Hochschule Fulda
University of Applied Sciences

Development of a Change Data Capture (CDC) tool Medical care register

Name: Muhammad Hassan

Matrikelnummer: 1503186

Role: Backend developer & DevOps

Course: Software Development Project (AI5094)

Presented to: Prof. Michael Jahn

Project Overview

Hochschule Fulda
University of Applied Sciences

This project involved the development of a functioning Clinical Data Collection (CDC) application focused on gathering patient data for diabetic foot syndrome in the form of a digital register.

The goal was to design and implement a software solution that supports structured, secure, and efficient data entry, management, and retrieval to assist healthcare professionals in monitoring and research.

The project was developed collaboratively, with each team member contributing to a specific functional area (backend, frontend, DevOps, etc.), and documented through regular reports and presentations.

Technology & Tools

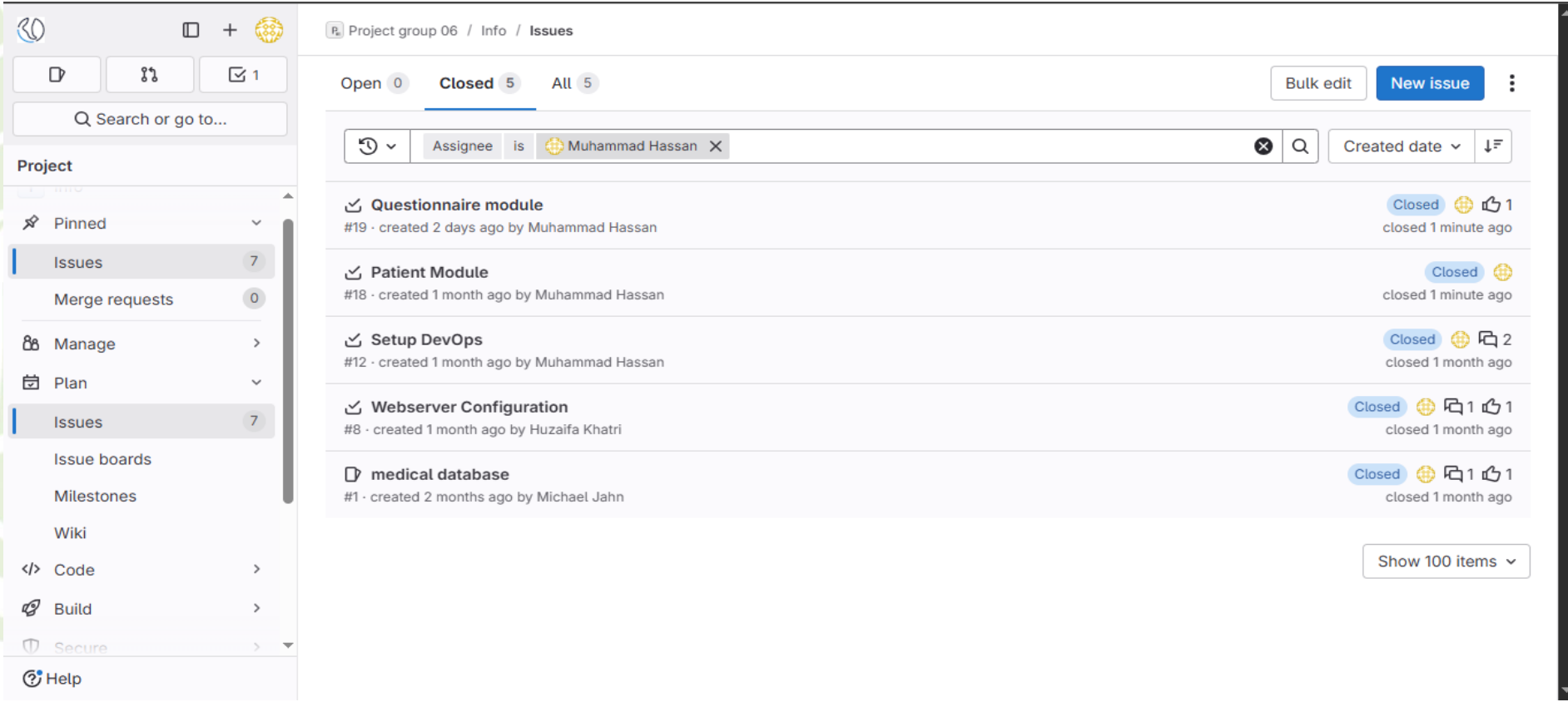
- **Framework:** Express.js (Runtime engine : Node.js)
- **ORM:** Knex.js with Objection.js
- **Database:** MariaDB
- **Validation:** Joi schema validators
- **Error Handling:** Custom success/error response classes
- **Logging:** Pino Asynchronous Logger

My Role in the Project

- Implementation of Questionnaire module
- Gitlab Repository management
- Backend boiler code setup
- Backend DevOps

Agile project management

Hochschule Fulda
University of Applied Sciences



Backend module – Questionnaire API

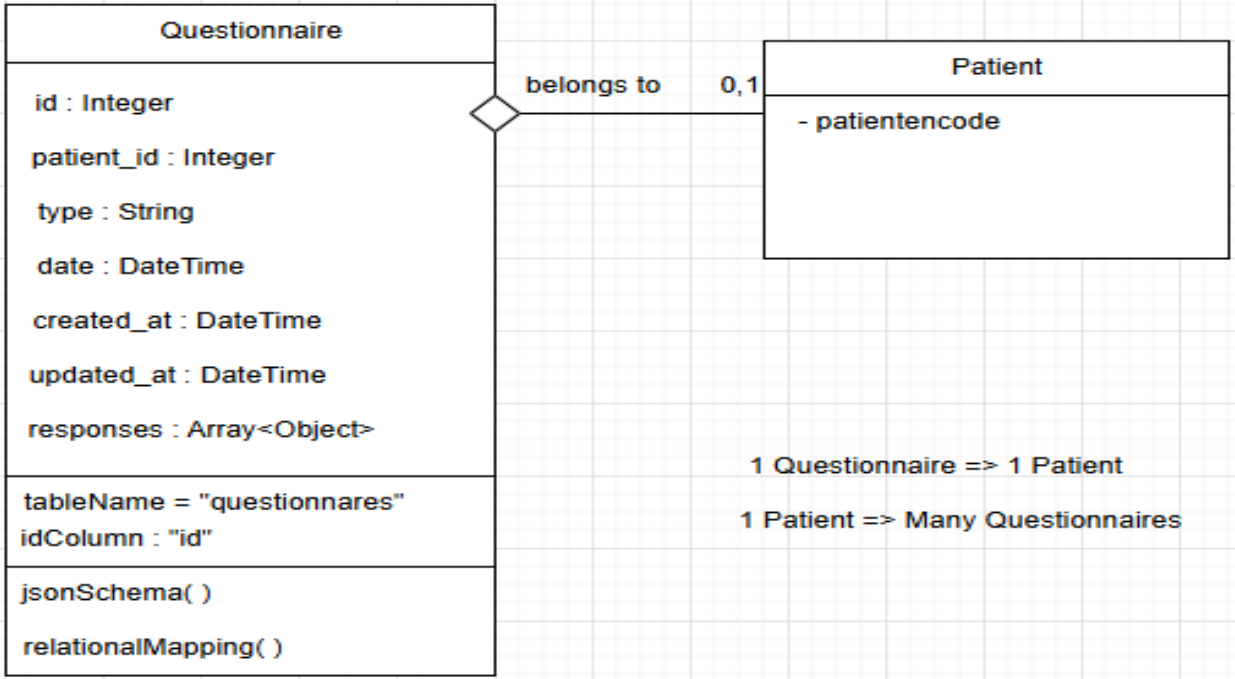
Hochschule Fulda
University of Applied Sciences

Purpose of the module: This module handles the creation, retrieval, and updating of patient questionnaires related to diabetic foot syndrome. It ensures secure, validated, and structured data entry in the CDC register.

Key Features:

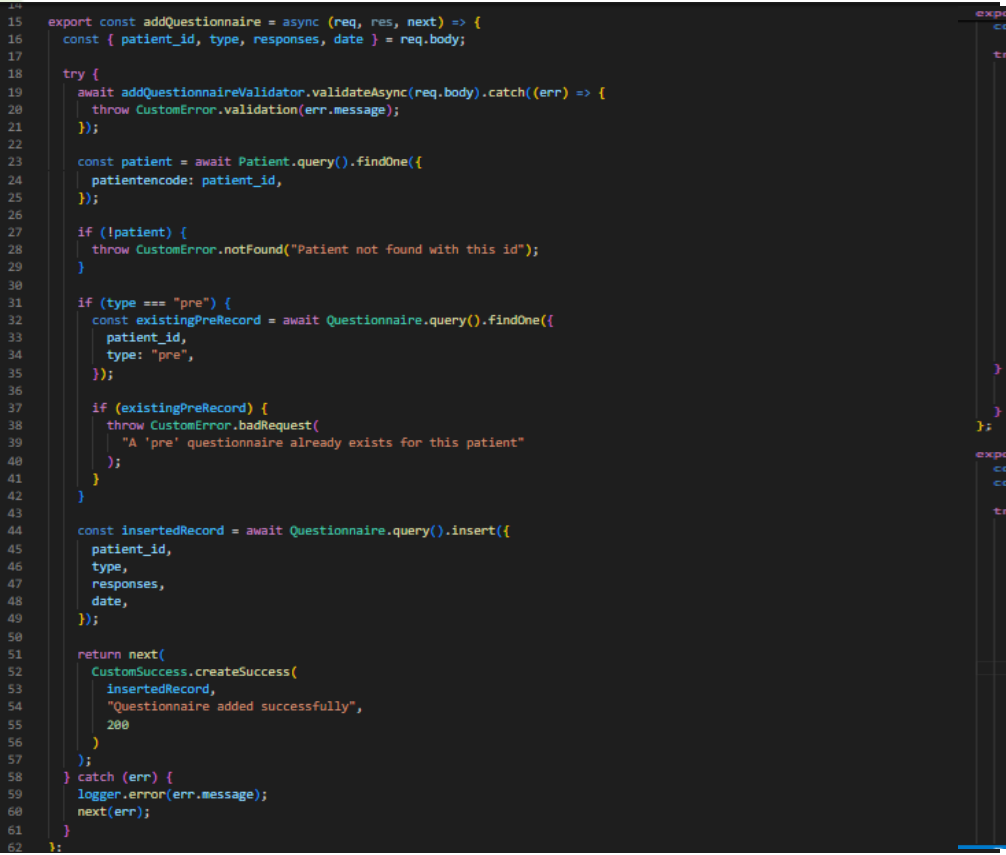
- **Create Questionnaire**
  - Ensures *"pre"* type is unique per patient
  - Validates schema before insertion
  - Adds records with timestamps

- **Retrieve Questionnaire by patient ID**
  - Returns all questionnaires linked to a specific patient
- **Update Questionnaire**
  - Validates ownership
  - Update ‘responses’ and ‘updated_at’

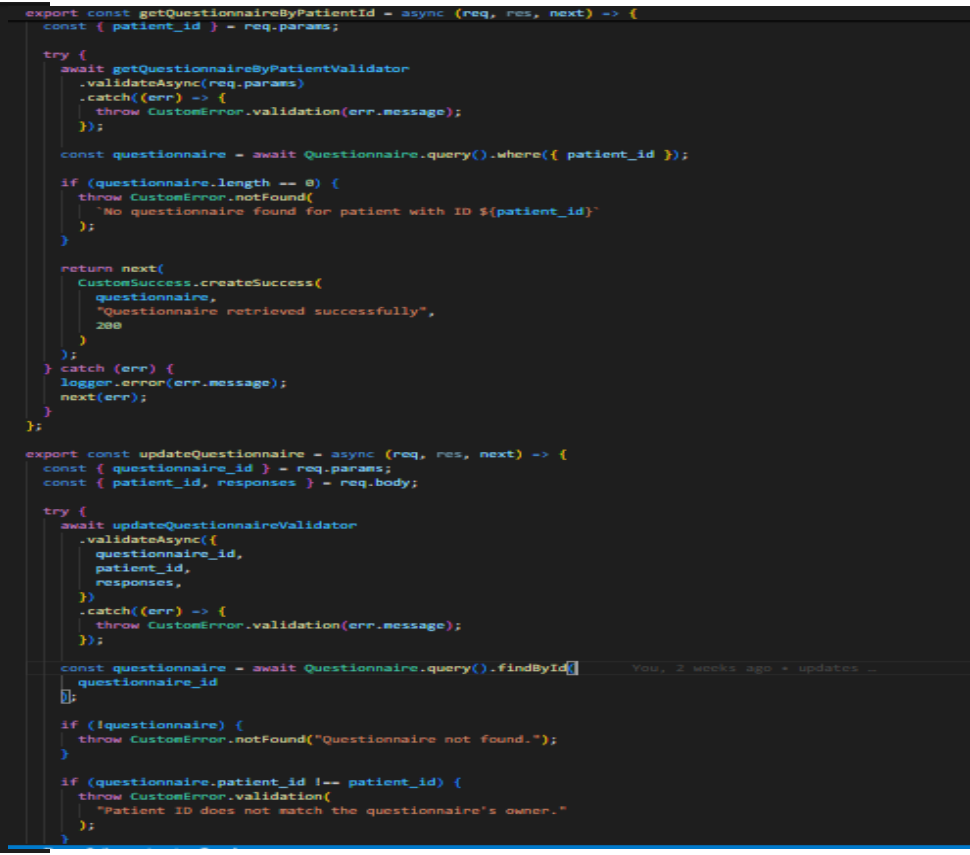


Class Diagram (Questionnaire)

Source code



```
export const addQuestionnaire = async (req, res, next) => {
  const { patient_id, type, responses, date } = req.body;

  try {
    await addQuestionnaireValidator.validateAsync(req.body).catch((err) => {
      throw CustomError.validation(err.message);
    });

    const patient = await Patient.query().findOne({
      patientencode: patient_id,
    });

    if (!patient) {
      throw CustomError.notFound("Patient not found with this id");
    }

    if (type === "pre") {
      const existingPreRecord = await Questionnaire.query().findOne({
        patient_id,
        type: "pre",
      });

      if (existingPreRecord) {
        throw CustomError.badRequest(
          "A 'pre' questionnaire already exists for this patient"
        );
      }
    }

    const insertedRecord = await Questionnaire.query().insert({
      patient_id,
      type,
      responses,
      date,
    });

    return next(
      CustomSuccess.createSuccess(
        insertedRecord,
        "Questionnaire added successfully",
        200
      )
    );
  } catch (err) {
    logger.error(err.message);
    next(err);
  }
};
```



```
export const getQuestionnaireByPatientId = async (req, res, next) => {
  const { patient_id } = req.params;

  try {
    await getQuestionnaireByPatientValidator
      .validateAsync(req.params)
      .catch((err) => {
        throw CustomError.validation(err.message);
      });

    const questionnaire = await Questionnaire.query().where({ patient_id });

    if (questionnaire.length == 0) {
      throw CustomError.notFound(
        `No questionnaire found for patient with ID ${patient_id}`
      );
    }

    return next(
      CustomSuccess.createSuccess(
        questionnaire,
        "Questionnaire retrieved successfully",
        200
      )
    );
  } catch (err) {
    logger.error(err.message);
    next(err);
  }
};

export const updateQuestionnaire = async (req, res, next) => {
  const { questionnaire_id } = req.params;
  const { patient_id, responses } = req.body;

  try {
    await updateQuestionnaireValidator
      .validateAsync({
        questionnaire_id,
        patient_id,
        responses,
      })
      .catch((err) => {
        throw CustomError.validation(err.message);
      });

    const questionnaire = await Questionnaire.query().findById(
      questionnaire_id
    );

    if (!questionnaire) {
      throw CustomError.notFound("Questionnaire not found.");
    }

    if (questionnaire.patient_id !== patient_id) {
      throw CustomError.validation(
        "Patient ID does not match the questionnaire's owner."
      );
    }
```

DevOps Contributions

Hochschule Fulda
University of Applied Sciences

- **CI/CD Pipeline Setup (GitLab):**
  - Implemented .gitlab-ci.yml for defining pipeline stages
  - Configured a self-hosted GitLab runner to execute the pipeline on the target server
  - Set up Git remote URL with embedded **Personal Access Token** for secure repository access
  - Set up secure SSH access:
    - Generated SSH keys and configured permissions
    - Added public key to authorized_keys
    - Stored private key in GitLab CI/CD variables
- **Deployment Automation:**
  - Pipeline triggers post deployment script that pulls the latest commit and uses PM2 to start and manage the server

.gitlab-ci.yml | Questionnaire.js | QuestionnaireRouter.js | QuestionnaireValidator.js | QuestionnaireController.js

.gitlab-ci.yml

pate

You, last month | 1 author (You)

```yaml
stages:
  - deploy

deploy_dev:
  stage: deploy
  tags:
    - ci-cd-job-runner
  only:
    - dev
  before_script:      You, last month • ci config pushed
    - echo "== BEFORE SCRIPT START =="
    - which ssh-agent || ( apt-get update -y && apt-get install openssh-client -y )
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh

  script:
    - ssh -o StrictHostKeyChecking=no pg06-web@pg06.regifor.de 'bash /home/deploy_scripts/deploy_dev.sh'
```

Pipeline configuration

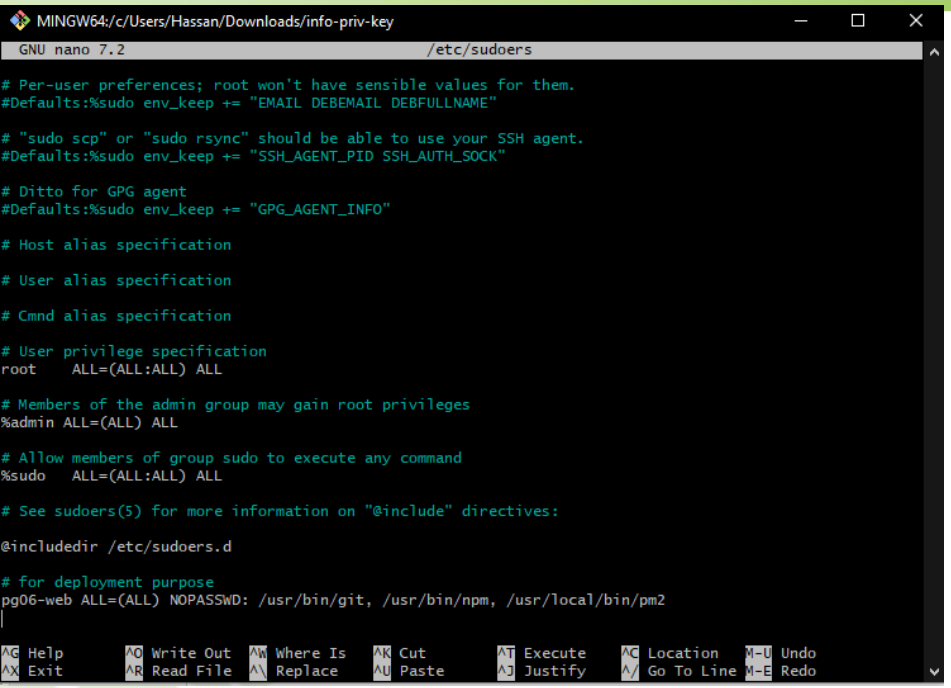
MINGW64:/c/Users/Hassan/Downloads/info-priv-key

GNU nano 7.2 pg06.regifor.de-le-ssl.conf

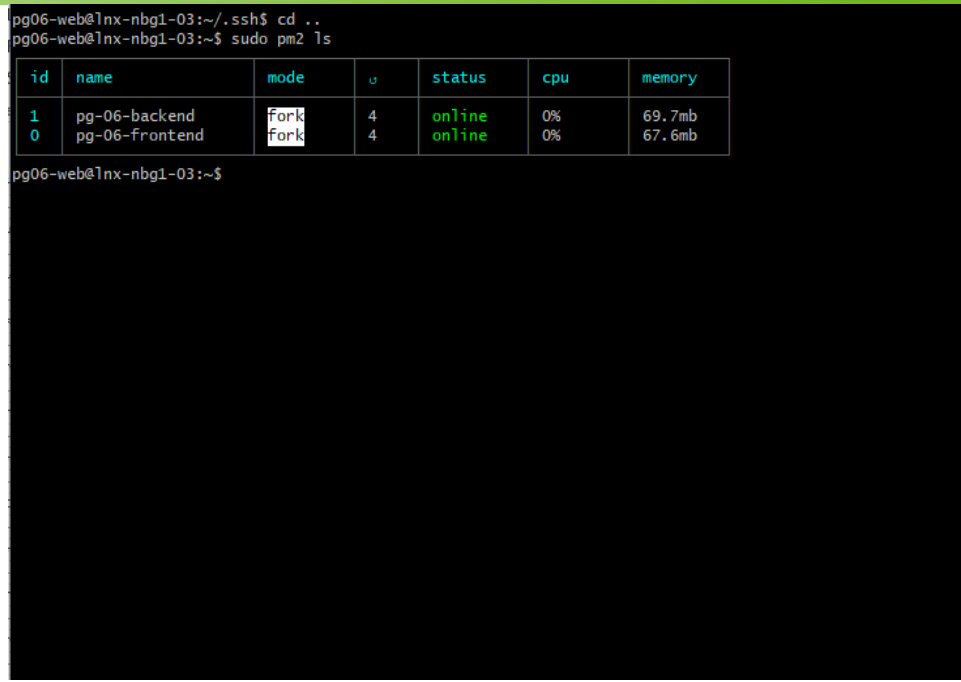
```
<IfModule mod_ssl.c>
        <VirtualHost *:443>
                ServerAdmin administrator@regifor.de

                ServerName pg06.regifor.de
                ServerAlias www.pg06.regifor.de

                DocumentRoot /var/www/pg06.regifor.de

                ErrorLog ${APACHE_LOG_DIR}/pg06.regifor.de-error.log
                CustomLog ${APACHE_LOG_DIR}/pg06.regifor.de-access.log combined

                ProxyPass "/api/" "http://localhost:8000/api/"
                ProxyPassReverse "/api/" "http://localhost:8000/api/"

                ProxyPass "/" "http://localhost:3000/"
                ProxyPassReverse "/" "http://localhost:3000/"

                SSLCertificateFile /etc/letsencrypt/live/pg06.regifor.de/fullchain.pem
                SSLCertificateKeyFile /etc/letsencrypt/live/pg06.regifor.de/privkey.pem
                Include /etc/letsencrypt/options-ssl-apache.conf
        </VirtualHost>
</IfModule>
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo

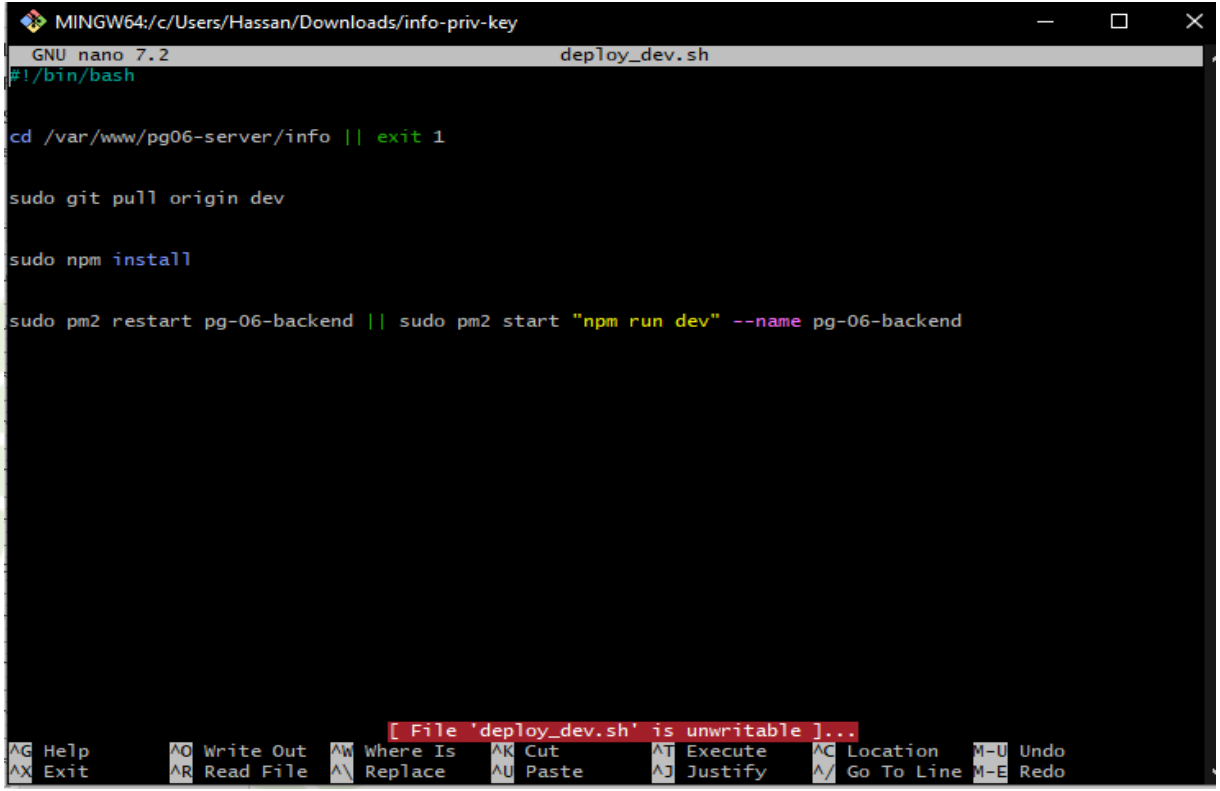
Setting up Reverse proxy using Apache2



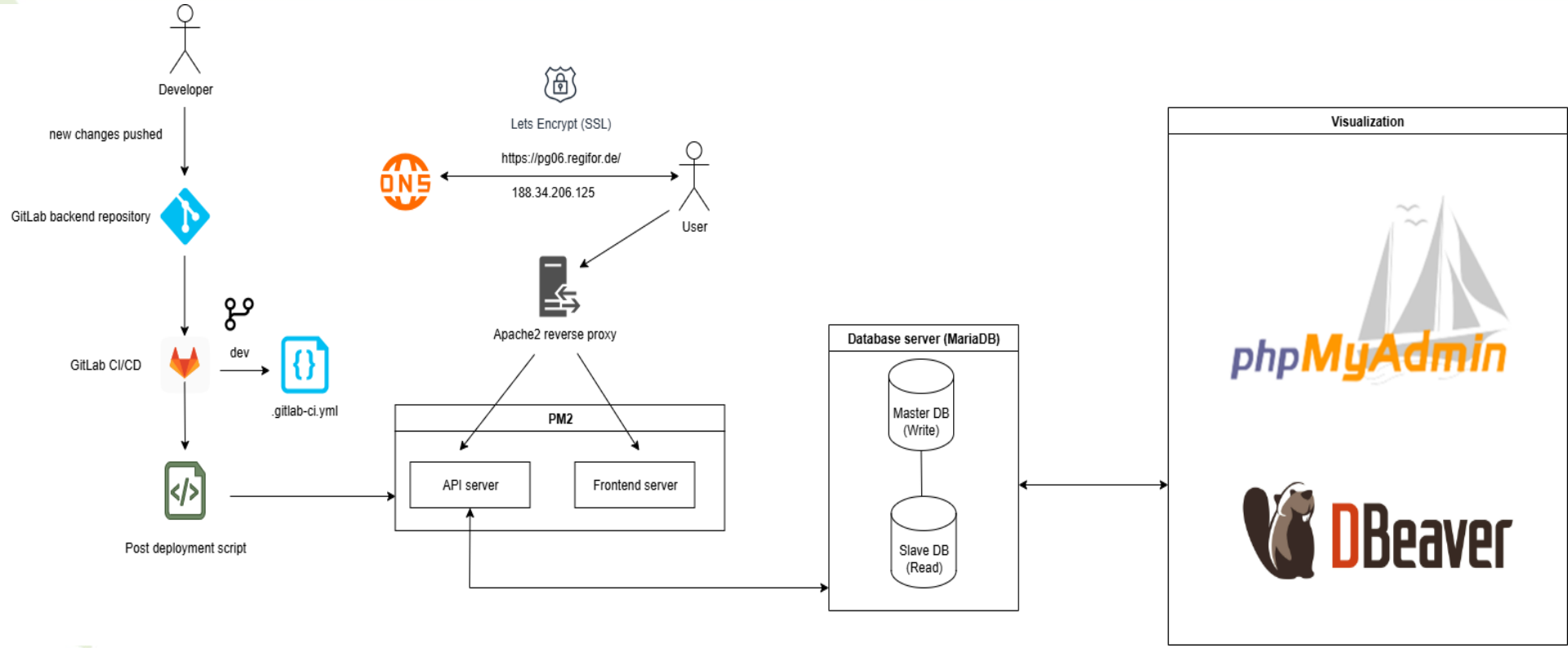
Configure PM2, npm and git to work with escalated privileges



Instances running with PM2



Post-deployment script



High Level Architecture Diagram

Thank you.

Questions?