|  |  |  |
| --- | --- | --- |
|  | | |
|  | **CMPS 350 Project Phase 2 – Conference Management System (ConfPlus)**  **(15% of the course grade)** | |
| **Group Id:** | | G |
| **Group Members:** | | Mohamed Dhia Abdaoui (202005886)  Ridhwan Athaullah (202005171)  Mohamad Allaham (202005872)  Saoud Ali Al Khelaifi (201908762)  **Emails:**  [**ma2005886@student.qu.edu.qa**](mailto:ma2005886@student.qu.edu.qa)**;**  [**ra2005171@student.qu.edu.qa**](mailto:ra2005171@student.qu.edu.qa)**;**  [**ma2005872@student.qu.edu.qa**](mailto:ma2005872@student.qu.edu.qa)**;**  [**sa1908762@student.qu.edu.qa**](mailto:sa1908762@student.qu.edu.qa)**;** |

**Grading Rubric - In the Functionality column please specify either: *Working (completed x%)*, *Not Working (completed x%)* or *Not done or Not Applicable*.**

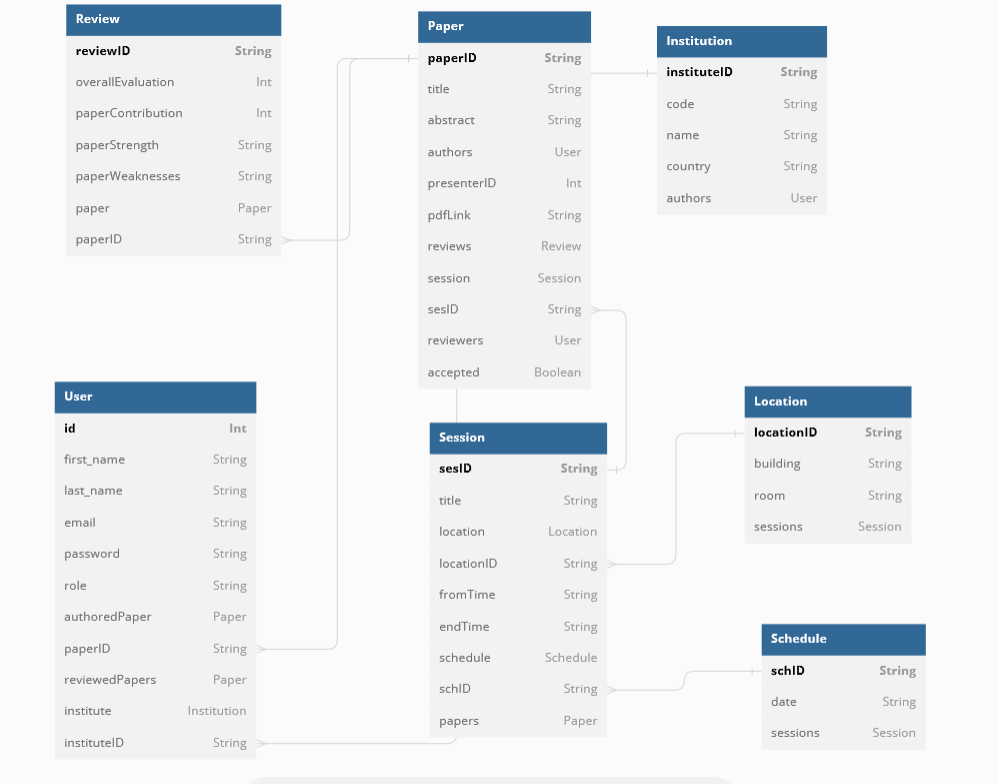
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Weight** | **Functionality**\* | **Quality of the implementation** | **Grade** |
| **Improvement over the first phase:**  *Depending on the implantation status of the previous phase, the following might apply:*   * *Completing missing functionalities;* * *Improving the design and implementation of paper submission: paper status, etc.* * *Improving the design and implementation of paper review: distinction between reviewed papers and papers to review, etc.* * *Various filtering possibilities for the conf schedule* * *Correct interpretation of session* * *Clarity of the various UIs.*   *By default, if no improvement is made, the student will have the same grade of previous phase for this category.*  *Everything you improve will add up to your previous grade that will be used as a baseline.* | 25 |  |  |  |
| Design and implement the Data Model.  Clarity of data entities, their attributes and relations (in Prisma and the conceptual model (the diagram)) | 10 |  |  |  |
| Init DB: populate the database with the data from the json files. | 5 |  |  |  |
| Repository Implementation to read/write data from the database | 10 |  |  |  |
| Database:   * The design and implementation of the statistics page * All other use-cases use the database, not JSON files or local storage. * All queries function correctly. | 40 |  |  |  |
| **Design and Testing Documentation**  **\* Design documentation:**  - 3 key lessons learned from Phase 1.  - Data Model diagram.  - UI Design table  - Data caching table  **\* Testing documentation:** with evidence of working implementation using snapshots illustrating the results of your solution testing (you must use the provided template).  \* **Discussion of the project contribution** of each team member [-10pts if not done] | 10 |  |  |  |
| **Total** | 100 |  |  |  |
| Bonus - successful deployment of the app and the Database to a cloud hosting service such as <https://vercel.com/> - successful implementation of use authentication. | 5 |  |  |  |
| Bonus- authentication through another service provider e.g. Google, Github, etc. | 5 |  |  |  |
| Copying and/or plagiarism or not being able to explain or answer questions about the implementation. | 0 |  |  |  |

# Application Design

# Improvement over the first phase

The only improvement we had was the clarity of some UI’s

# Data Model diagram



# Database population

We first created the json folders and added data in them.

A picture containing text, screenshot, font, design

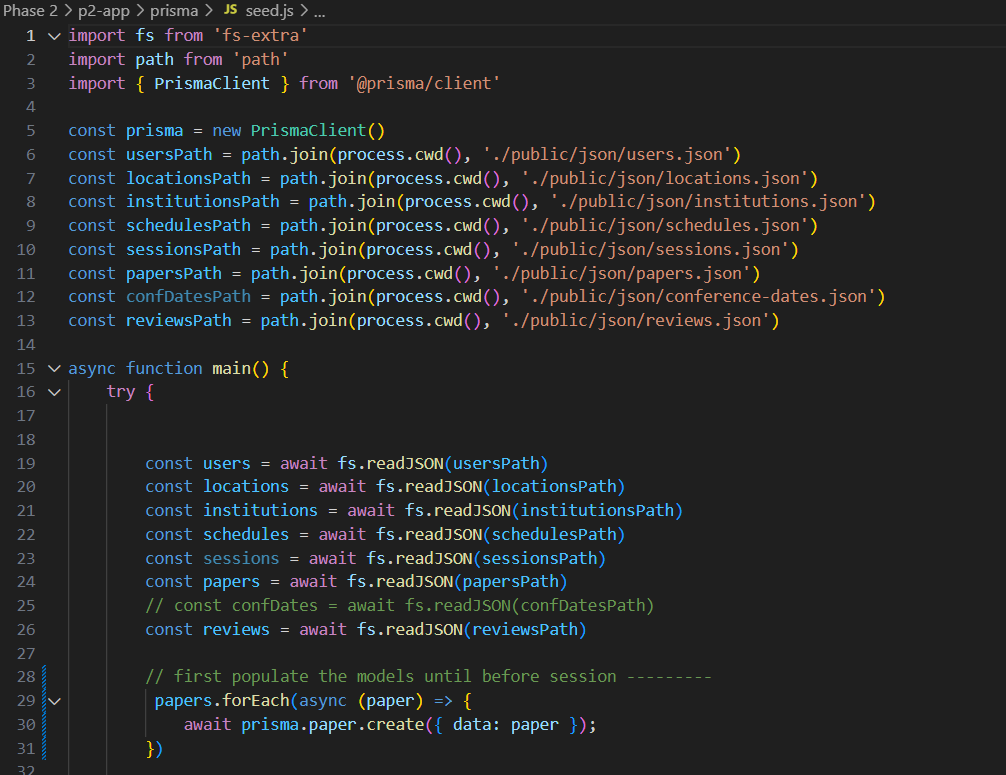
Description automatically generated A screen shot of a computer

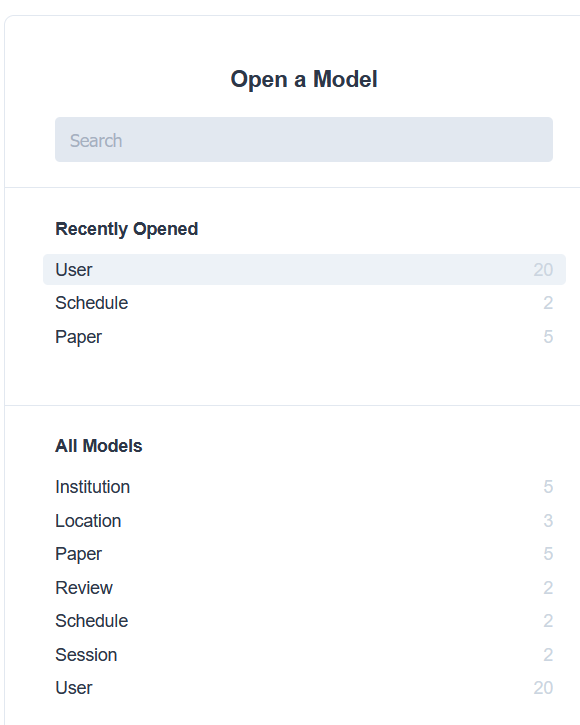
Description automatically generated with medium confidence

We then created the prisma model (this screenshot is an example)



We then created the seed.js and gave the path to the json folders readed each of them and created them in the database.





# Database



We have updated our application and used these commands (npx create-next-app@13.2 my-project, npm install prisma -g, prisma init and npx prisma generate) in the terminal to create a project with next.js and install prisma and intiliaze it.

**Prisma queries**

const locations= await prisma.location.findMany();

return await prisma.location.findUnique({

            where: {

              locationID: locationID,

            },

export const addReview = async(review)=> await prisma.review.create({data:review});

export const getReview= async(paperID)=> await prisma.review.findMany({where:{paperID:paperID}});

const addedPaper= await prisma.paper.create({data:paper});

const review={

    paperID:addedPaper.paperID,

    reviewerId:4

}

console.log("added Paper: ",addedPaper);

const addedReview= await prisma.review.create({data:review})

console.log("added review: ",addedReview);

 return paper

export const getPapers = async()=> await prisma.paper.findMany();

export const getPaper = async(paperID)=> await prisma.paper.findUnique({where:{paperID},include:{reviews:true}});

export const getUsers= async ()=> await prisma.user.findMany();

export const getUser= async (id)=> await prisma.user.findMany({where:{id}});

export const getAuthor= async (id)=> await prisma.author.findUnique({where:{id:+id},include:{authoredPaper:true}});

export const getAuthors= async ()=> await prisma.author.findMany({include:{authoredPaper:true}});

export const getReviewer= async (id)=> await prisma.reviewer.findUnique({where:{reviewerId:+id}});

export const getReviewers= async ()=> await prisma.reviewer.findMany();

const data = await prisma.review.findFirst({

          where : {

            reviewID

          }

        });

 async updateReview(reviewID, reviewData) {

      try {

        const data = await prisma.review.update({

          where : {

            reviewID

          },

          data :

            reviewData

        });

        console.log("\* Prisma review-repo, update review");

        return data;

async getPapersToReview(reviewerID) {

      try {

        reviewerID = parseInt(reviewerID)

        console.log(reviewerID);

        let data = await prisma.review.findMany({

          where : {

            reviewerId : reviewerID

          },

          select : { reviewID : true ,paper : {include : {authors: true}}}

        });

        // data = data.map(obj => obj.paper);

        console.log("\* Prisma review-repo, get papers to review");

        return data;

async getSchedules(){

      try {

        const schedules = await prisma.schedule.findMany({

          include: {

            sessions:{

              include:{

                location : true

              }

            }

            }

        });

        return schedules;

      } catch (error) {

        console.error('Error retrieving dates:', error);

        throw new Error('Failed to retrieve dates');

      }

    }

    //ADD SCHEDULE:

    async addSchedule(schedule) {

      try {

          const newSchedule = await prisma.schedule.create({

              data: schedule

          })

          return newSchedule

      } catch (error) {

          return { error: error.message }

      }

  }

        //might not need this

        async getAllDates() {

          try {

            const schedules = await prisma.schedule.findMany();

            const dates = schedules.map((schedule) => schedule.date);

            return dates;

          } catch (error) {

            console.error('Error retrieving dates:', error);

            throw new Error('Failed to retrieve dates');

          }

        }

 async countPapers() {

        const totalPapers = await prisma.paper.count();

        return totalPapers;

      }

      async calculateAverageAuthorsPerPaper() {

        const totalPapers = await prisma.paper.count();

        const totalAuthors = await prisma.user.count();

        if (totalPapers === 0) {

          return 0;

        }

        const averageAuthorsPerPaper = totalAuthors / totalPapers;

        return averageAuthorsPerPaper;

      }

      async calculateAveragePresentationsPerSession() {

        const totalSessions = await prisma.session.count();

        const totalPresentations = await prisma.paper.count();

        const presentationsPerSession = totalPresentations/ totalSessions;

        return {

          totalSessions,

          presentationsPerSession,

        };

      }

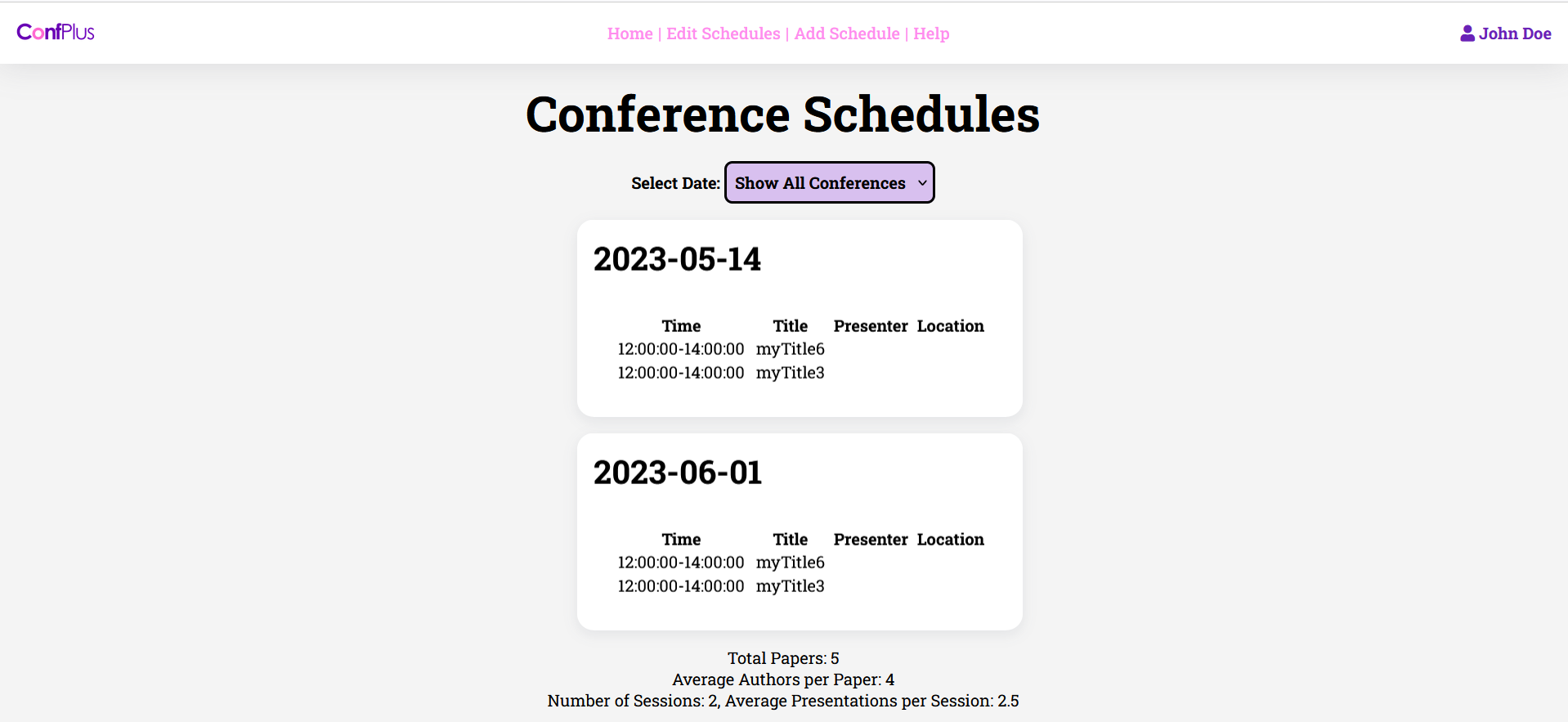
# 3 technical lessons learned from your submitted solution vs. the model solution

# Testing

# Custom Login

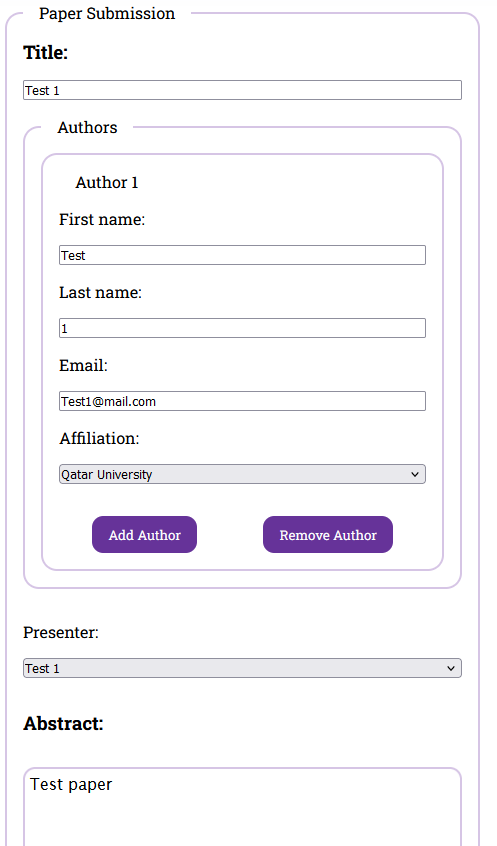
A screenshot of a computer

Description automatically generated with medium confidence

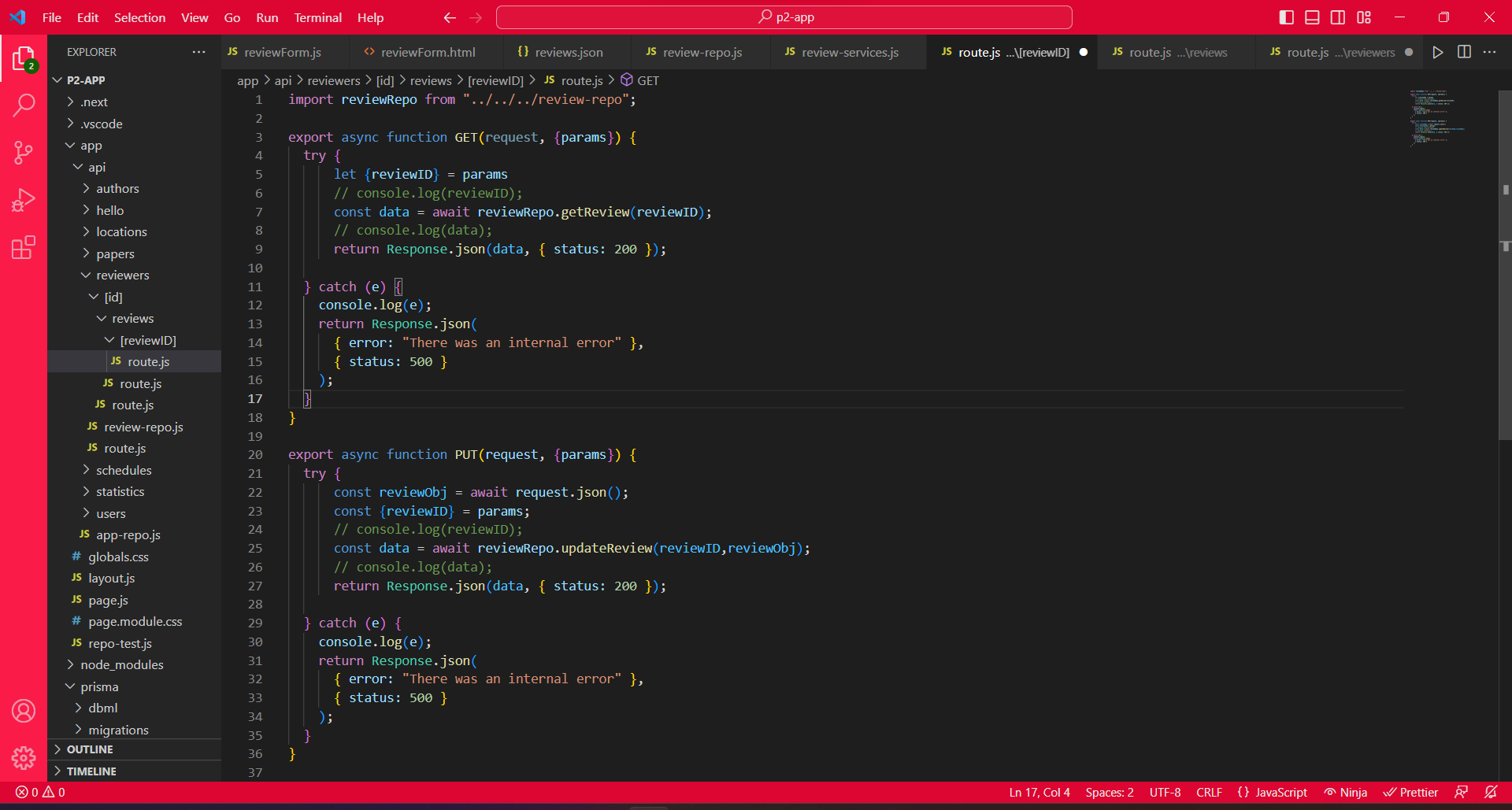
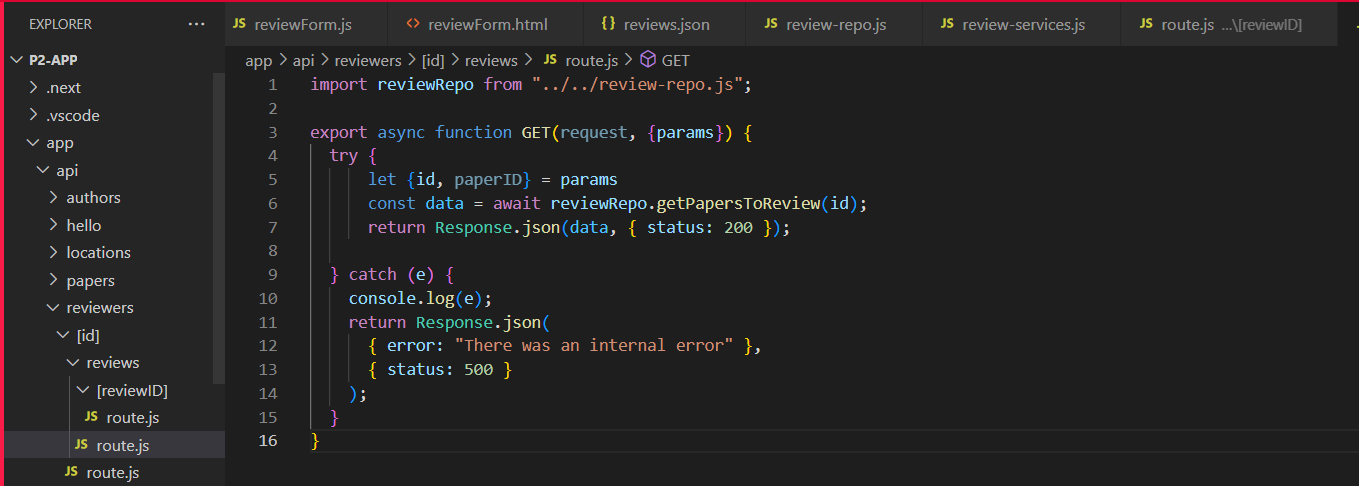


# Login using 2 Authentication Providers

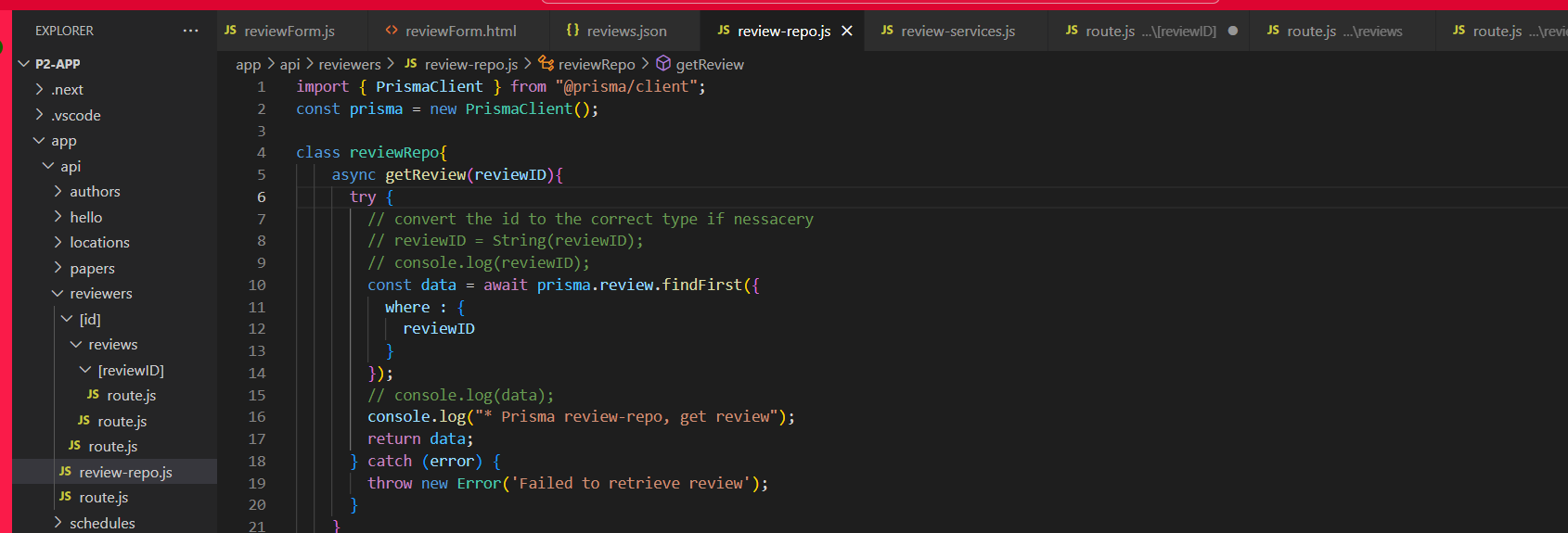
# Submit paper



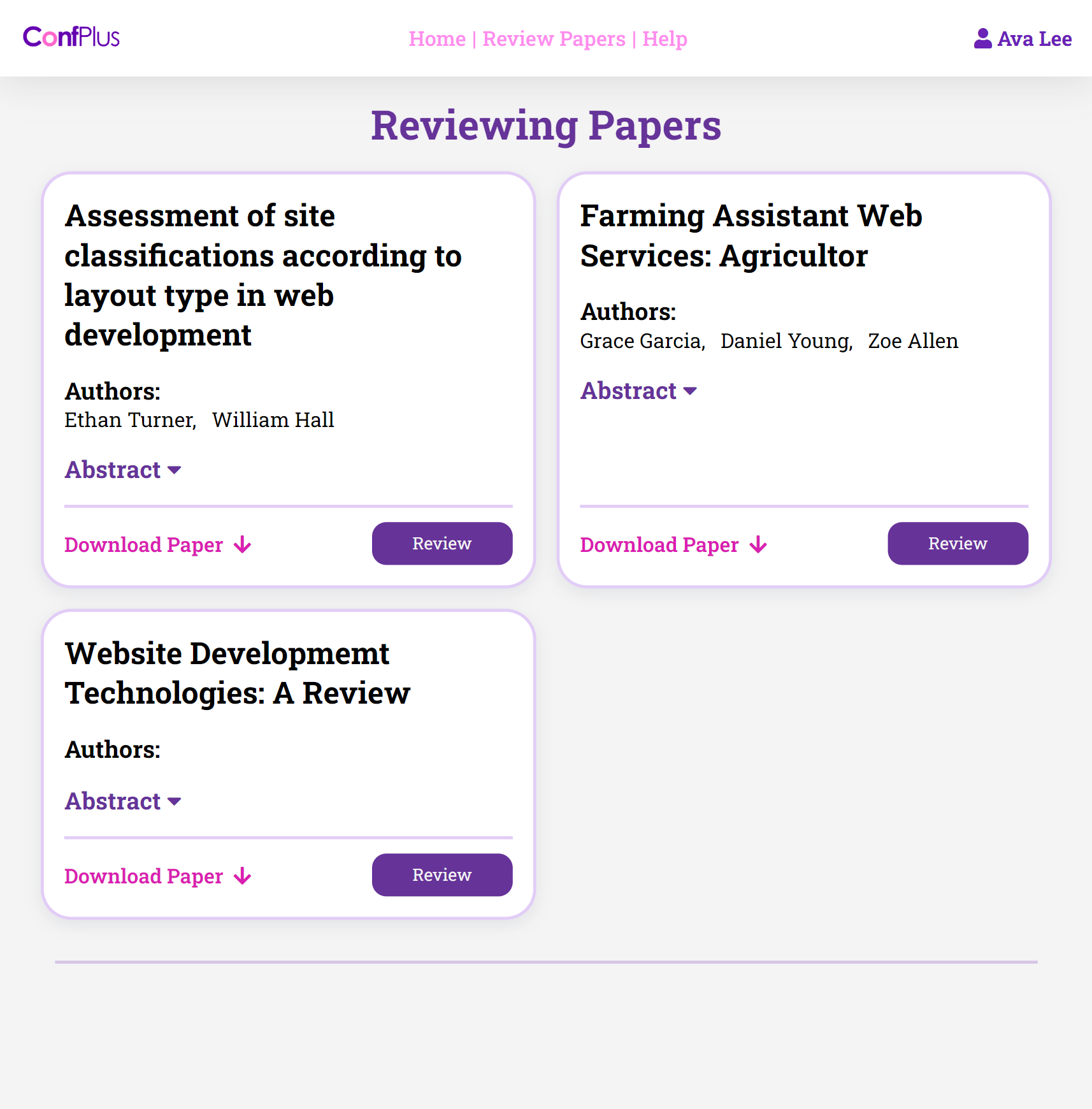
# Review paper

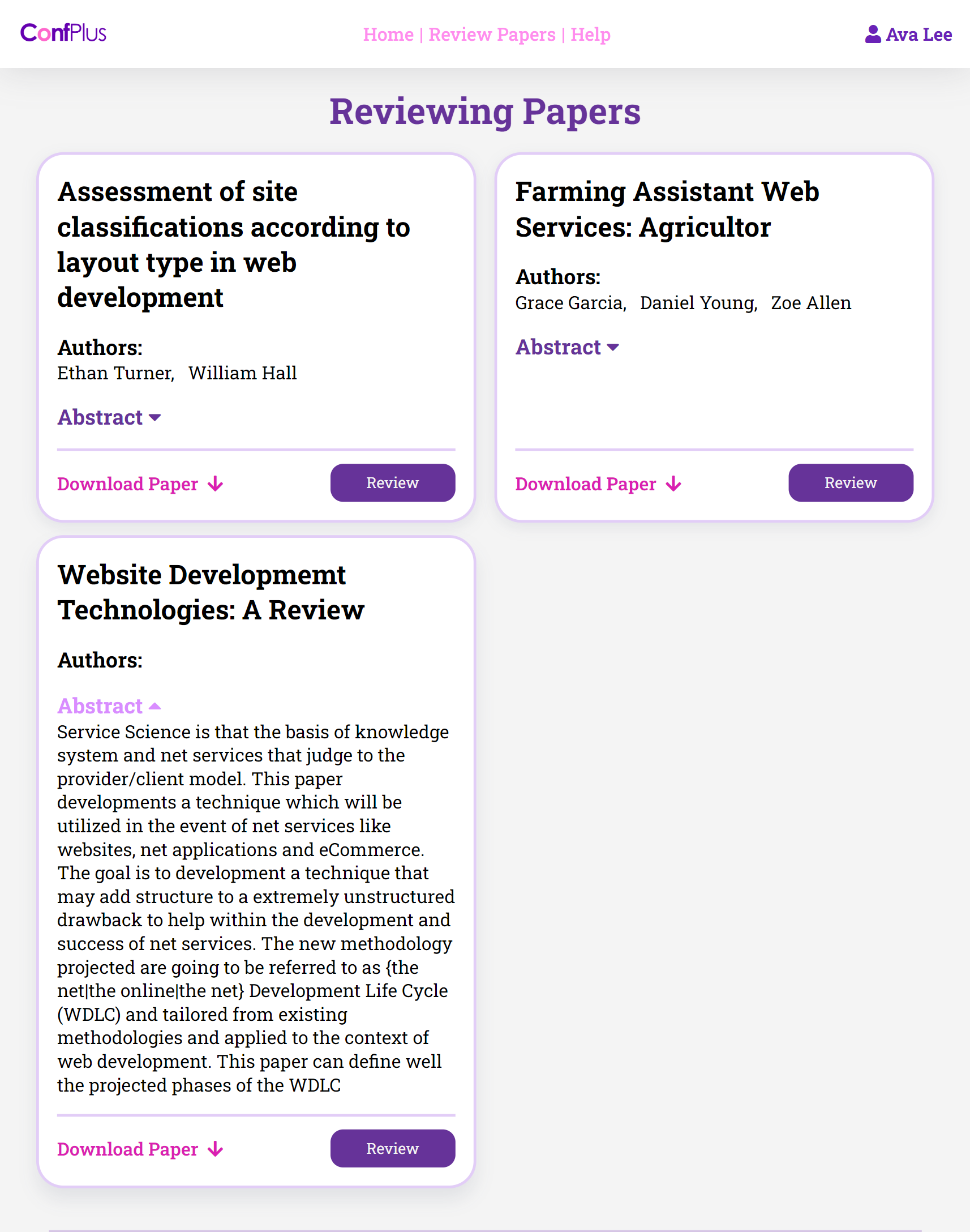


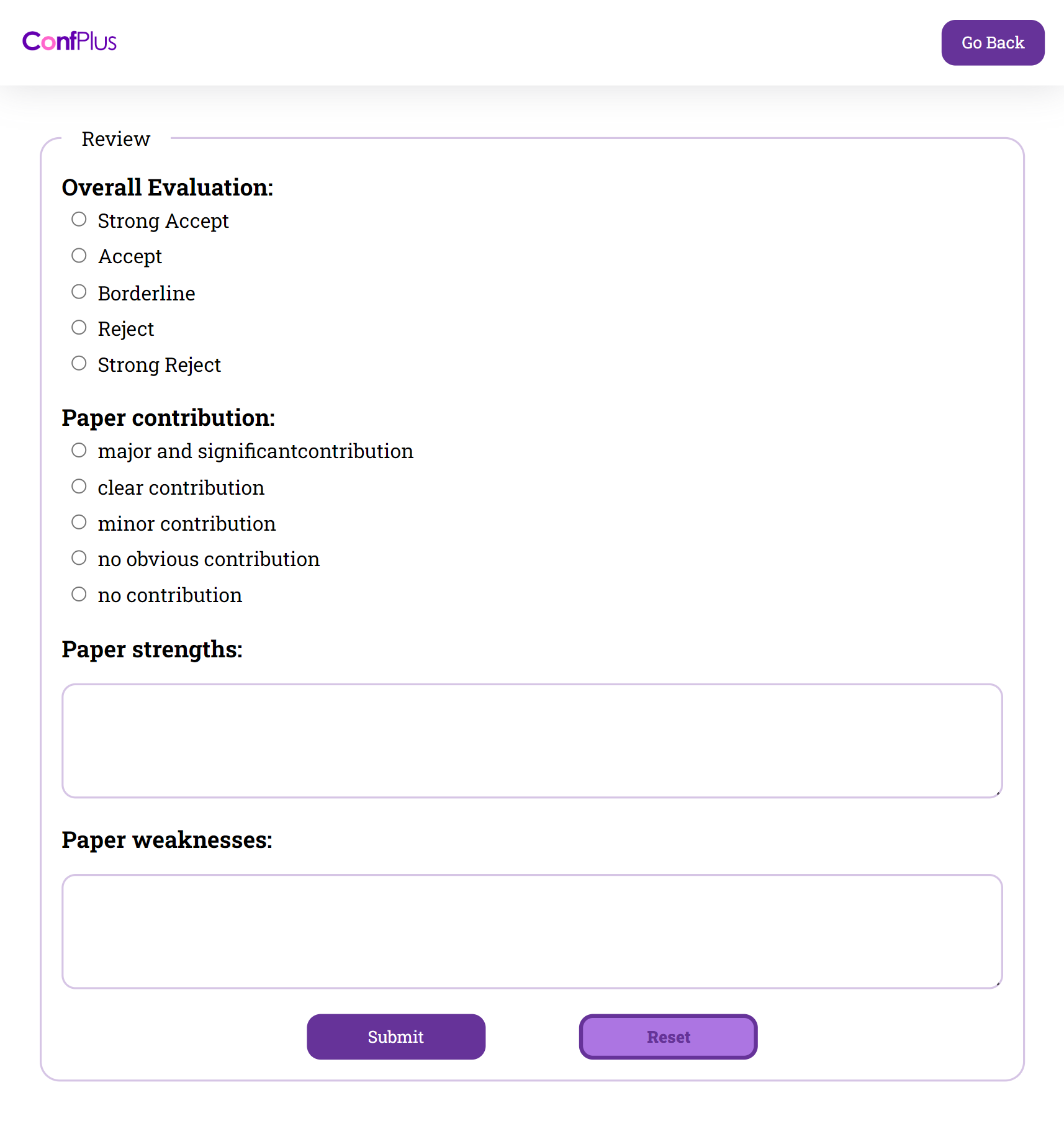
# 

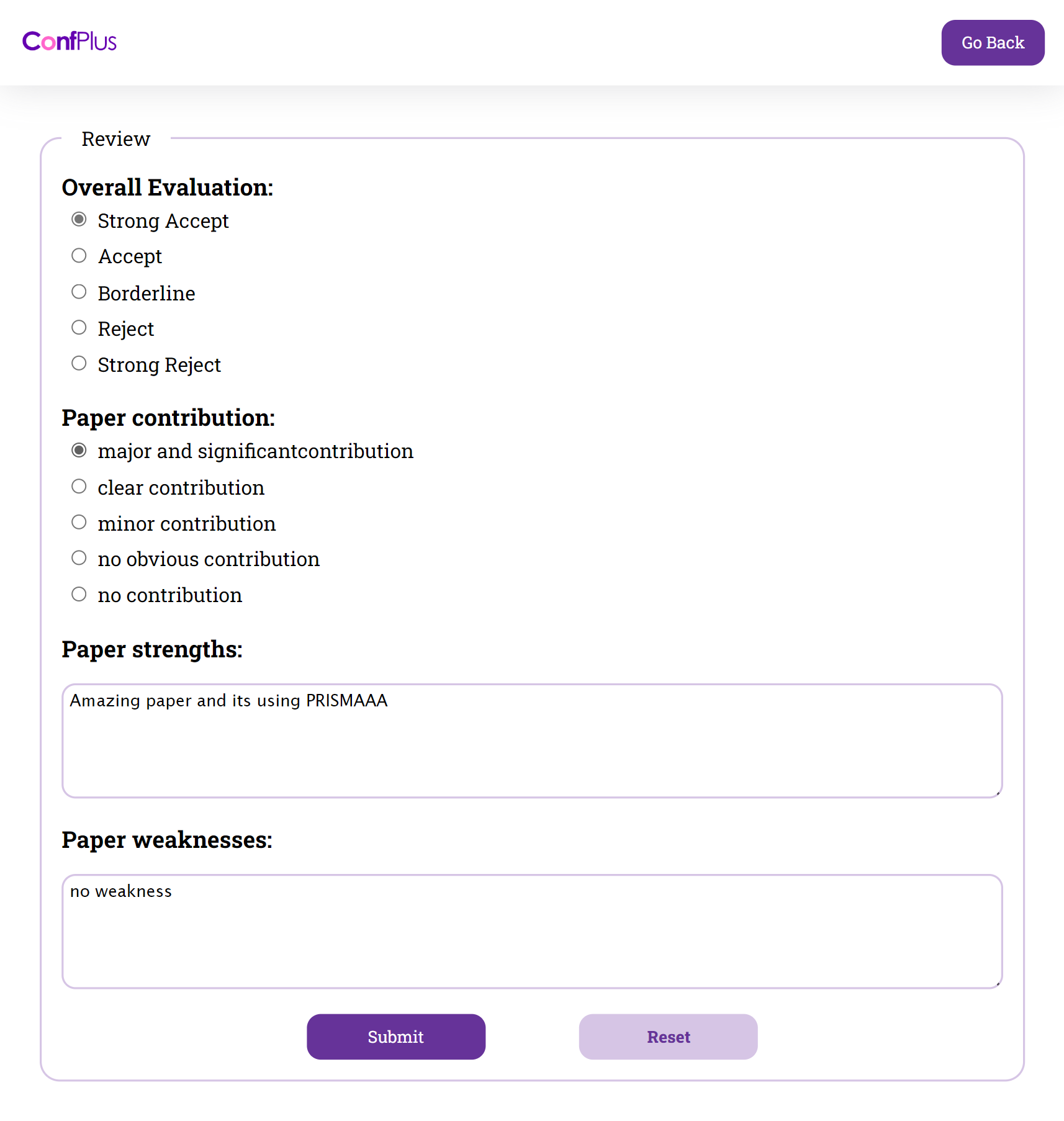


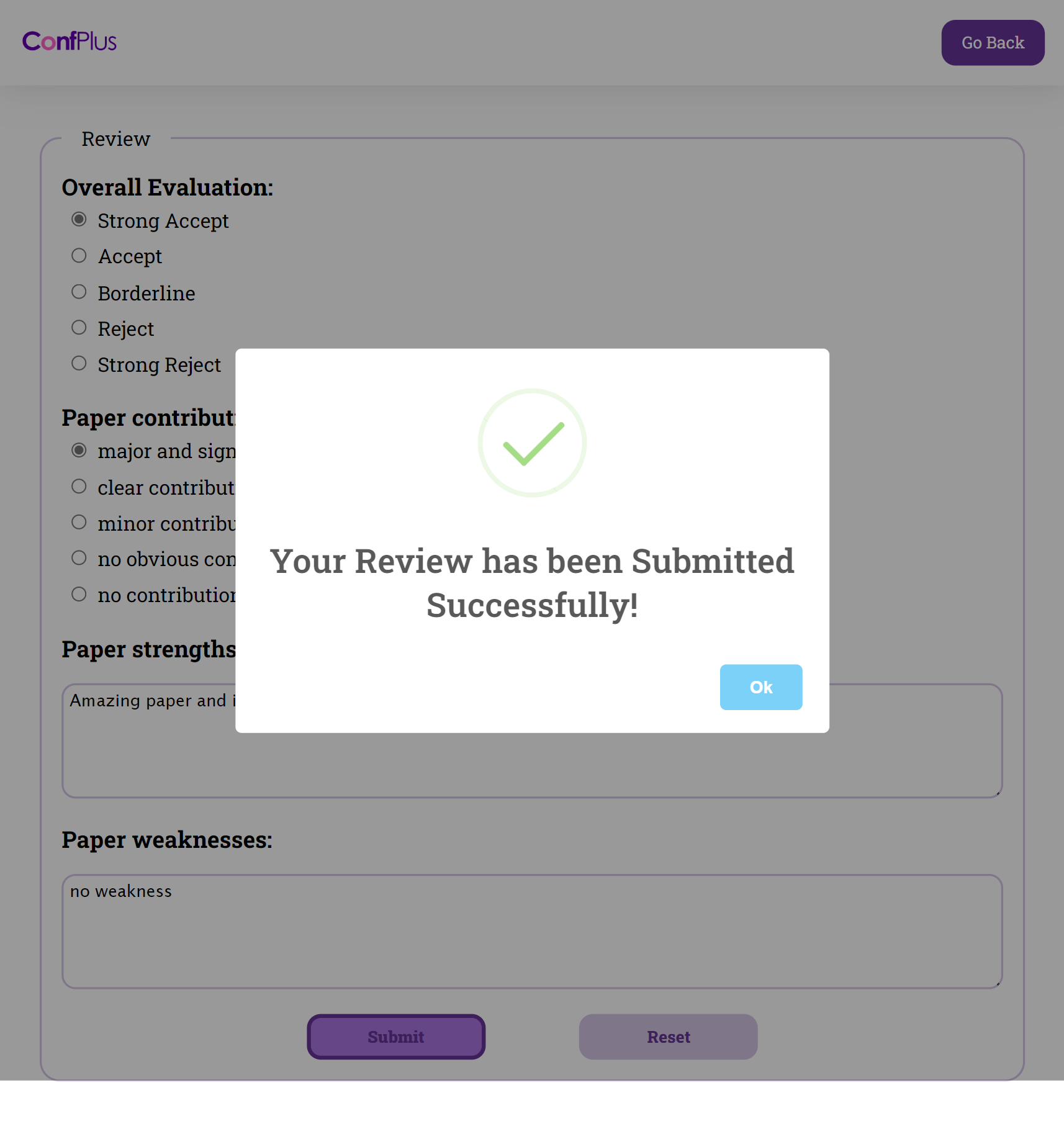
# 







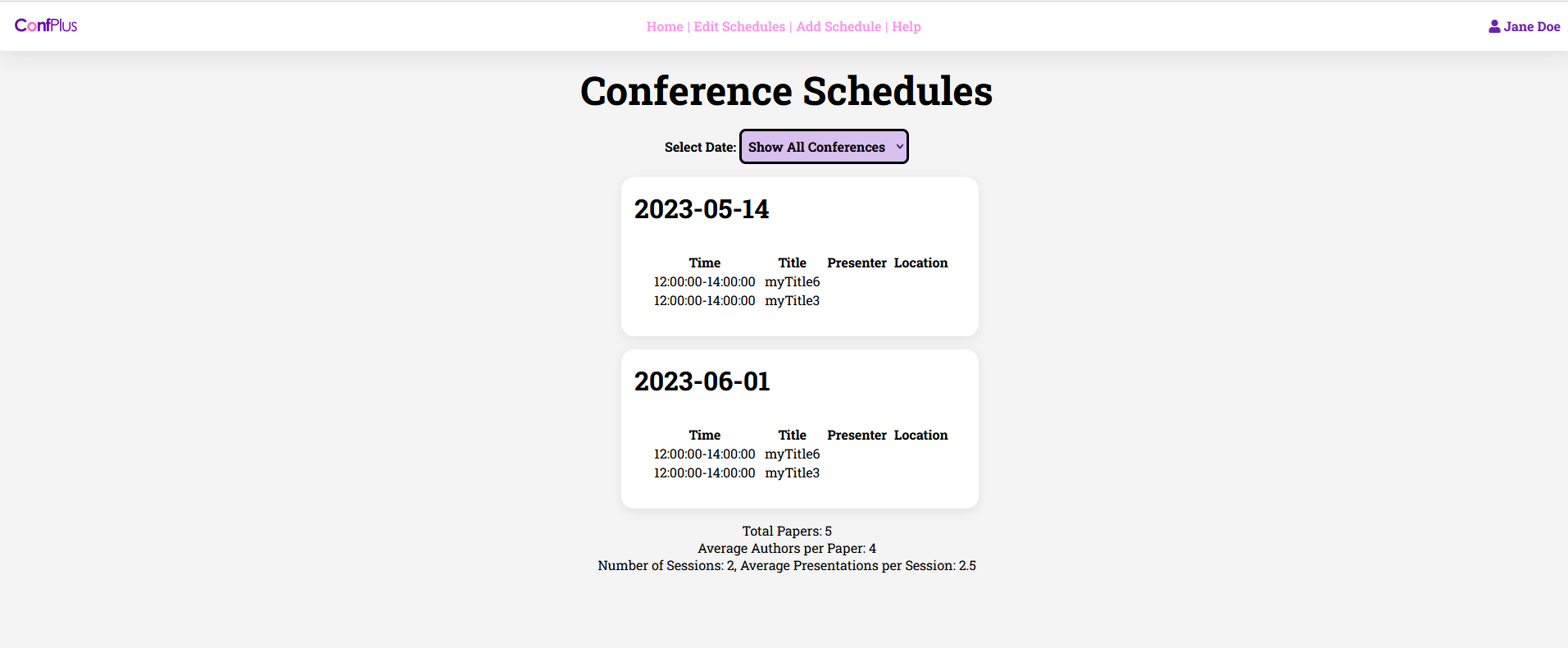




A screenshot of a computer

Description automatically generated with low confidence

# Get conference schedule



A screenshot of a computer

Description automatically generated with medium confidence

# Edit conference schedule

# Conference Statistics Report

A screen shot of a conference schedule

Description automatically generated

# Discussion of the project contribution of each team member

For the contribution it was 25% spread between all four of us, we all worked together to help finish some usecases or troubleshoot the database schema model. All of us worked together to install next.js and create the prisma models, for the use cases we also worked together trying to fix them and making them work with our database