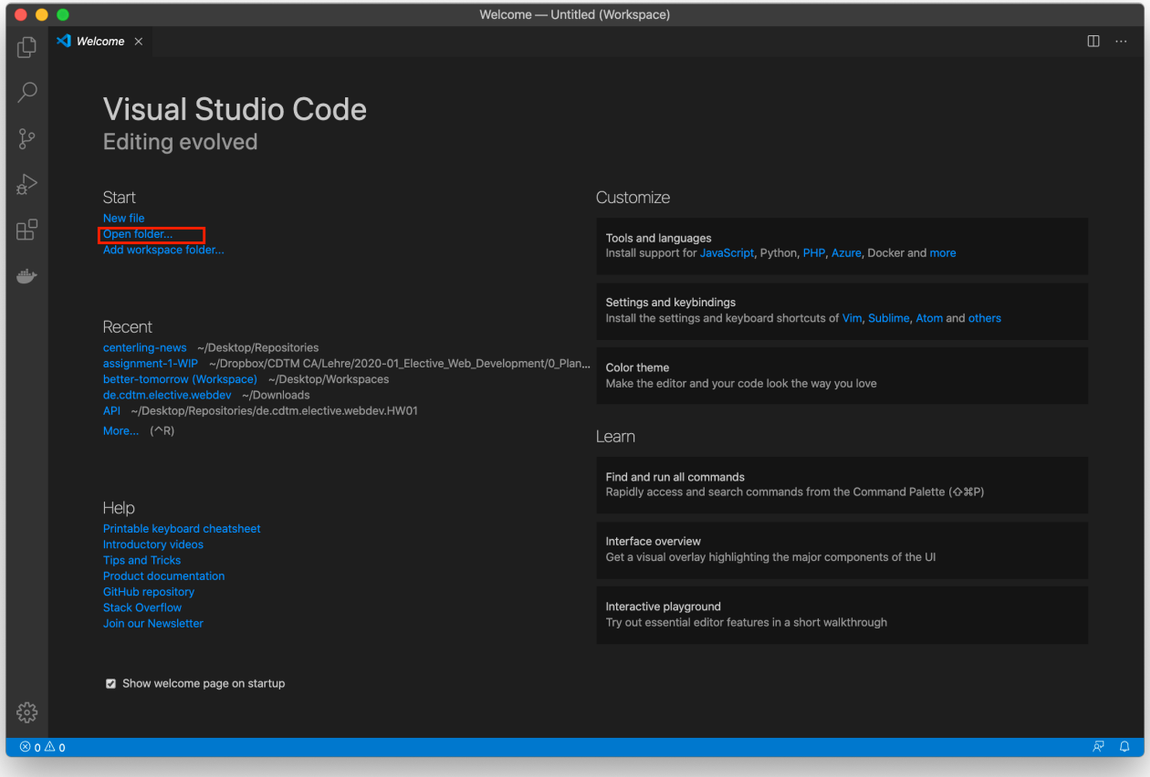
In Sourcetree: Clone the repository (2/2)

1. In Sourcetree under the **Remote** tab you should now be able to see the **Assignment 2** repository under your namespace.
   (If you cannot see repository 2, click on the Settings Wheel and select Refresh)
2. Clone the repository (make sure to set the **Destination Path** so you can find it)
3. Once you open the repository you will already see that this repository already has some history.
4. Make sure you are on the **develop** branch



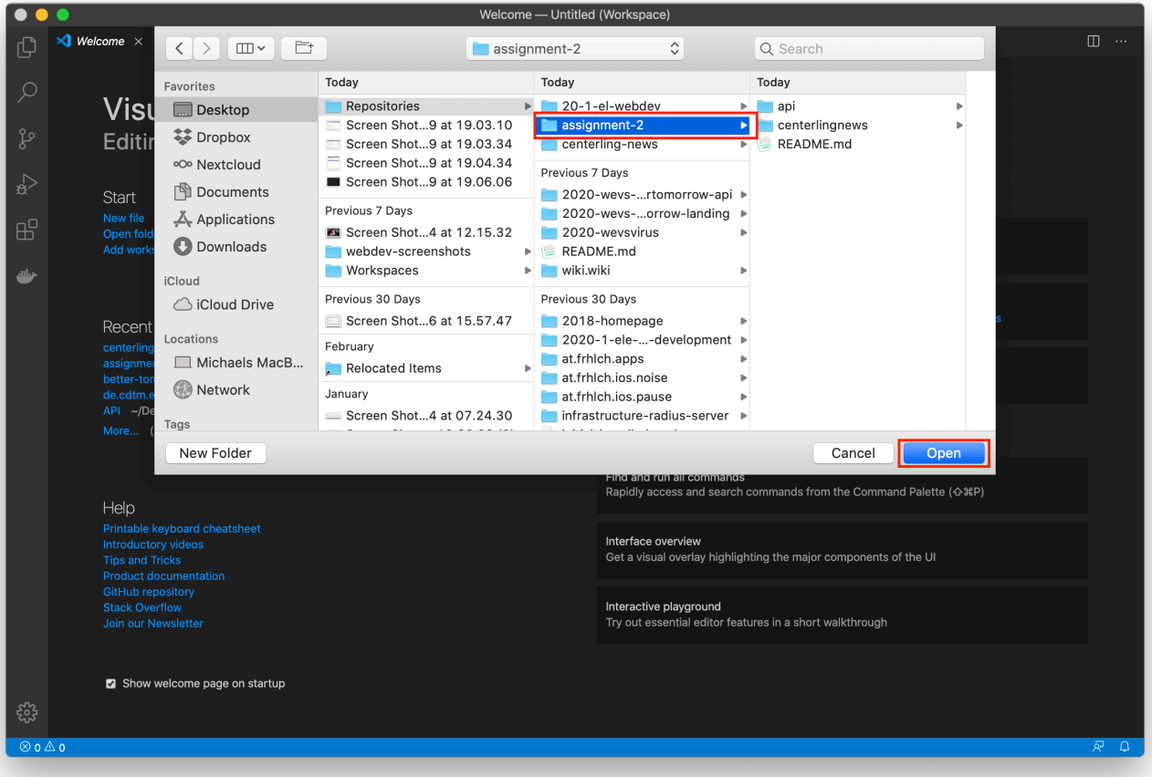
Open the repository in VS Code (1/3)

1. Open Visual Studio Code
2. Click on Open folder …
3. Find and select the repository and click on Open
4. You should now see the to folders at the root of the directory **api/** and **centerlingnews/**



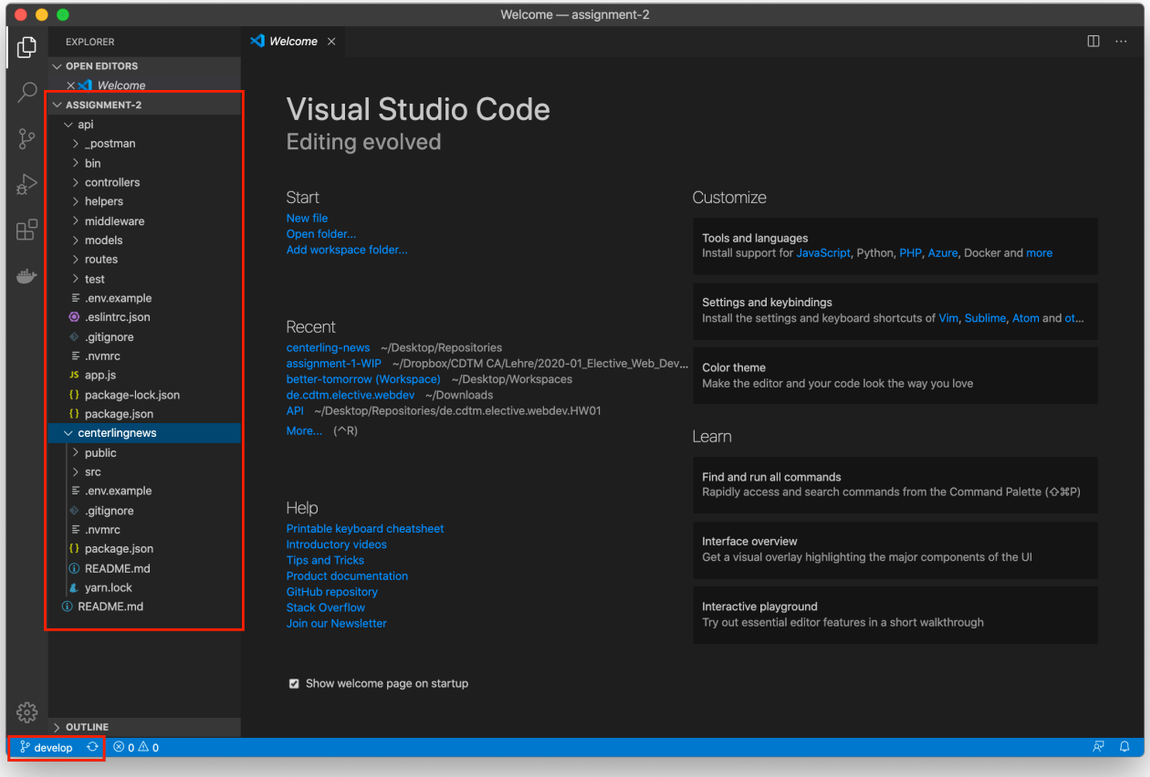
Open the repository in VS Code (2/3)

1. Open Visual Studio Code
2. Click on Open folder …
3. Find and select the repository and click on Open
4. You should now see the to folders at the root of the directory **api/** and **centerlingnews/**



Open the repository in VS Code (3/3)

1. Open Visual Studio Code
2. Click on Open folder …
3. Find and select the repository and click on Open
4. You should now see the to folders at the root of the directory **api/** and **centerlingnews/**



3. REGISTER AND SET UP MONGO ATLAS

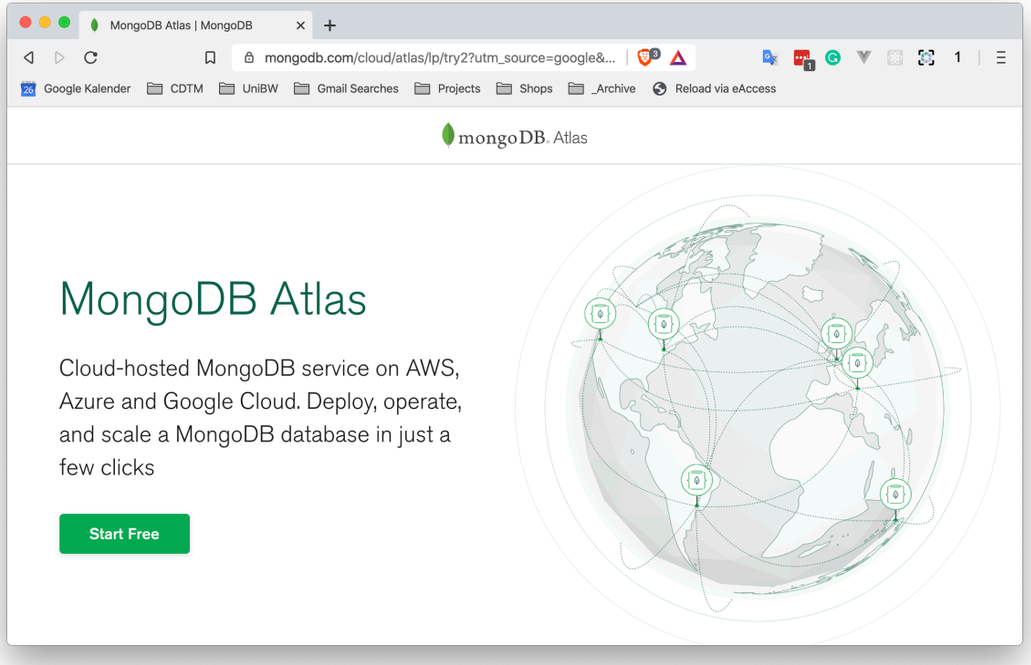
We will use a hosted MongoDB instance as our database for this exercise. This reduces SETUP effort, but also comes with a drawback. You will only be able to work on the assignment while you have a stable internet connection.

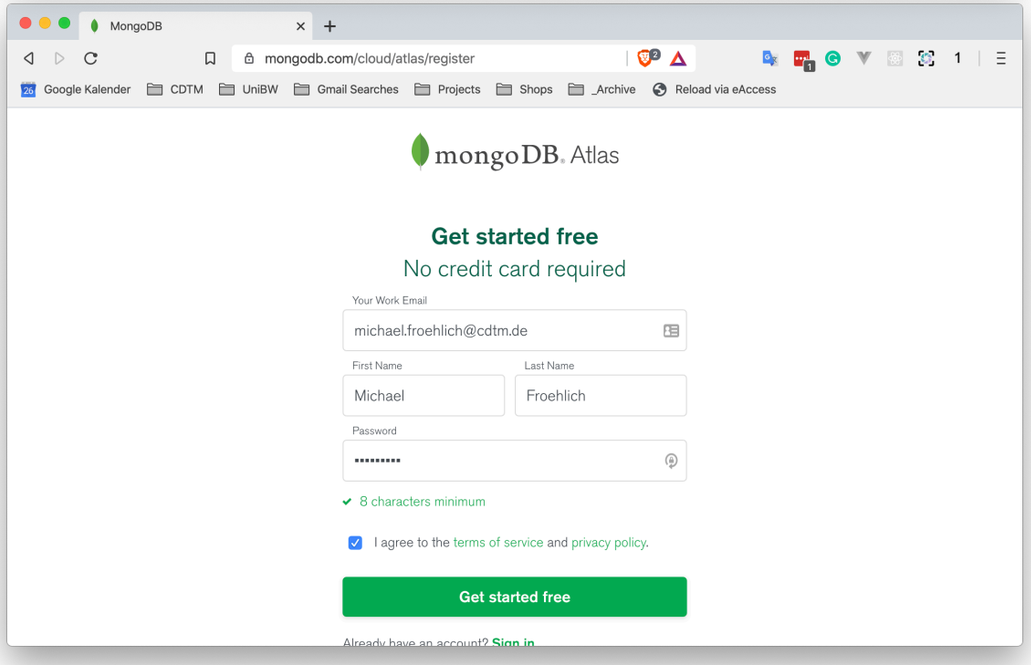
(1) If you need to work offline, consider installing MongoDB locally on your machine: https://docs.mongodb.com/manual/administration/install-community/

(2) To manually inspect the content of your MongoDB instance, you can install MongoDB Compass: https://www.mongodb.com/download-center/compass

CREATE A MONGO ATLAS ACCOUNT

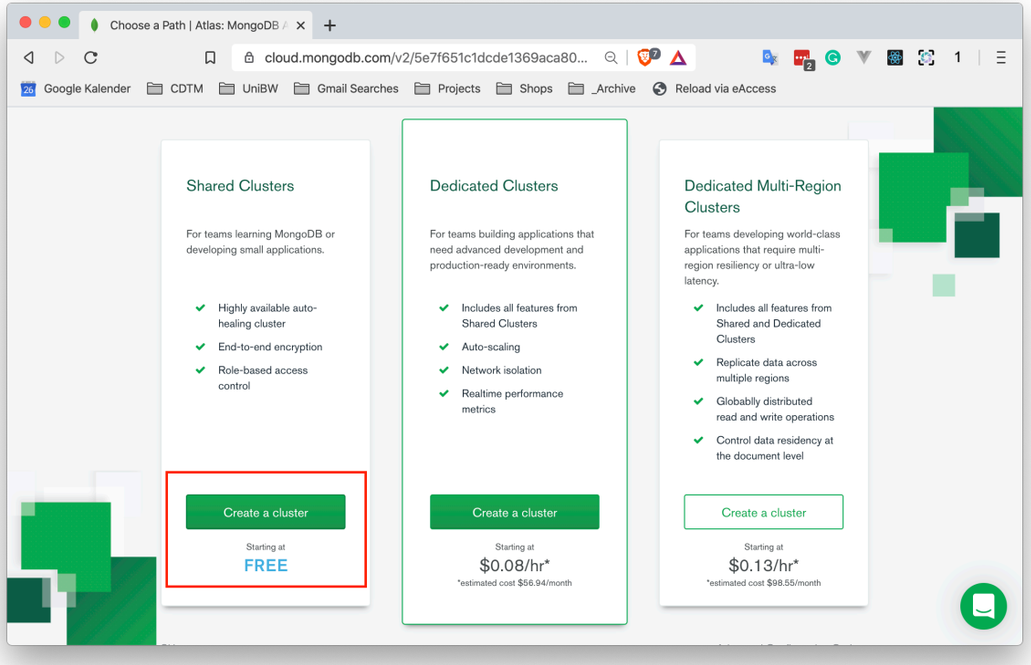
1. Open Chrome
2. Browse to https://www.mongodb.com/cloud/atlas
3. Click on “Start Free” or “Try Free”
4. Enter your information to **register a new account** and agree to the terms and conditions.

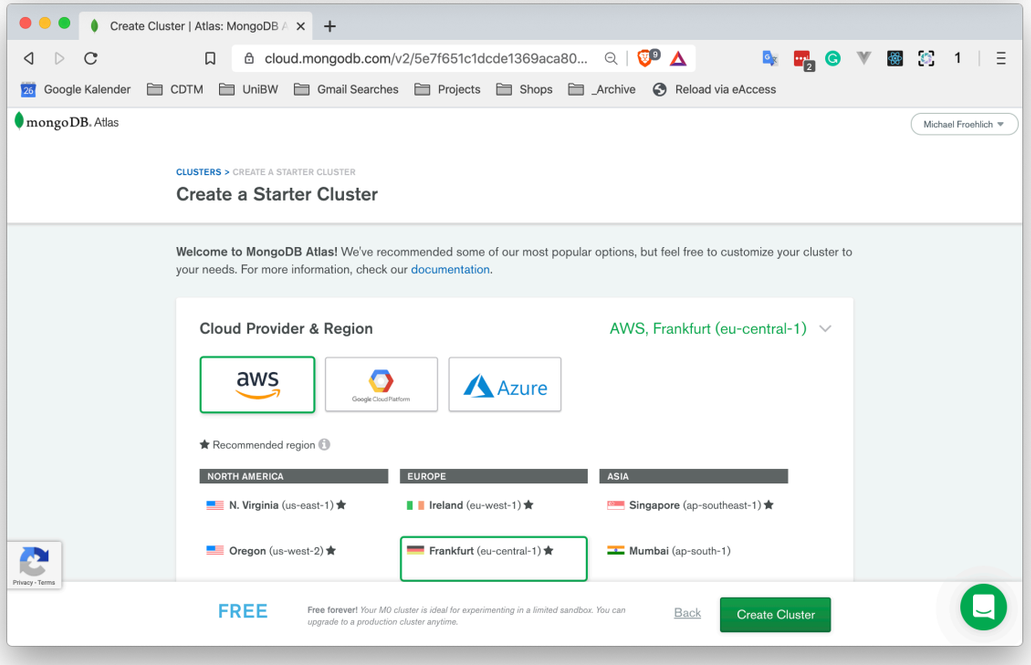




SET UP A FREE CLUSTER

1. On the next page select top create a free cluster
2. Choose AWS as Provider
3. Choose Frankfurt as Region
4. Click “Create Cluster”

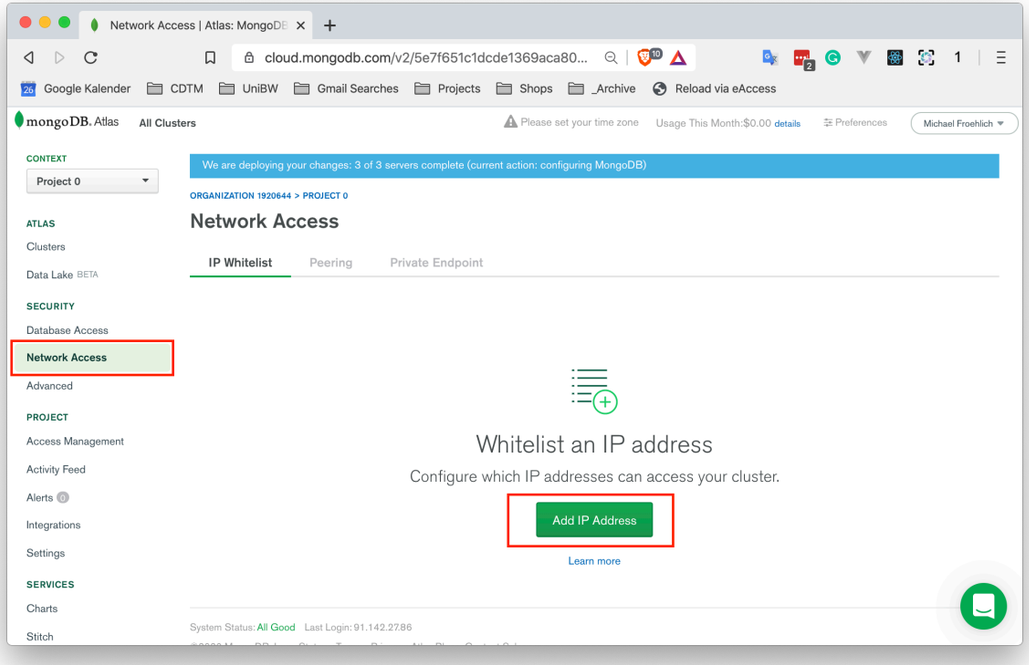


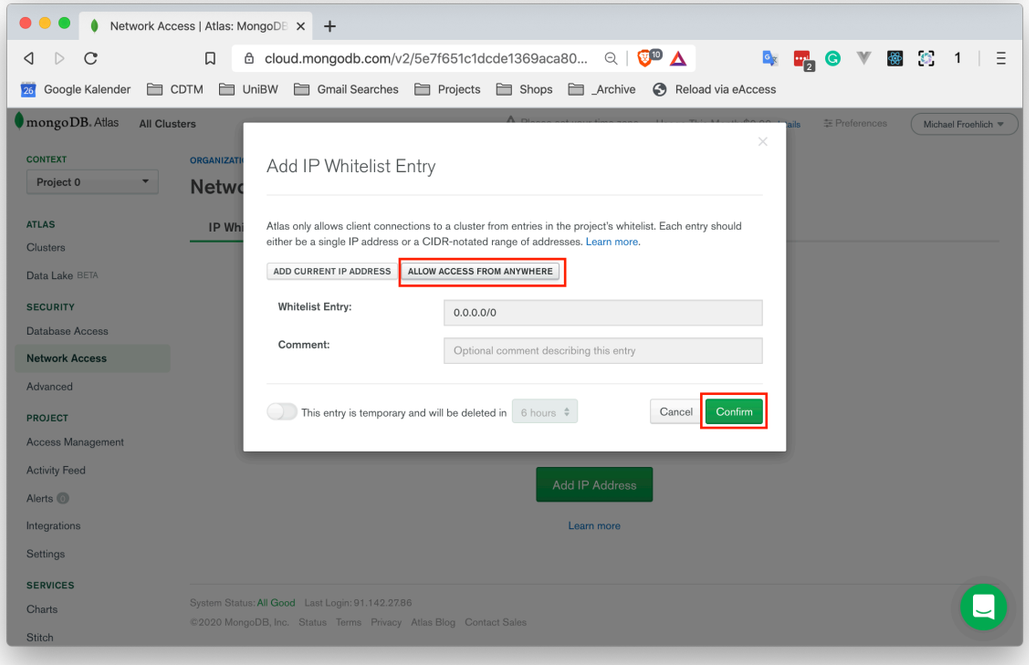


ENABLE NETWORK ACCESS FROM ANYWHERE

By default your cluster restrict access to whitelisted IP addresses for security reasons. Since we won't store critical information on the cluster, we can enable global access for this course.

1. On the menu on the left select "Network Access"
2. Click "Add IP Address"
3. In the modal, click "Allow access from anywhere"
4. Click "Confirm"



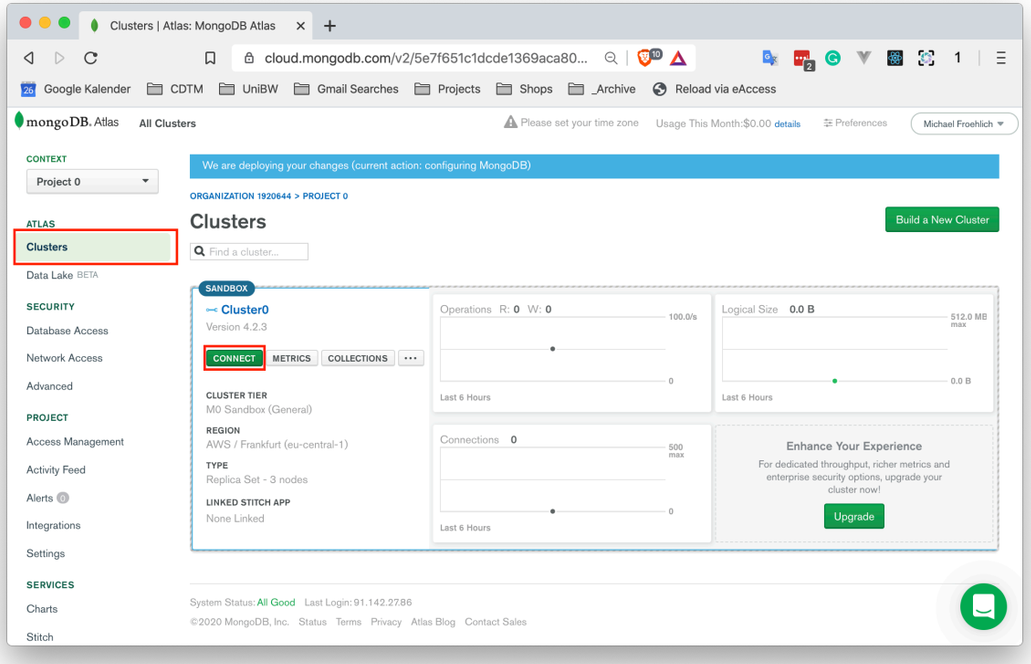


CONNECT TO YOUR MONGO ATLAS CLUSTER (1/3)

To connect the server to your cluster we need to (1) set up a user and a password and (2) copy the connection string.

1. On the menu on the left select “Clusters”
2. Click “Connect”

Note: If your cluster needs some more time until it is being created, use the wait time to take a one-function break 🚽☕🏋️

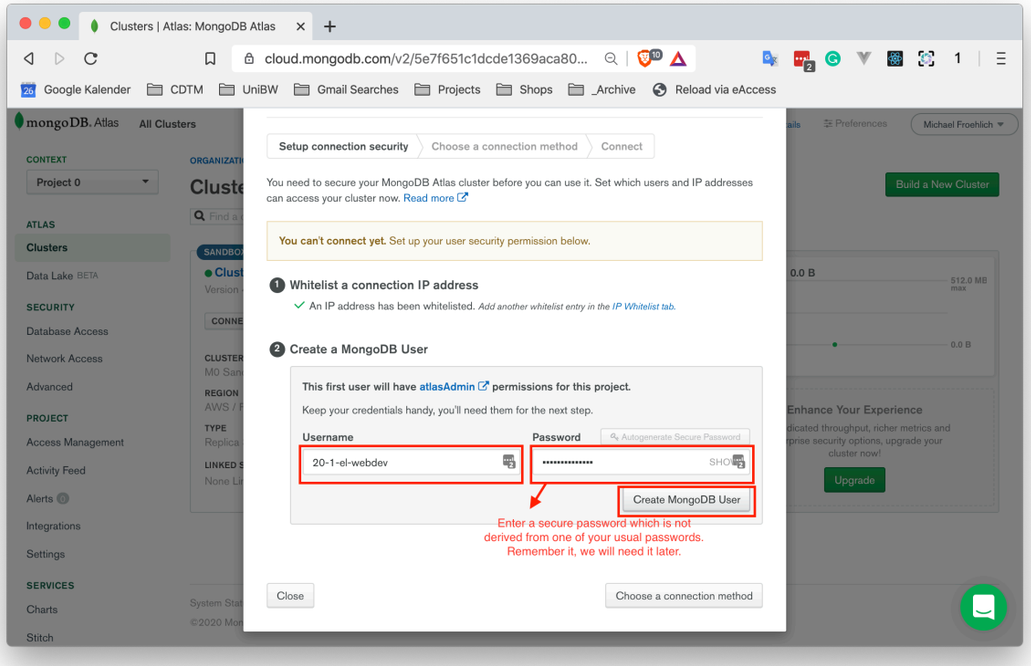


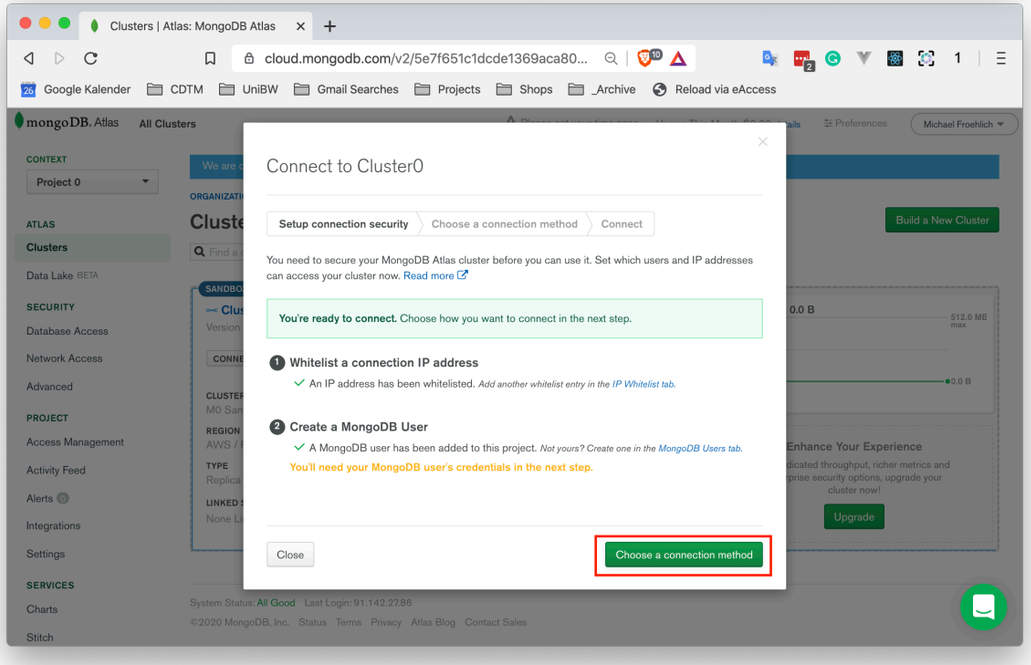
CONNECT TO YOUR MONGO ATLAS CLUSTER (2/3)

1. In the modal, we first need to create a user
   1. Enter “20-1-el-webdev” as username
   2. Enter a secure password
   3. Click “Create MongoDB User”

Important: Please do not use a password, that is derived from passwords you usually use. We will need the password in a minute, so please note it down.

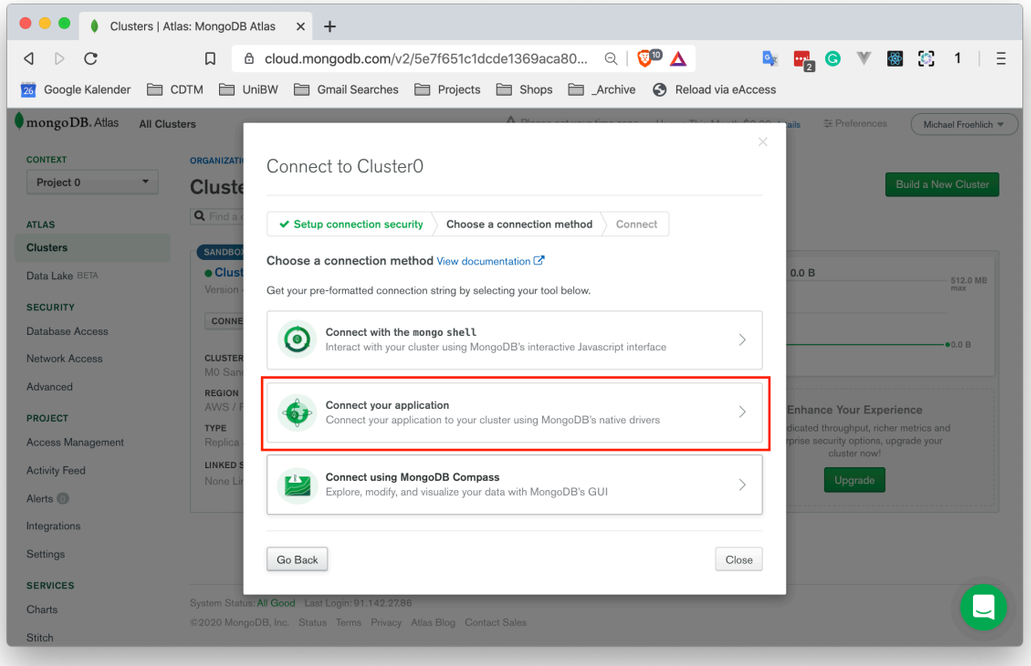
2. Click on “Choose a connection method”

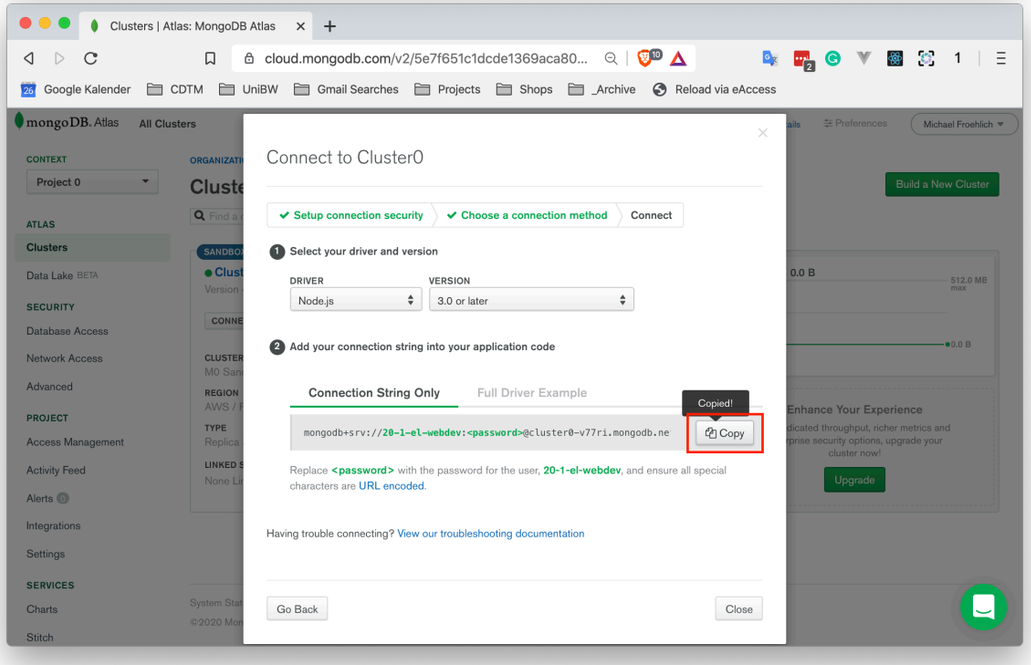




CONNECT TO YOUR MONGO ATLAS CLUSTER (3/3)

1. On the next page select “Connect your application”
2. And copy the Connection String
3. Leave this page opened in the browser

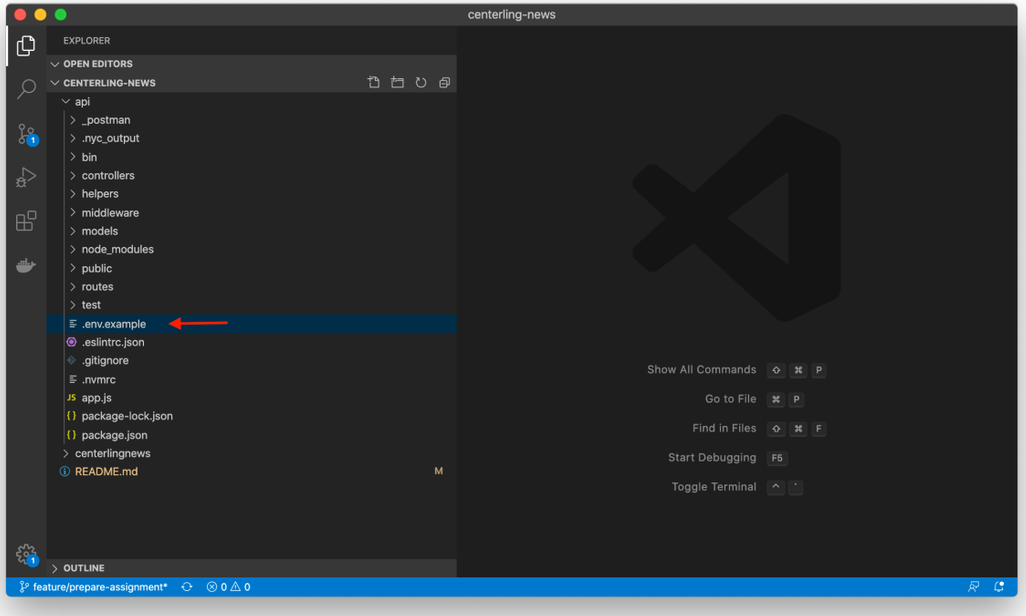


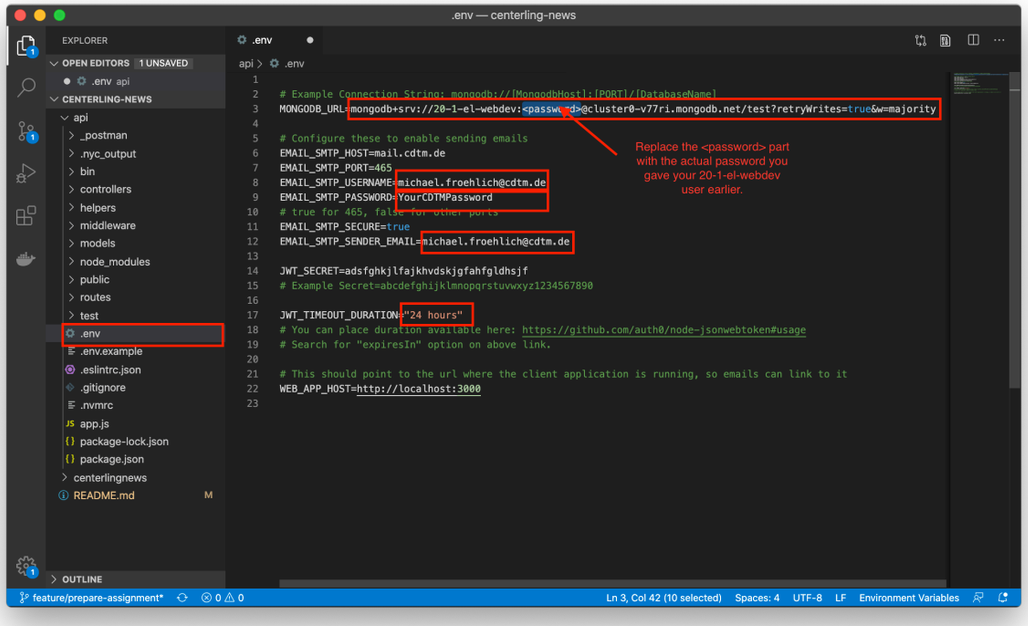


4. SET UP THE SERVER

SET THE ENVIRONMENT VARIABLES

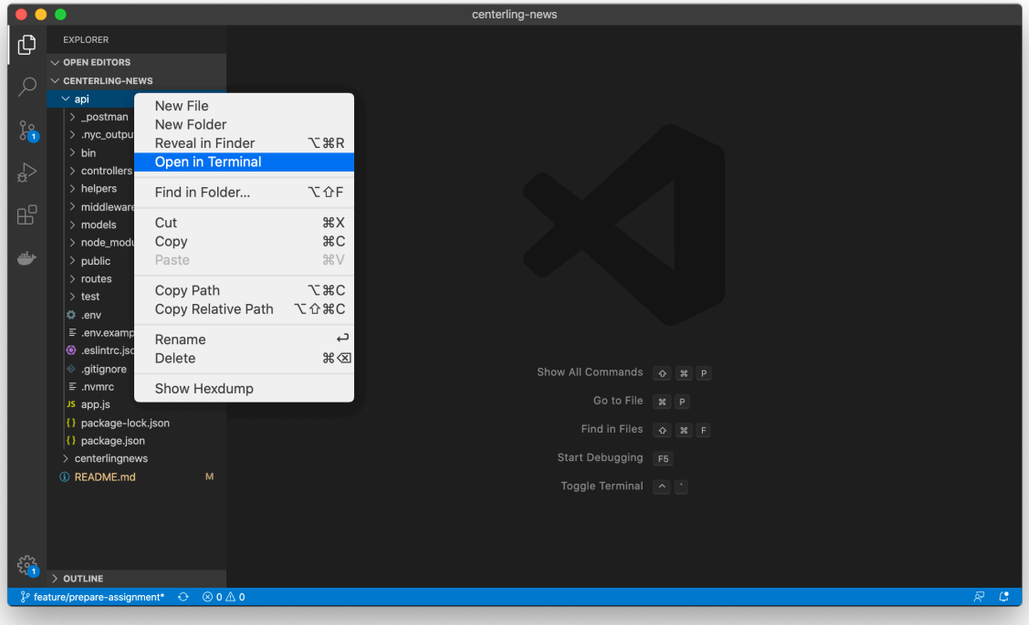
1. In the api/ folder copy the .env.example file
2. Rename the copied file to ".env"
3. Modify the following variables
    1. **MONGODB_URL**: Paste the connection string you copied from MongoDB Atlas. Make sure to replace the "<password>" part with the password you have chosen for the DB user earlier.
    2. **EMAIL_SMTP_USERNAME**: Your cdtm email address (firstname.lastname@cdtm.de)
    3. **EMAIL_SMTP_PASSWORD**: Your CDTM email password (the same you use at home.cdtm.de)
    4. **EMAIL_SMTP_SENDER_EMAIL**: Your cdtm email address
    5. **JWT_TIMEOUT_DURATION**: "24 hours"
4. Save the file

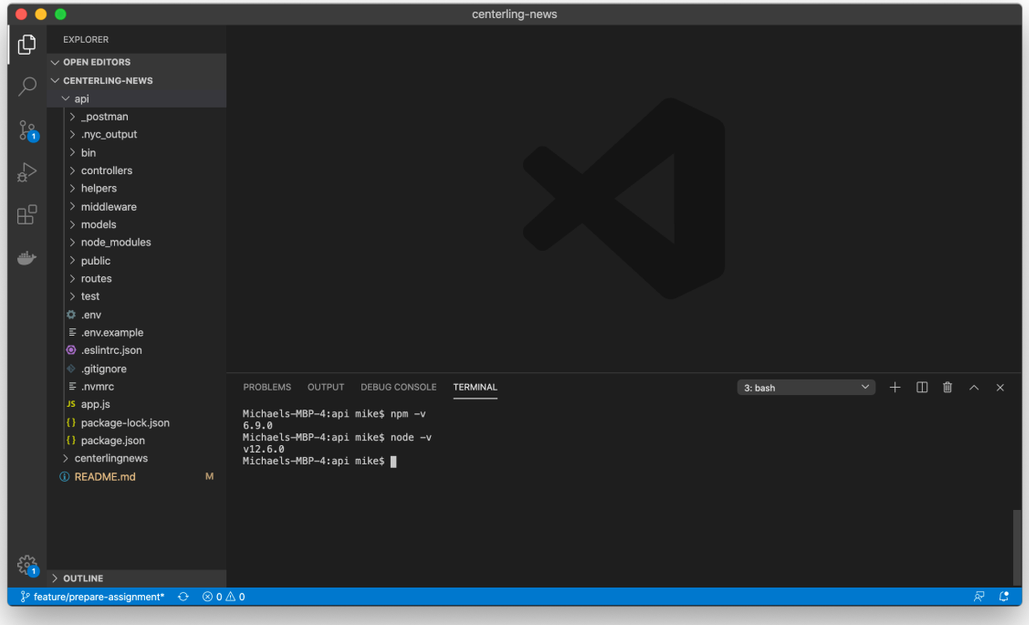




OPEN THE TERMINAL

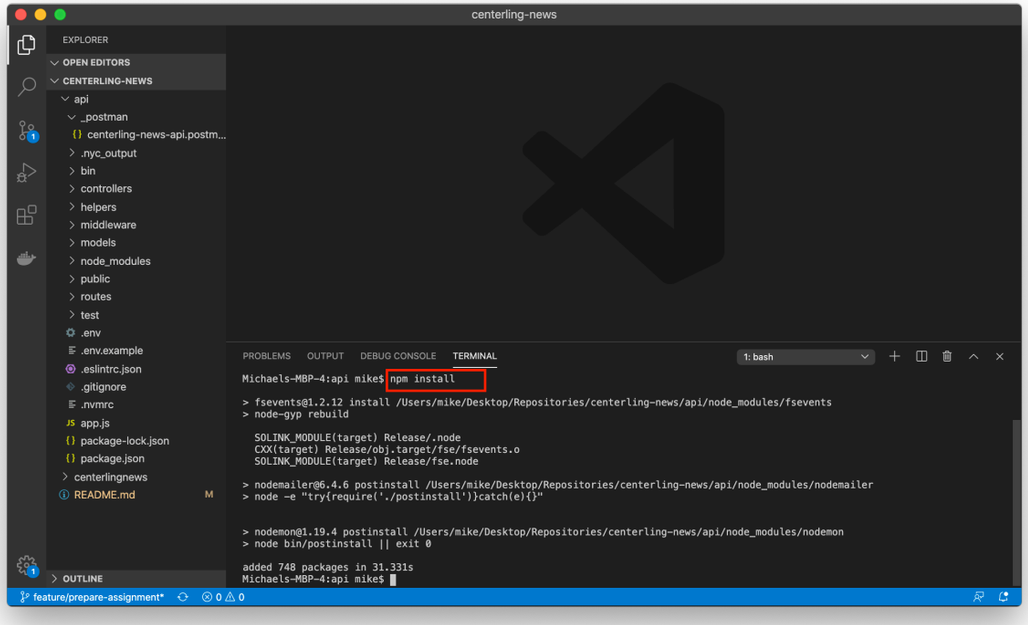
1. Right-click on the api/ folder and select Open in Terminal. (Alternatively you can also open a terminal program of your choice and cd into the api/ directory)
2. In the terminal window check the node and npm version that are running on your system.
    1. Check the node version: `node -v`
    2. Check the npm version: `npm -v`
    3. You should have a node version, running, which is >= 12.6.0 and npm >= 6.9.0.





INSTALL THE DEPENDENCIES

1. In the terminal enter `npm install` to install all the required dependencies.
2. The required dependencies are stored in the package.json file, if you want to look them up.
3. To install new dependencies and third-party packages in the future (not now) you can do so by running `npm install --save <packagename>`



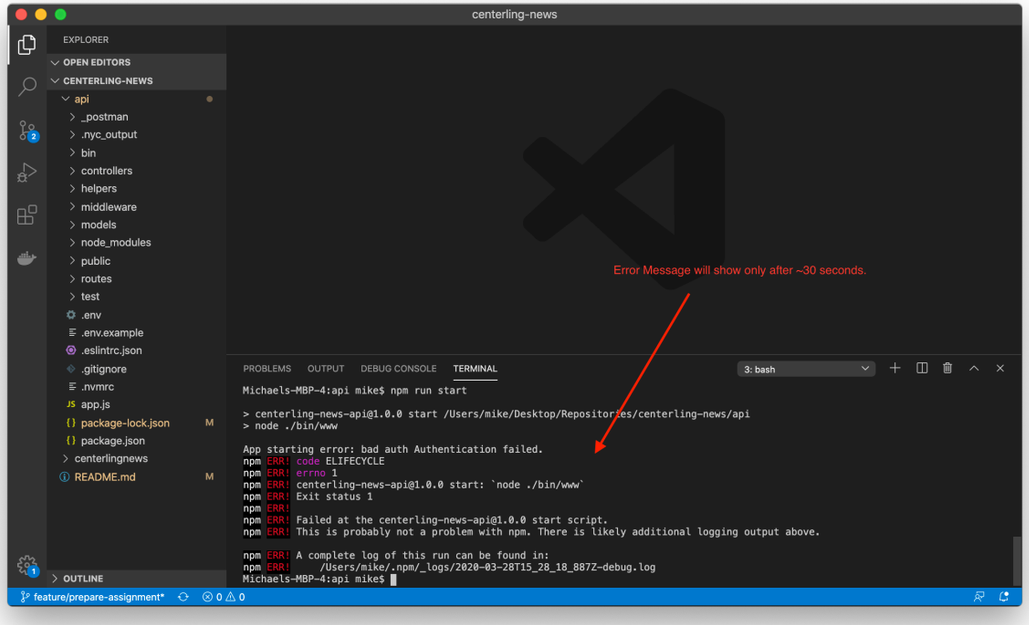
START THE SERVER

1. To start the server in development mode you can now run `npm run dev`

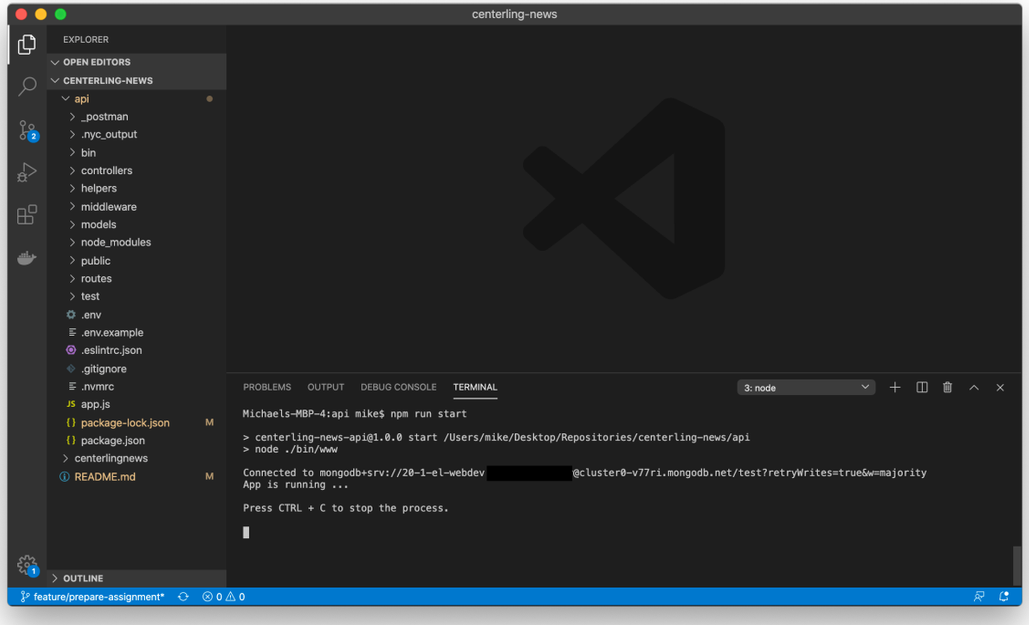
   This executes the start command you can find in the package.json file. It starts up the server at port 5000 on your laptop.

2. If everything worked, you should see the connection string printed to the terminal
3. If the server could not connect to MongoDB Atlas, it will show an error message after approximately 30 seconds. If this is the case, double check your connection string.

FAILURE

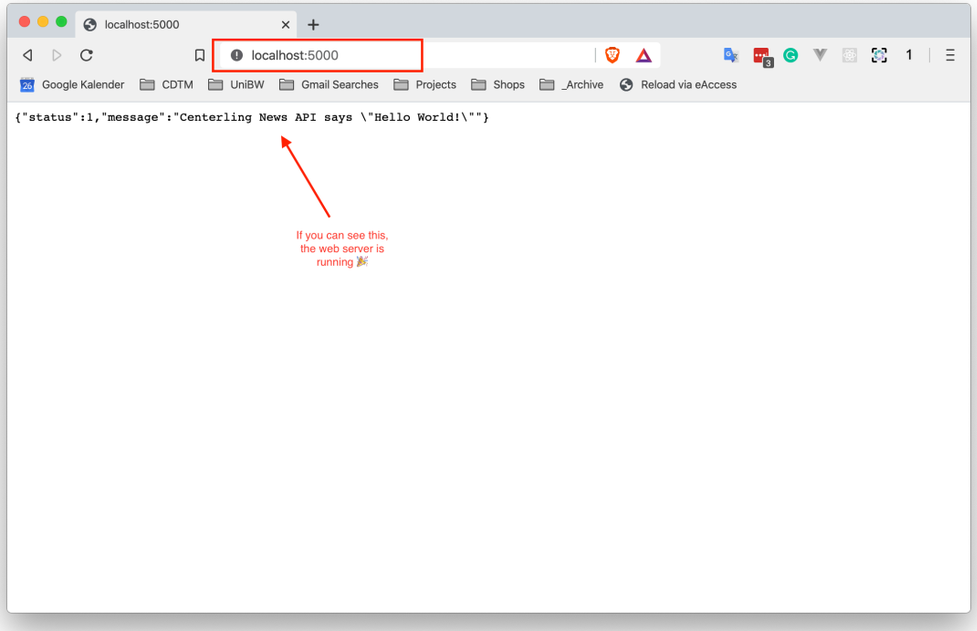


SUCCESS



BROWSER TO LOCALHOST:5000

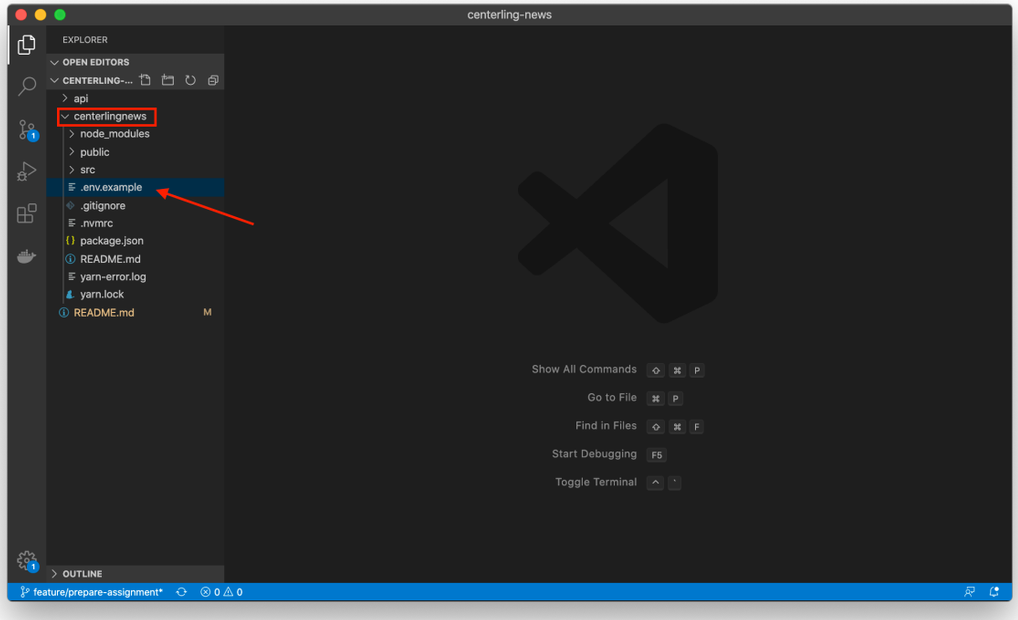
1. Open a new tab in Chrome
2. Enter localhost:5000 or alternatively 127.0.0.1:5000 in the address bar and press enter.
3. You should a welcome message from the server (see screenshot)

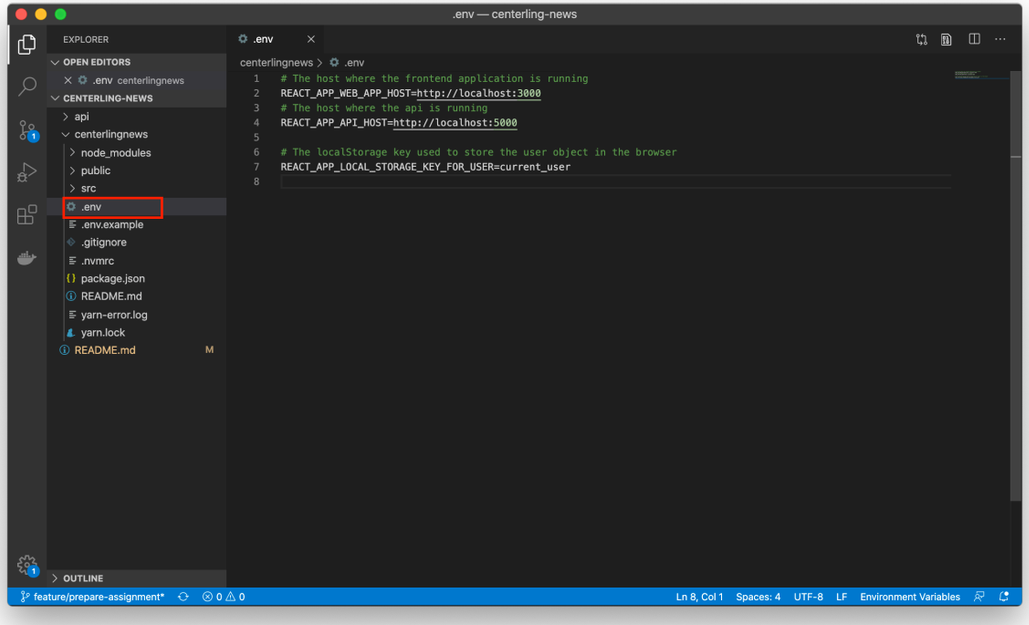


5. SET UP THE REACT CLIENT

SET THE ENVIRONMENT VARIABLES

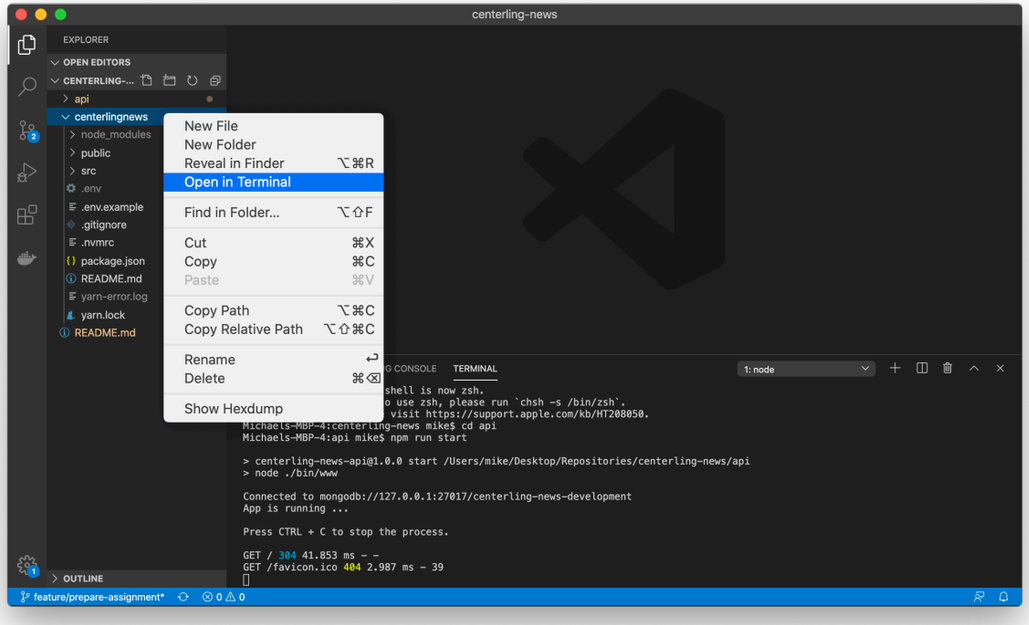
1. In the centerlingsnews/ folder copy the .env.example file
2. Rename the copied file to ".env"
3. Save the file

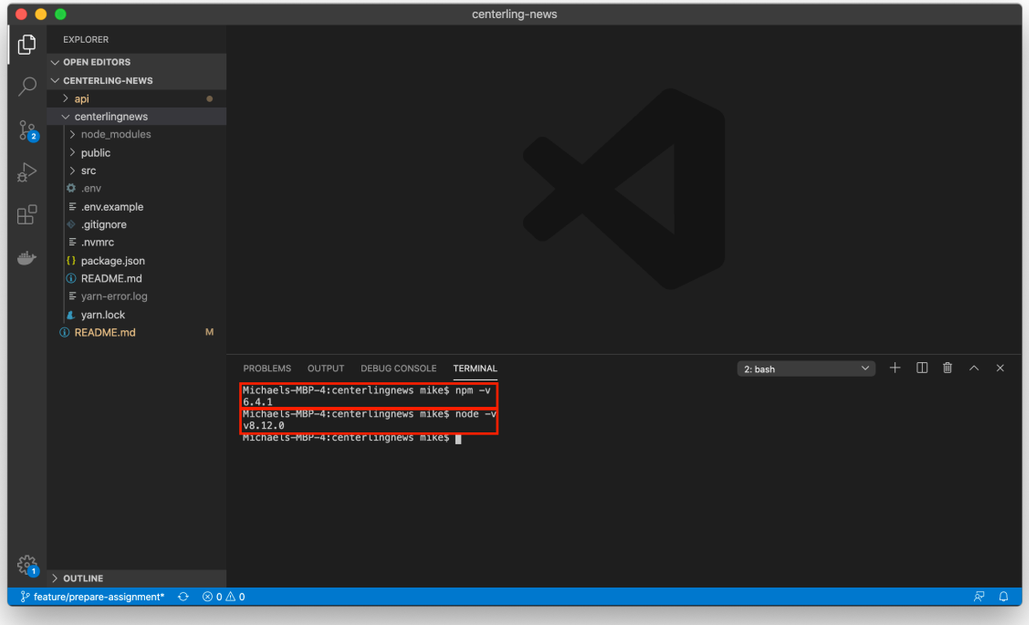




OPEN THE TERMINAL

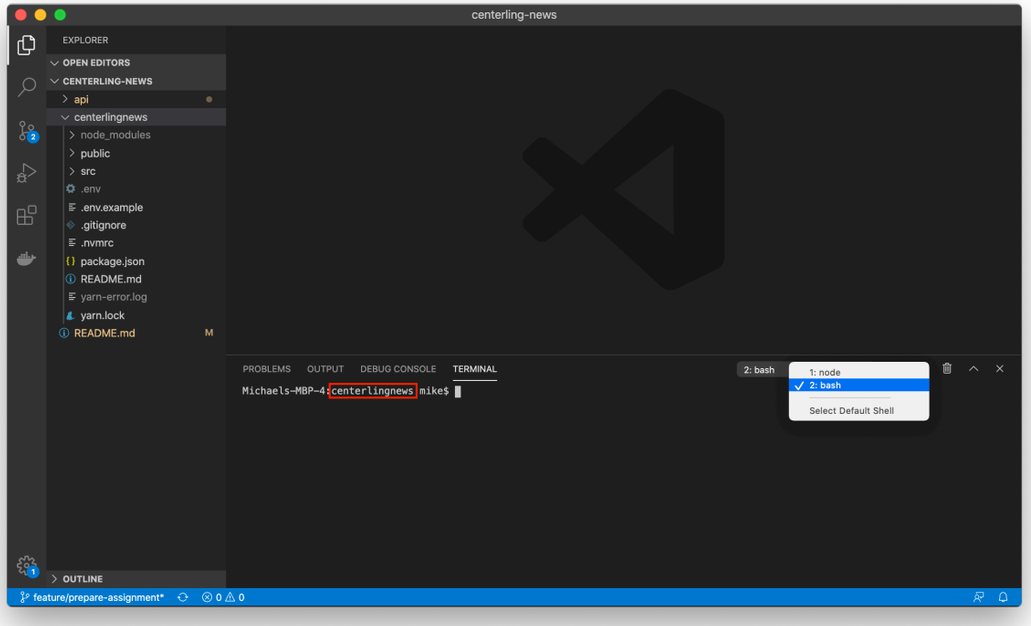
1. Right-click on the api/ folder and select Open in Terminal. (Alternatively you can also open a terminal program of your choice and cd into the api/ directory)
2. In the terminal window check the node and npm version that are running on your system.
    1. Check the node version: `node -v`
    2. Check the npm version: `npm -v`
    3. You should have a node version, running, which is >= 12.6.0 and npm >= 6.9.0.





HOW TO SWITCH BETWEEN TERMINALS

1. If you have the terminals opened in VS Code you might be wondering, where your “Server Terminal” just went and how to reopen it.
2. It is still there. You can switch between terminal windows in the dropdown on the right.



INSTALL YARN

MacOS:

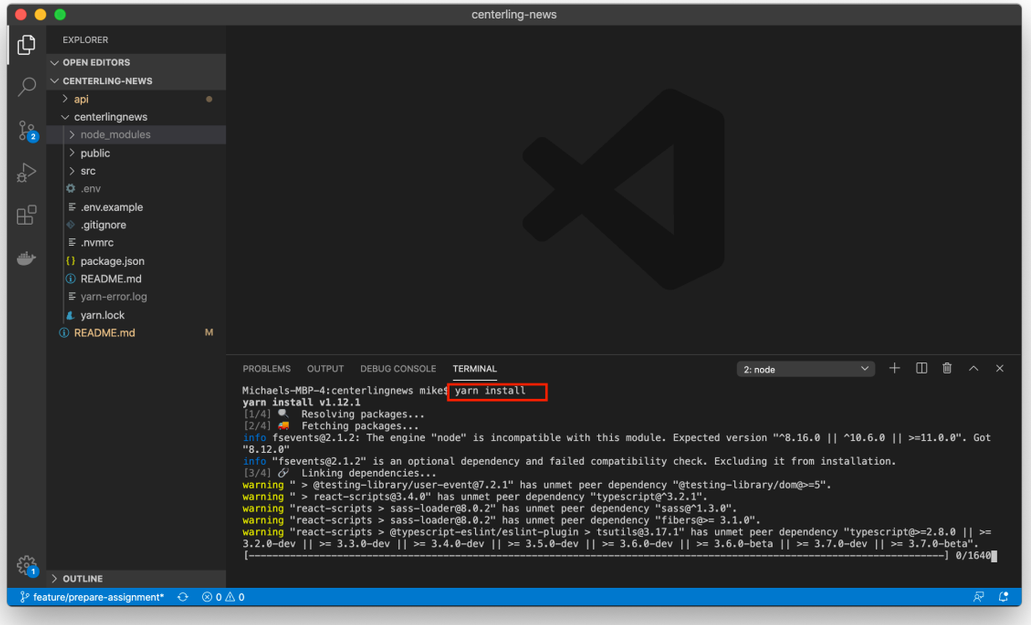
- Run `curl -o- -L https://yarnpkg.com/install.sh | bash`
- Close and reopen the terminal
- (Source: https://classic.yarnpkg.com/en/docs/install/#mac-stable)

MacOS:

- Download the Installer and run it: https://classic.yarnpkg.com/en/docs/install/#windows-stable
- Close and reopen the terminal

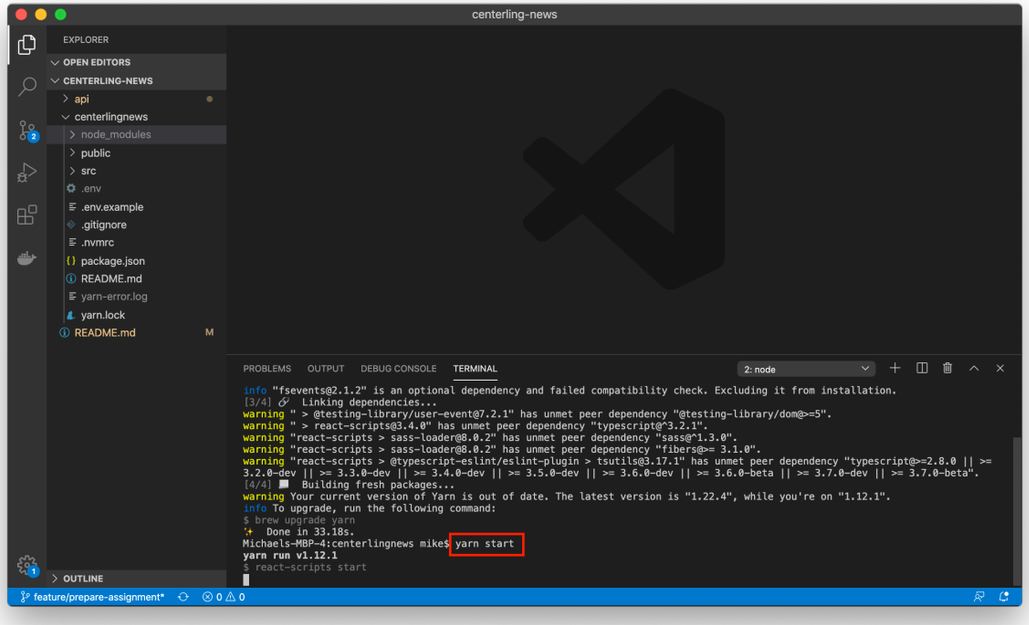
INSTALL THE DEPENDENCIES

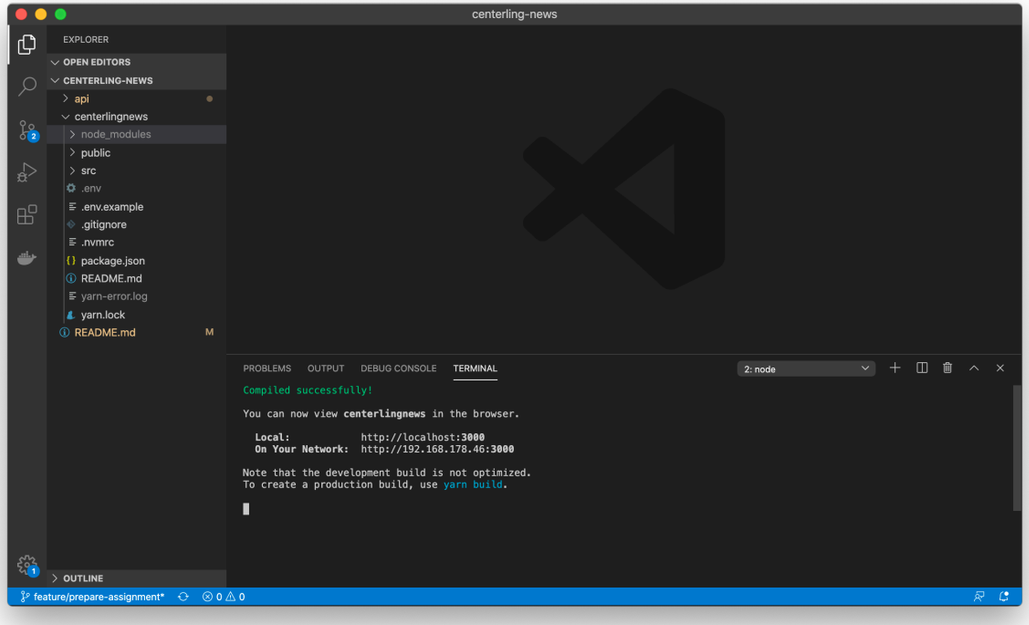
1. In the terminal enter `yarn install` to install all the required dependencies.
    1. Yarn is an alternative package manager to npm which uses the same repository behind the scenes.
    2. It is developed and maintained by Facebook, as is React.
    3. Yarn and React therefore go well together and we will use yarn instead of npm for the frontend
2. The required dependencies are stored in the package.json file, if you want to look them up.
3. To install new dependencies and third-party packages in the future (not now) you can do so by running `yarn add <packagename>`



START THE DEVELOPMENT SERVER

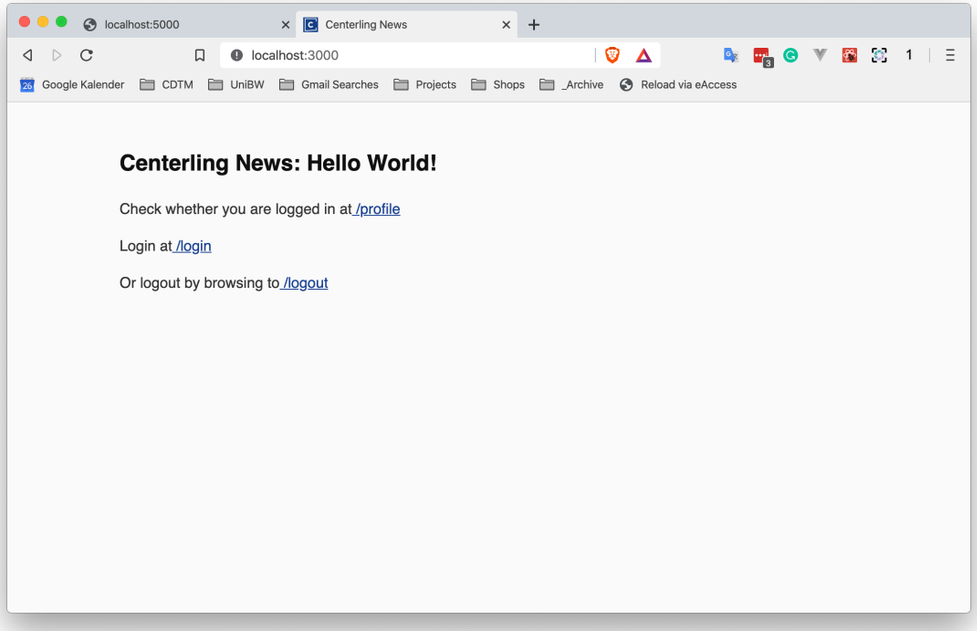
1. To run our frontend we need to start the development server.
    1. Yes, our frontend also needs a "server" during development.
    2. This is required as React needs to be bundled and transpiled to Vanilla JS before it can run in the browser.
    3. It also gives us advanced features such as hot reloading and simulates a more real deployment scenario.
2. Run `yarn start`
3. If everything works, the terminal will show you a message with the address where the site is running (most probably localhost:3000).





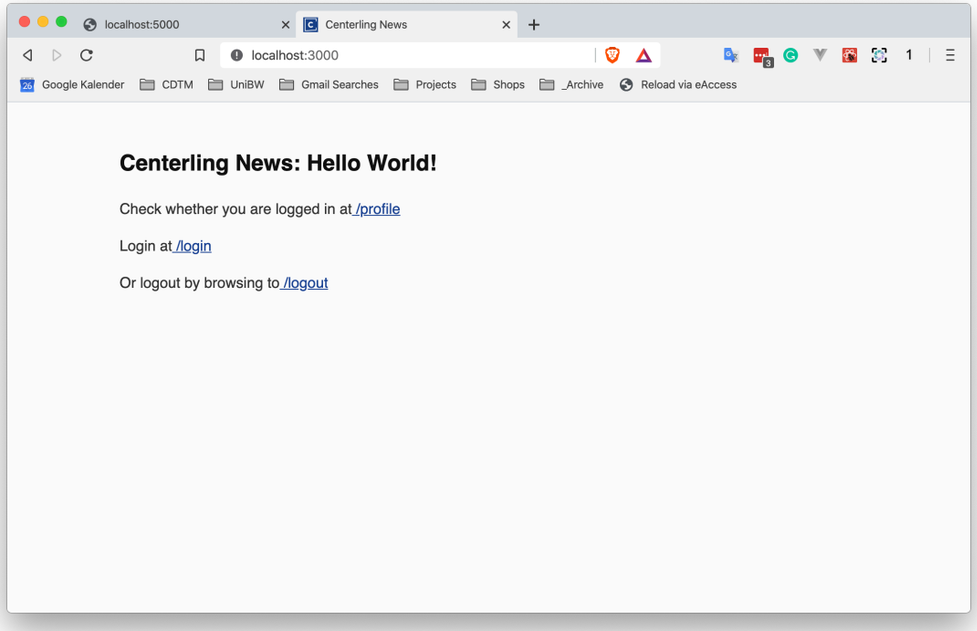
BROWSER TO LOCALHOST:3000

1. Open a new tab in Chrome
2. Enter localhost:3000 or alternatively 127.0.0.1:3000 in the address bar and press enter.
3. You should a welcome message from the server (see screenshot)
4. Click around and try to register a new account to check whether your email account is set up properly.



REGISTER A NEW ACCOUNT

1. On localhost:3000 browser to **/register**
2. Fill out the information (with a valid email address) and create a new account.
3. You should receive an email (from your @cdtm.de email address



DEVELOPMENT WORKFLOW

We have prepared a repository for you to start this exercise with. It consists of an Express server providing an API and a React application as frontend.

The prepared setup provides basics authentication – registering new accounts, confirming their email, login and logout functionality as well as reset password functionality.

/API

In this folder you can find the express server providing the api.(not all folders are mentioned here)

```
– api
    |– .nvmrc                 // the node version for the repository
    |– package.json           // the package.json contains installed npm packages
    |– .env                   // the .env file configuring the environment variables
    |– app.js                 // the initial configuration of the express server
    |– model/                 // contains the mongoose models
    |– routes/                // contains all the different routes
    |– controllers/           // contains the controller (e.g. PostController, etc …)
    |– middlewares/           // contains custom middleware files
    |– helpers/               // contains additional helper functions
    |– test/                  // contains tests for the existing api routes
    |–_postman/               // you can import the file in Postman to manually test the existing api routes
```

How to run the server:

In the terminal make sure you are in the /api folder.

(1) Initial startup

- (Optional) Make sure you have the same version as in the .nvmrc file
- Copy the .env.example to .env and fill in the required information
- Install the required packages by running `npm install`
- Start the server by running `npm run dev`

(2) Regular startup

- `npm run dev` `(for development)`
- `npm run start` `(for deployment)`

/CENTERLINGNEWS

In this folder you can find the React application.

```
– centerlingnews
   |– .nvmrc                  // the node version for the repository
   |– package.json            // the package.json contain installed npm packages
   |– .env                    // the dotenv file configuring the environment variables
   |– index.js                // the root file
   |– App.js                  // the root component
   |– views/                  // contains components for all pages
   |– components/             // contains reusable components
   |– helpers/                // contains additional helper functions
   |– resources/style/        // contains the base style for the applications
   |– store/
      |– constants/           // constants that (uniquely) identify reducer functions
      |– reducers/            // react-redux reducers
      |– actions/             // react-redux actions
      |– services/            // services performing calls to the api
```

How to run the development server:

In the terminal make sure you are in the /api folder.

(1) Initial startup

- (Optional) Make sure you have the same version as in the .nvmrc file
- Copy the .env.example to .env and fill in the required information
- Install the required packages by running `yarn install`
- Start the server by running `yarn start`

(2) Regular startup

- `yarn start`

REPOSITORY STRUCTURE AND PROJECT SETUP

Here are some videos walking you through the repository and project setup (optional)

Introduction (1:09)
https://www.loom.com/share/53a0075bc1d1446b84b87bcf5d922ad3

Gitlab (10:20)
https://www.loom.com/share/5344f24bae1b414c9502d4303ab0a239

Sourcetree (11:01)
https://www.loom.com/share/e8189dd7b2b642b49df15ec26e23bbb2

Express (19:08)
https://www.loom.com/share/e75e48f1eebd43749be8337cba2f1675

Making HTTP requests with Postman and fetch() (22:05)
https://www.loom.com/share/deee1c7452304534b4e48cc9db787adc

React (8:34)
https://www.loom.com/share/71b0972fc9114a469835c1b91b483168

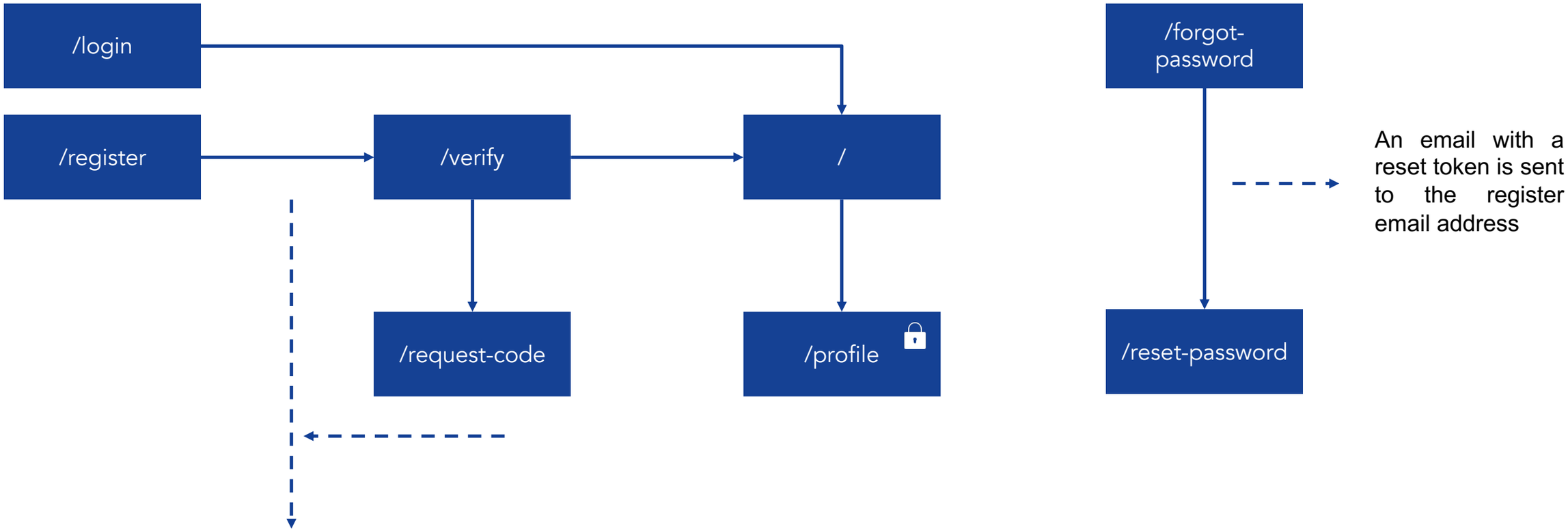
EXISTING API POINTS

Here is an overview of existing API endpoints.

POST /api/auth/register	required parameters (firstName, lastName, username, email, password)
POST /api/auth/verify-otp	required parameters (email, otp)
POST /api/auth/resend-verify-otp	required parameters (email)
POST /api/auth/login	required parameters (email, password)
GET /api/auth/user	required header: `Authorization: Bearer <token>`
PATCH /api/auth/update	required header: `Authorization: Bearer <token>` required parameters (firstName, lastName),
PATCH /api/auth/update-password	required header: `Authorization: Bearer <token>` required parameters (password, newPassword)
POST /api/auth/send-reset-password	required parameters (email)
POST /api/auth/reset-password	required parameters (token, password)

APPLICATION LAYOUT

Here is the authentication flow for the prepared repository. All urls refer to the route in the frontend (React App), not the API endpoints.



AUTHENTICATION – JSON WEB TOKENS (JWT)

Here is a bit of information on how the authentication works behind the scenes.

How does the server detect that a user is logged in?

- The HTTP request needs to have an `Authorization: Bearer <token>` header present
- The `<token>` must be a valid JWT

What is a JWT?

- It is (basically) an encrypted JSON object
- The JWT is encrypted with the secret from the `JWT_SECRET` environment variable
- With the `JWT_SECRET` known, the server can decrypt the token when it receives a request and read the information

Where can I see the JWT code?

- /api/helpers/utility.js
  // creates a new JWT
- /api/middleware/authenticationRequired.js
  // checks for a valid JWT in the request header
  - if present and valid: stores the decrypted json in req.user
  - if not present or valid: rejects the request with a 401
- /centerlingsnews/helpers/authHeader.js
  // returns a header object for fetch requests

More Info: https://blog.angular-university.io/angular-jwt/

TYPICAL DEVELOPMENT WORKFLOW (EXAMPLE)

To help you get started with developing your first user stories I have added an exemplary workflow of how I would approach the development process. This is not by any means how you need to do it, but another data point to give you orientation.

(A) Gitlab / Sourcetree

(1) Select the issue you are working on and create a feature branch from the develop branch.
- Name it "feature/<name-of-the-feature-branch>"

(2) In Sourcetree press fetch and checkout the new feature branch
- (!) Commit often and use meaningful commit messages during development

(B) Implement the API endpoint

(1) Add a new or update an existing mongoose model
- In the models/ directory create a new file & define a schema (e.g. PostModel)
- This might not be necessary, if you are using an existing model.

(2) Create a Controller that bundles the logics for related routes
- In the controllers/ directory create a new controller (e.g. a PostController for all routes related to Posts)
- This might not be necessary, if you add it to and existing controller.

(3) Implement the functionality in the controller
- You might need to import files, e.g. the model(s) or other files.
- You might need to create and adapt other files (middleware, helpers, .env, etc)

(4) Link your function with a HTTP method and route in the routes file.
- In the routes/ folder create a new Router (e.g. postRouter) and define the route there. Link the postRouter int the routes/api.js file.

(5) Test your route with Postman.

(C) Implement the Frontend

(1) Create a new file/ files in the views folder
- Add the .jsx code in the render() function that defines the style first.
- Link an .scss or css file to be able to change the style of the specific component. Style changes affecting several views should be done in resources/style/index.scss
- Regularly check back to the terminal to see errors. Fix them :)

(2) Link the new view in the App.js file
- Note, that the route you choose here is independent of any api routes
- Use the `<Route>` component for routes that are accessible without authentication and `<PrivateRoute>` for those that should be accessible only for authenticated users.
- You can now view your new page in the browser => Style it!

(3) Fetch data (or submit) from the server
- You might do so directly in your component or use advanced concept like redux. For routes that require authentication make sure to use the header exported from helpers/authHeader.js.

(4) Figure out how to deal with errors

(5) (Potentially) Refactor your application into subcomponents
- Put code you want to reuse into functional subcomponents (e.g. Navbar)

(D) Gitlab / Sourcetree

(1) Push your code and create a Merge Request back to develop. Describe the changes you implemented and merge it.

EXAMPLE: IMPLEMENTING YOUR FIRST USER STORY

Here is a step by step guide on how to apply Git Flow to implementing your user stories in this exercise.

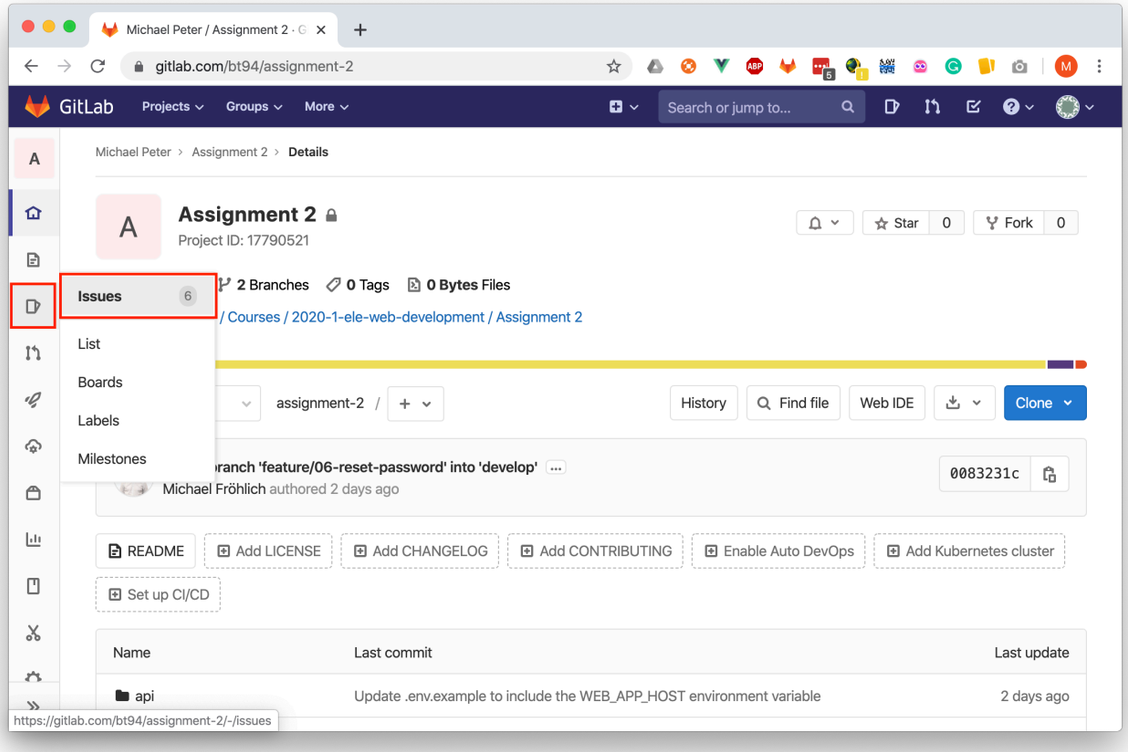
We realize that this session is densely filled with information. This should serve as a guide you can come back to during development.

Select a User Story (1/3)

1. Open Chrome
2. Browse to gitlab.com and go to the repository you are working on
3. From the side bar on the left select **Issues**
4. On the next page select the issue you want to work on.

   For Assignment II we have prefixed all issues with a number. This is the order in which we would recommend you, to implement the user stories. You are free to choose a different order.

5. Click on the issue title you selected
6. On the detail page of the issue, click on **assign self** and the copy the title of the issue

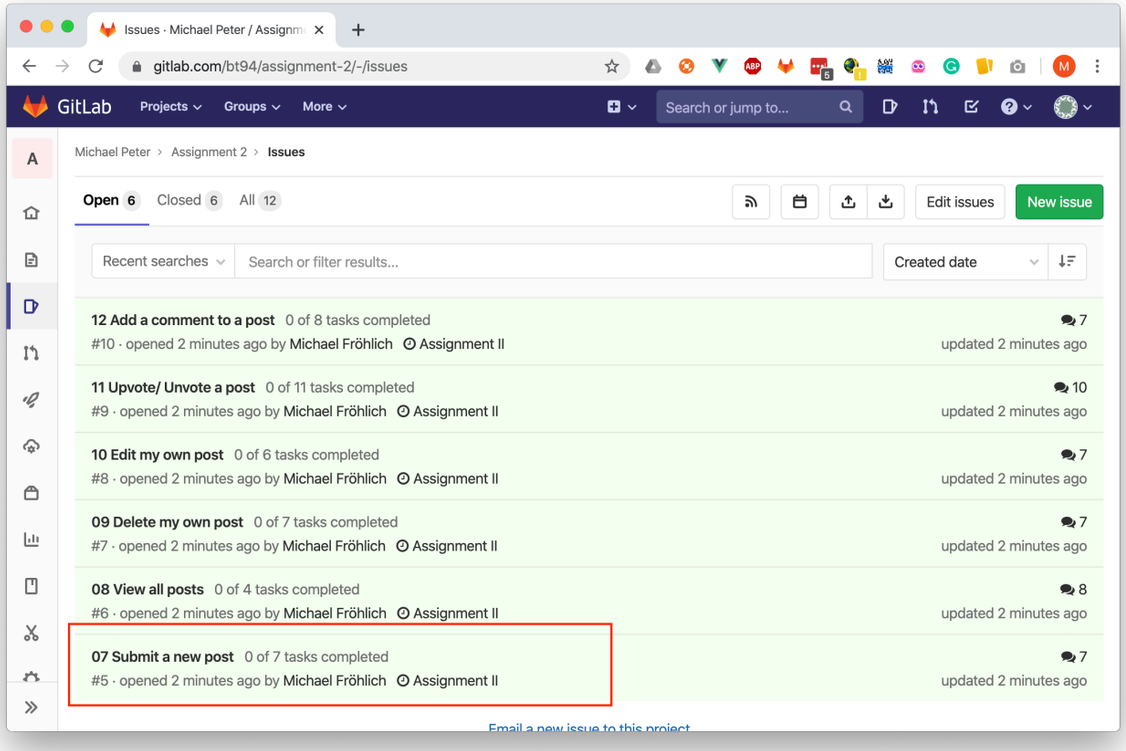


Select a User Story (2/3)

1. Open Chrome
2. Browse to gitlab.com and go to the repository you are working on
3. From the side bar on the left select **Issues**
4. On the next page select the issue you want to work on.

   For Assignment II we have prefixed all issues with a number. This is the order in which we would recommend you, to implement the user stories. You are free to choose a different order.

5. Click on the issue title you selected
6. On the detail page of the issue, click on **assign self** and the copy the title of the issue

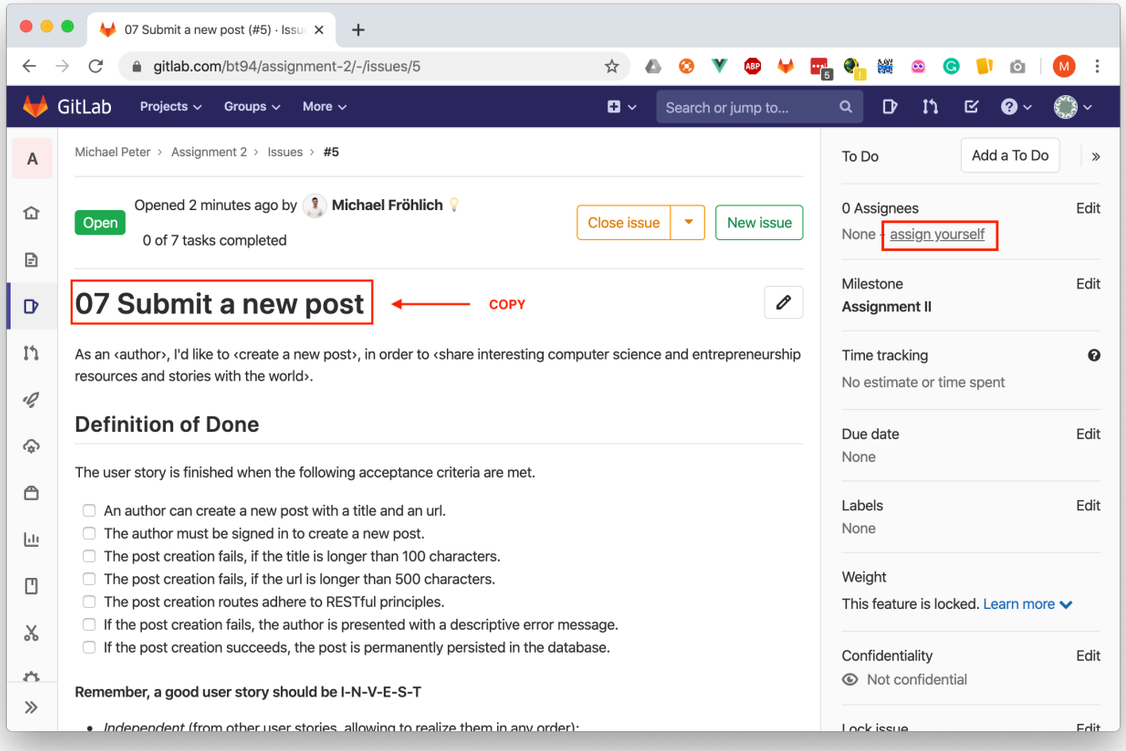


Select a User Story (3/3)

1. Open Chrome
2. Browse to gitlab.com and go to the repository you are working on
3. From the side bar on the left select **Issues**
4. On the next page select the issue you want to work on.

   For Assignment II we have prefixed all issues with a number. This is the order in which we would recommend you, to implement the user stories. You are free to choose a different order.

5. Click on the issue title you selected
6. On the detail page of the issue, click on **assign self** and the copy the title of the issue



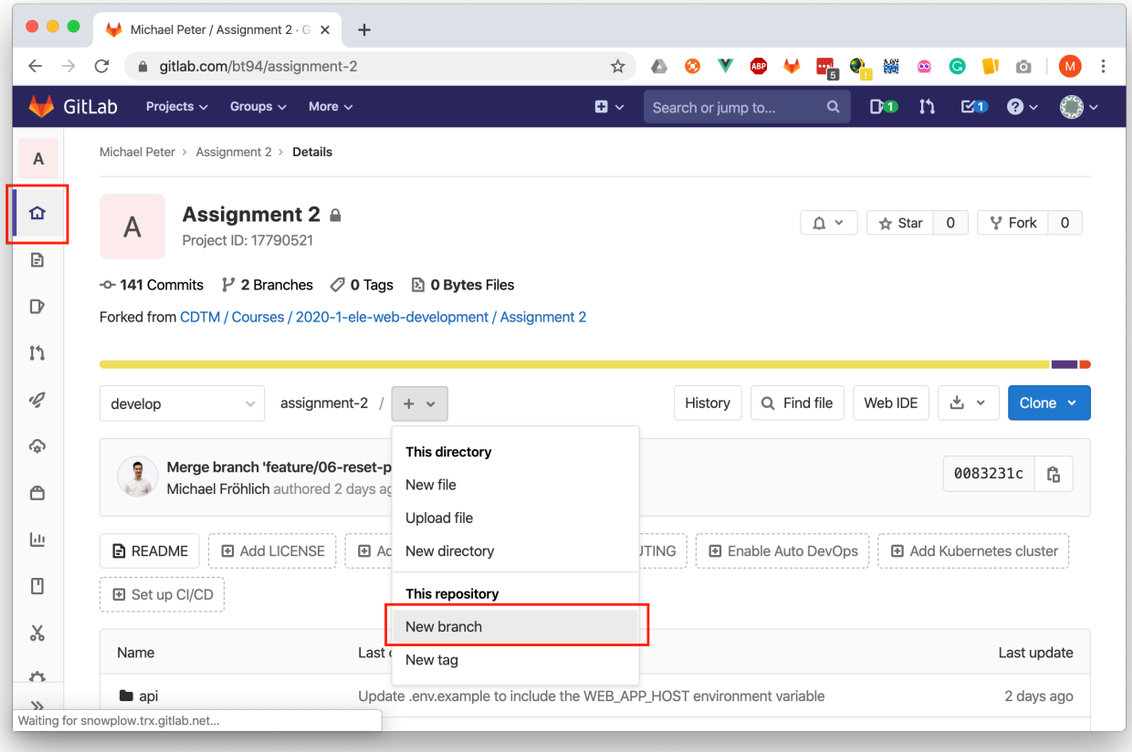
Create a feature branch (1/2)

1. From the side bar on the left select **Repository**
2. On the repository page make sure you are on the **develop** branch
3. Next click the **+** symbol and then **New branch**
4. Name the branch *feature/07-name-of-issue*

   The name after "feature/" should be identical to your issue title, except all lowercases and with dashes instead of spaces. Example

   Issue Title: 07 Submit a new Post
   Branch Name: feature/07-submit-a-new-post

5. Make sure the **Create from** is set to **develop**
6. Click on **Create branch**



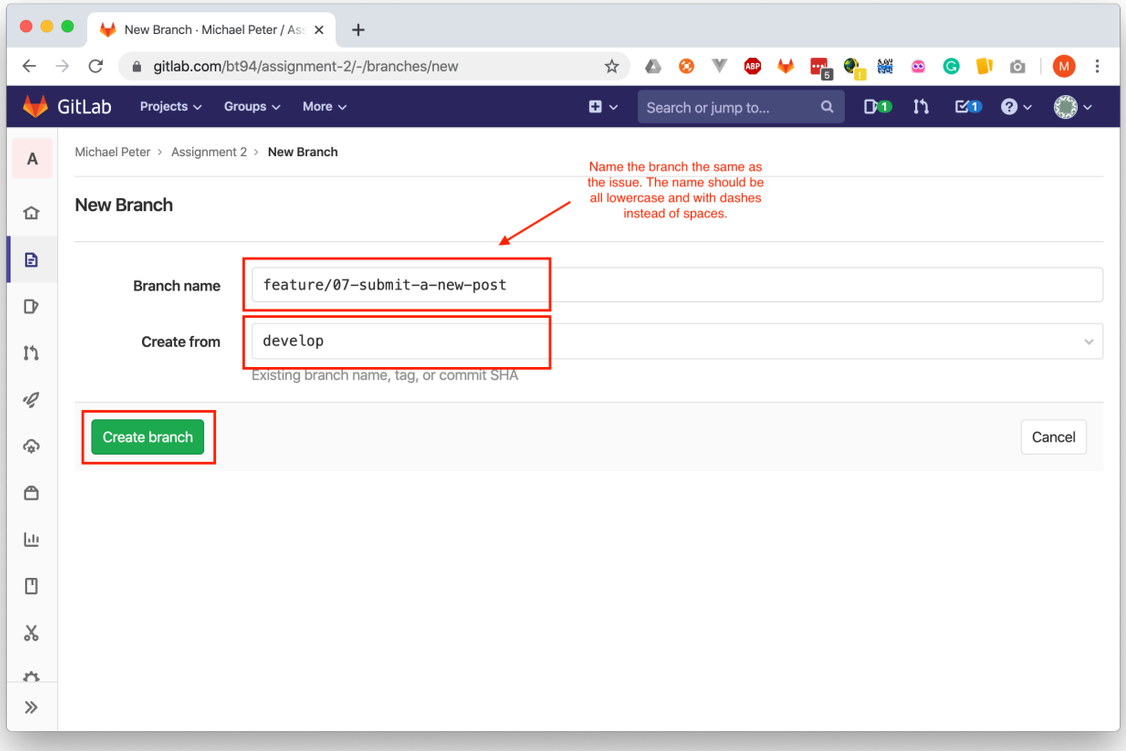
Create a feature branch (2/2)

1. From the side bar on the left select **Repository**
2. On the repository page make sure you are on the **develop** branch
3. Next click the **+** symbol and then **New branch**
4. Name the branch *feature/07-name-of-issue*

   The name after "feature/" should be identical to your issue title, except all lowercases and with dashes instead of spaces. Example

   Issue Title: 07 Submit a new Post
   Branch Name: feature/07-submit-a-new-post

5. Make sure the **Create from** is set to **develop**
6. Click on **Create branch**

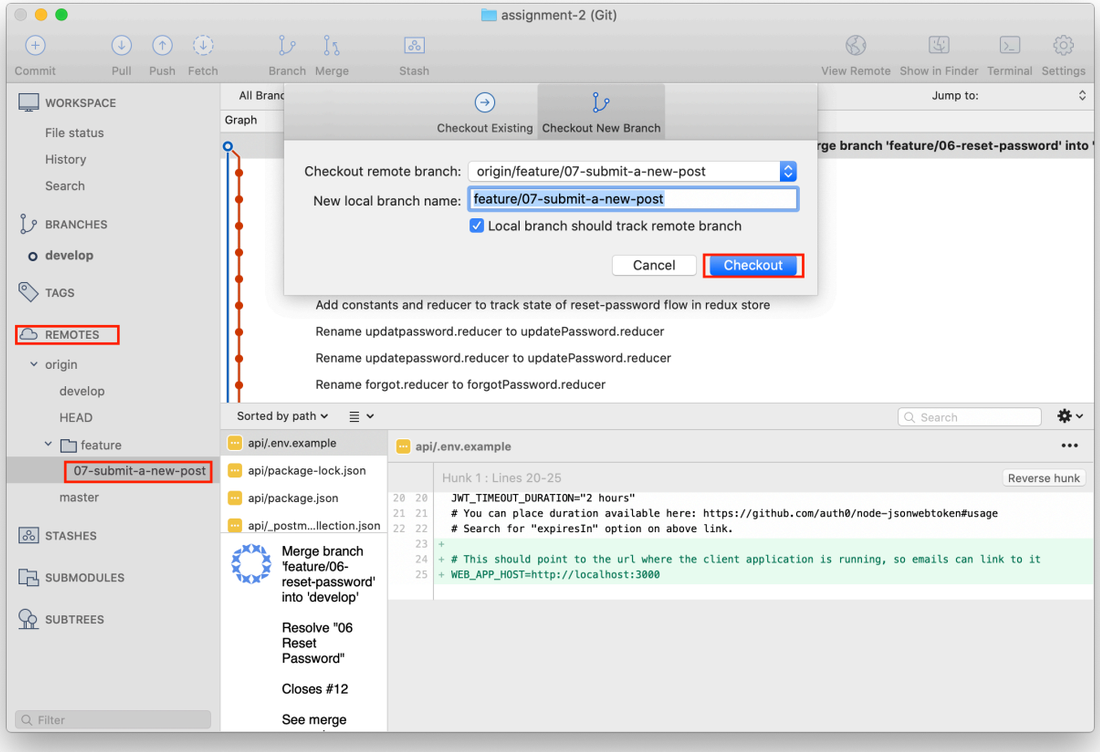


Checkout the feature branch in Sourcetree (1/2)

1. Open your repository in Sourcetree
2. On the left side expand **Remotes** and the **feature**
3. Right-Click on your feature branch and select **Checkout**
4. Make sure that it is the branch is active: It should now be **bold** under the **Branches** section in your sidebar.

Not on a screenshot:

1. Implement the user story 🎉
2. Commit often
3. Write good commit messages
4. Push from time to time

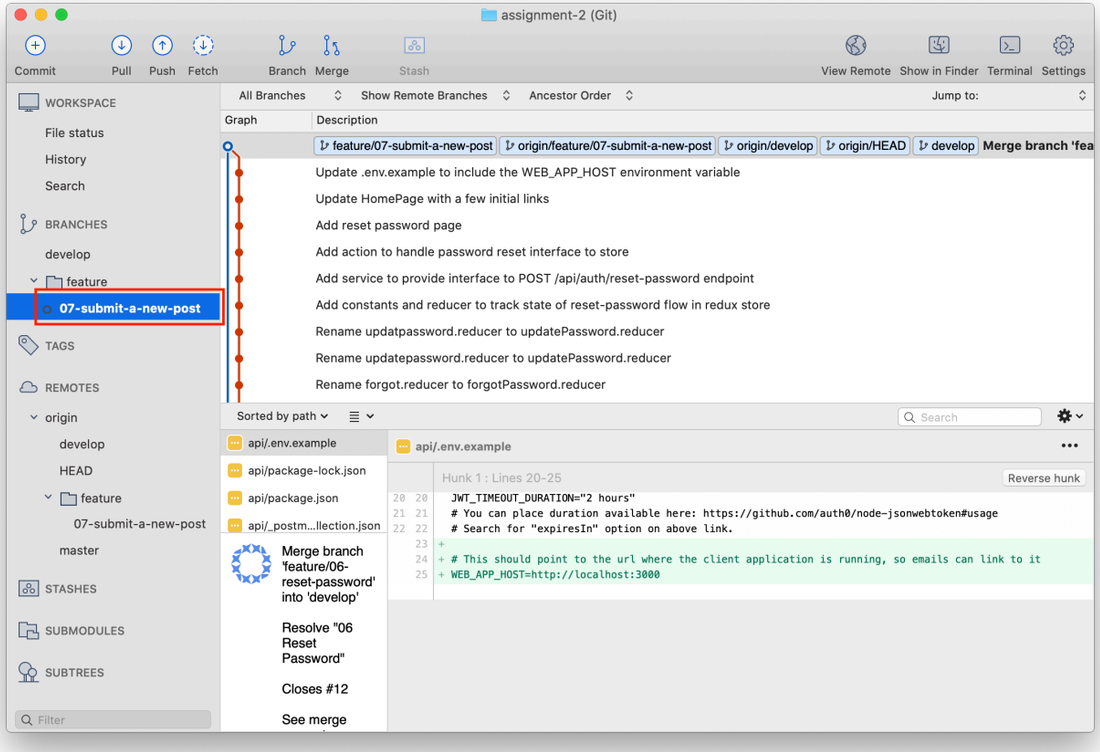


Checkout the feature branch in Sourcetree (2/2)

1. Open your repository in Sourcetree
2. On the left side expand **Remotes** and the **feature**
3. Right-Click on your feature branch and select **Checkout**
4. Make sure that it is the branch is active: It should now be **bold** under the **Branches** section in your sidebar.

Not on a screenshot:

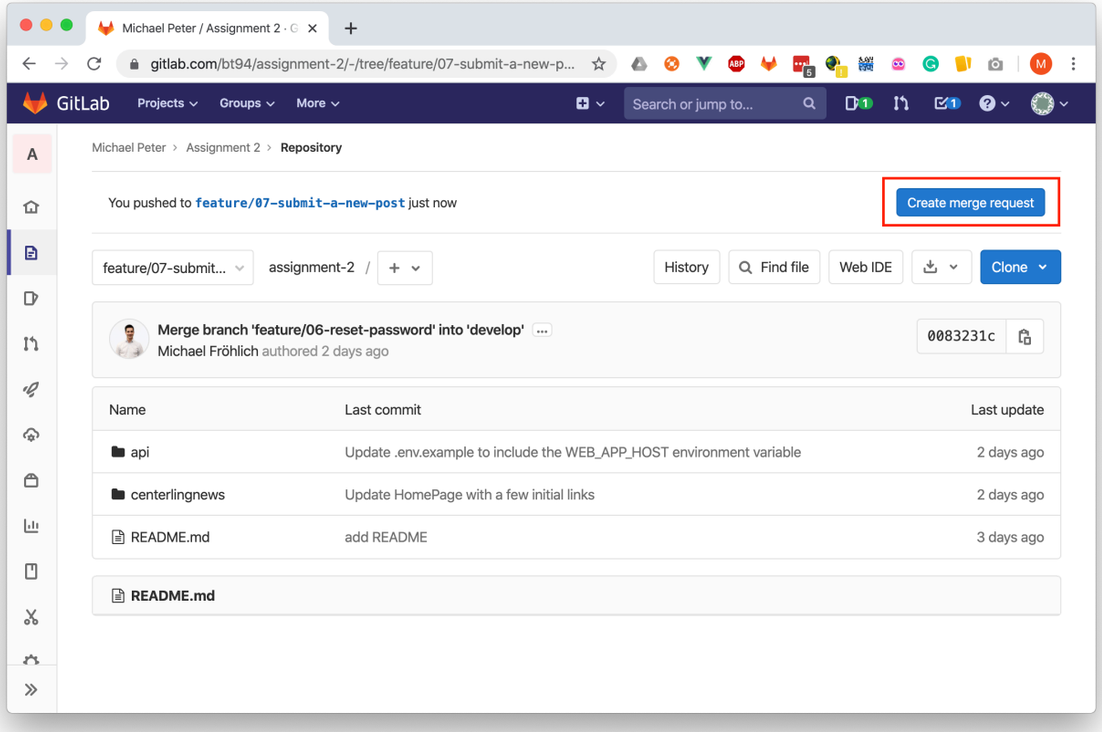
1. Implement the user story 🎉
2. Commit often
3. Write good commit messages
4. Push from time to time



Create a merge request (1/3)

Once you have finished implementing the user story (DoD).

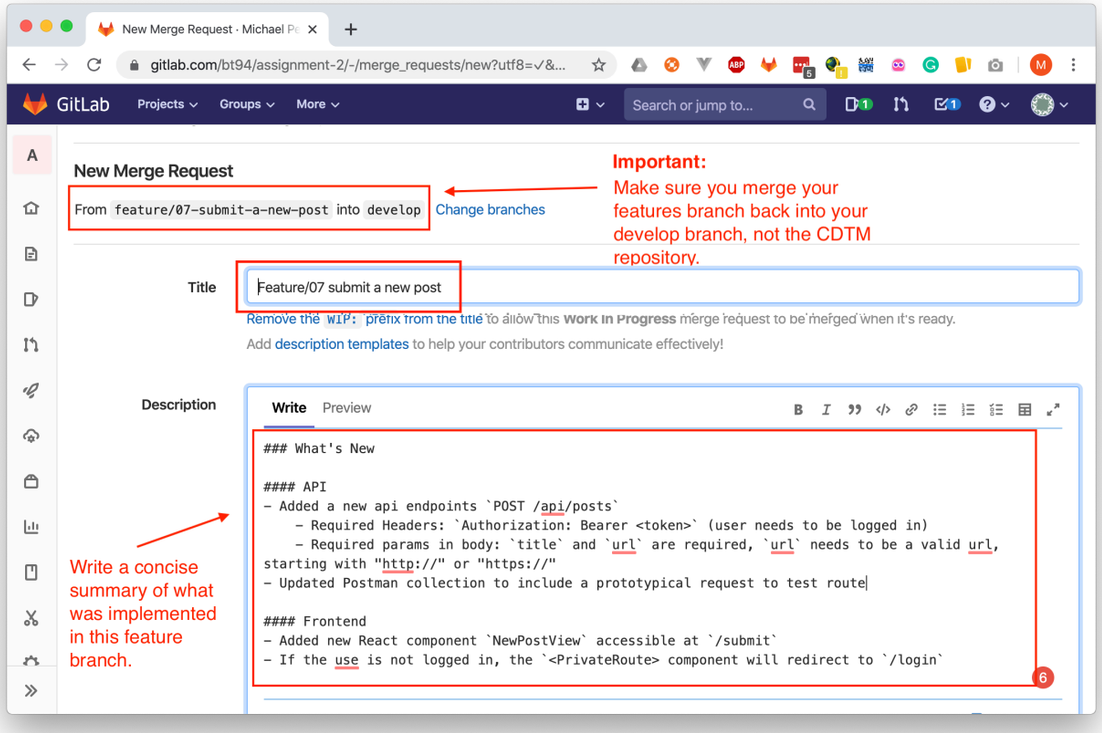
1. Make sure you have pushed your changes
2. Go to your repository on gitlab.com
3. Click on **Create merge request**
4. Make sure that the merge request is from your repository's feature branch into your repository's develop branch
5. Give the merge request a concise and meaning full title and describe all significant changes implemented on this feature branch.
6. (Not for develop → master) Make sure the **Delete source branch when merge request is accepted option is ticked** and press **Create Merge Request**
7. On the next screen you can review your changes and select an assignee.
   - during **Assignment II**, merge the request right away, before you start the next user story.
   - during your **Final Project**, select your partner as assignee and wait for them to code review and approve your changes



Create a merge request (2/3)

Once you have finished implementing the user story (DoD).

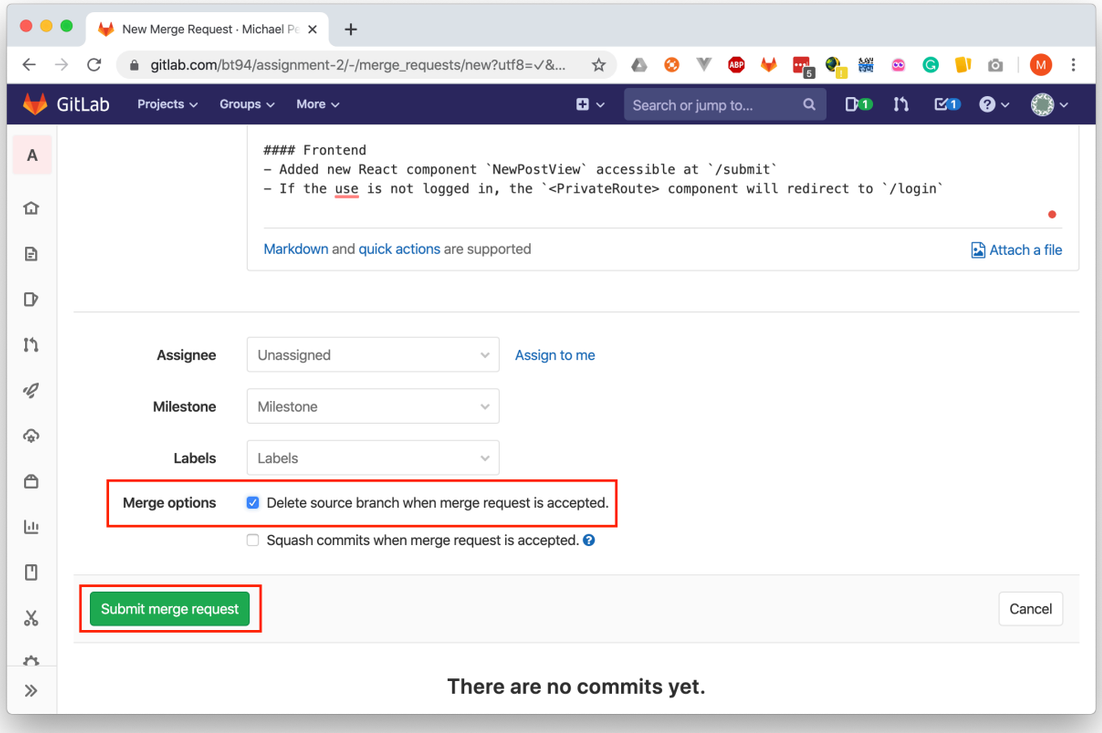
1. Make sure you have pushed your changes
2. Go to your repository on gitlab.com
3. Click on **Create merge request**
4. Make sure that the merge request is from your repository's feature branch into your repository's develop branch
5. Give the merge request a concise and meaning full title and describe all significant changes implemented on this feature branch.
6. (Not for develop → master) Make sure the **Delete source branch when merge request is accepted option is ticked** and press **Create Merge Request**
7. On the next screen you can review your changes and select an assignee.
   - during **Assignment II**, merge the request right away, before you start the next user story.
   - during your **Final Project**, select your partner as assignee and wait for them to code review and approve your changes



Create a merge request (3/3)

Once you have finished implementing the user story (DoD).

1. Make sure you have pushed your changes
2. Go to your repository on gitlab.com
3. Click on **Create merge request**
4. Make sure that the merge request is from your repository's feature branch into your repository's develop branch
5. Give the merge request a concise and meaning full title and describe all significant changes implemented on this feature branch.
6. (Not for develop → master) Make sure the **Delete source branch when merge request is accepted option is ticked** and press **Create Merge Request**
7. On the next screen you can review your changes and select an assignee.
   - during **Assignment II**, merge the request right away, before you start the next user story.
   - during your **Final Project**, select your partner as assignee and wait for them to code review and approve your changes



CONTACT

Michael Froehlich
Management Team
froehlich@cdtm.de

CDTM – CENTER FOR DIGITAL TECHNOLOGY AND MANAGEMENT

A joint institution of education, research and entrepreneurship of
Ludwig-Maximilians-Universität München and Technische Universität München

Visitor Address – Marsstr. 20-22, 80335 Munich, Germany
Postal Address – Arcisstr. 21, 80290 Munich, Germany