|  |  |  |
| --- | --- | --- |
|  | | |
|  | **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 |  |  |  |

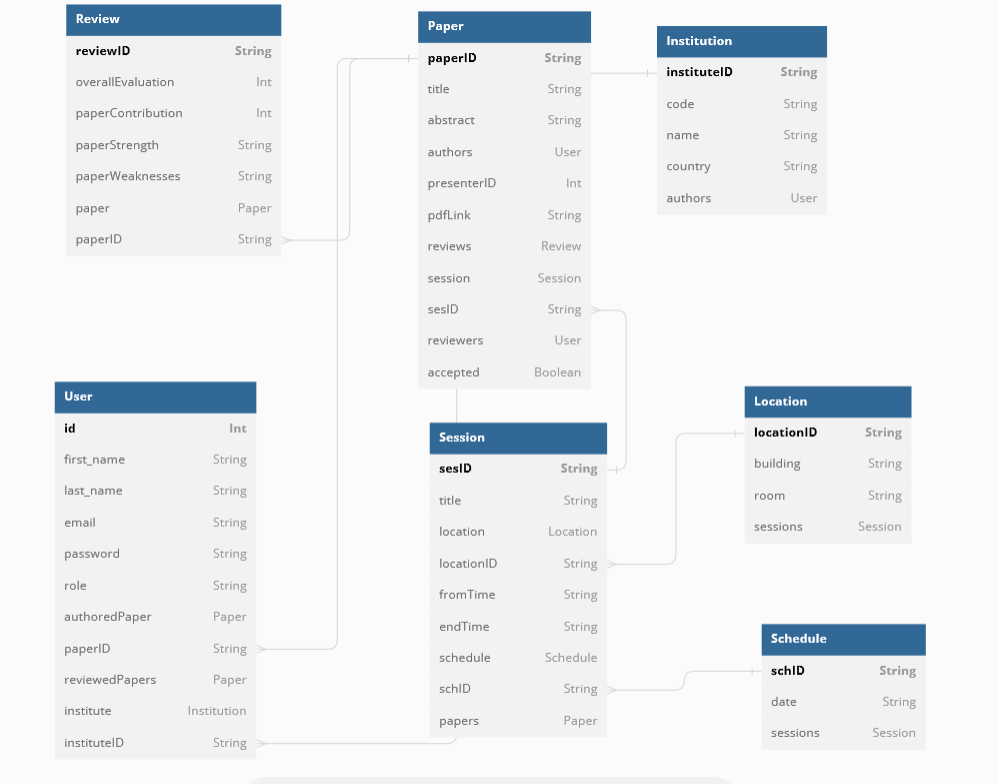
# Application Design

# Improvement over the first phase

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

# Data Model diagram

This is our data model diagram that we used to create our application with SQLite as our provider



# Database population

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

A picture containing text, screenshot, font, design

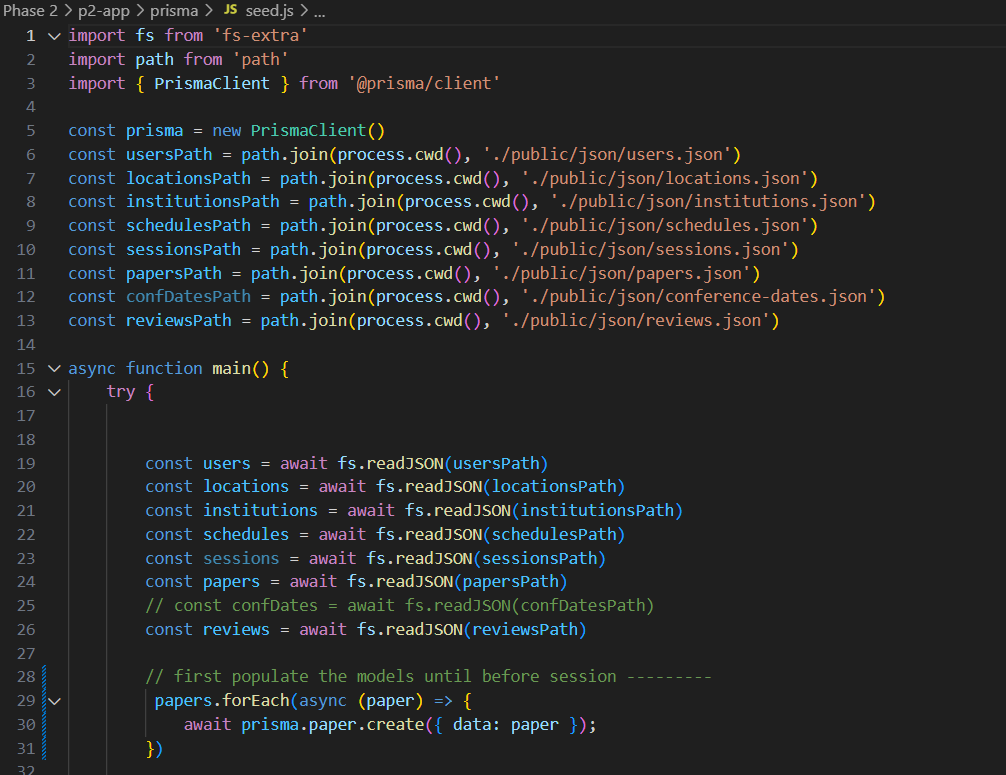
Description automatically generated A screen shot of a computer

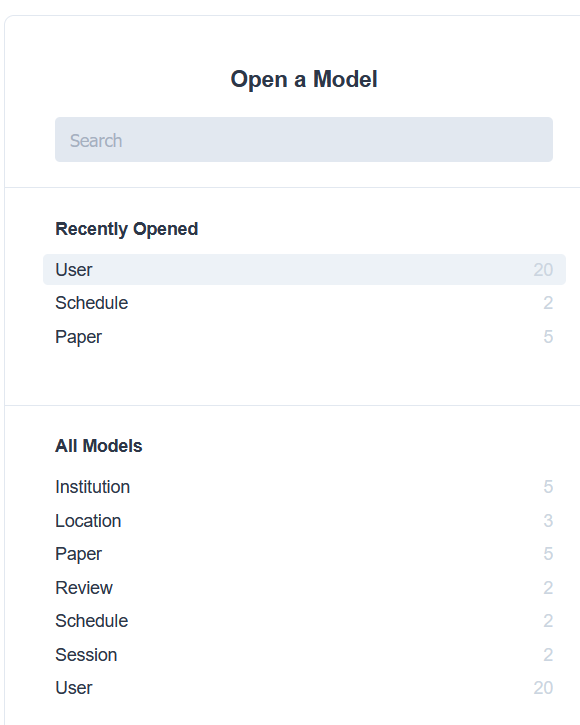
Description automatically generated with medium confidence

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**

This query as to find all the locations and return them

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

With this query we are looking for a specific locationID

return await prisma.location.findUnique({

            where: {

              locationID: locationID,

            },

These two queries are for adding and getting a review from the database

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

Here we mostly have queries that fetch and find specific papers, users, authors and reviewers

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

          }

        });

With this query we will look for a review with a specific ID and update it

 async updateReview(reviewID, reviewData) {

      try {

        const data = await prisma.review.update({

          where : {

            reviewID

          },

          data :

            reviewData

        });

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

        return data;

This query will look for papers with a specific reviewerID and show the reviewId of the paper and the authors.

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');

          }

        }

For this we have 3 queries countPapers which will count all the papers in the database, calculateAverageAuthorsPerPaper which will get the count of all the papers and all the authors and return the average. And lastly calculateAveragePresentationsPerSession will get the number of presentations and the average presentation per session.

 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,

        };

      }

**Data cache strategy**

|  |  |  |
| --- | --- | --- |
| **Page** | **Caching strategy** | **Brief justification** |
| All pages | Force-cache (default) | We’re not expecting data to change frequently so we made it force-cache so we don’t need to call the API every time. |

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

Since the model solution was not provided for us, we will write what we learned from phase 2:

In phase 1 manipulating data was difficult with JavaScript and required many lines of code. After we switched to Prisma we discovered how simple and efficient it is using a database as it gives us many ways to manipulate our data easily. From filtering to finding a specific id to getting the statistics

Another thing we learned was how API routing now works with our application and we have learned how to create the correct files and repositories to add our routes and how to write our URLs correctly.

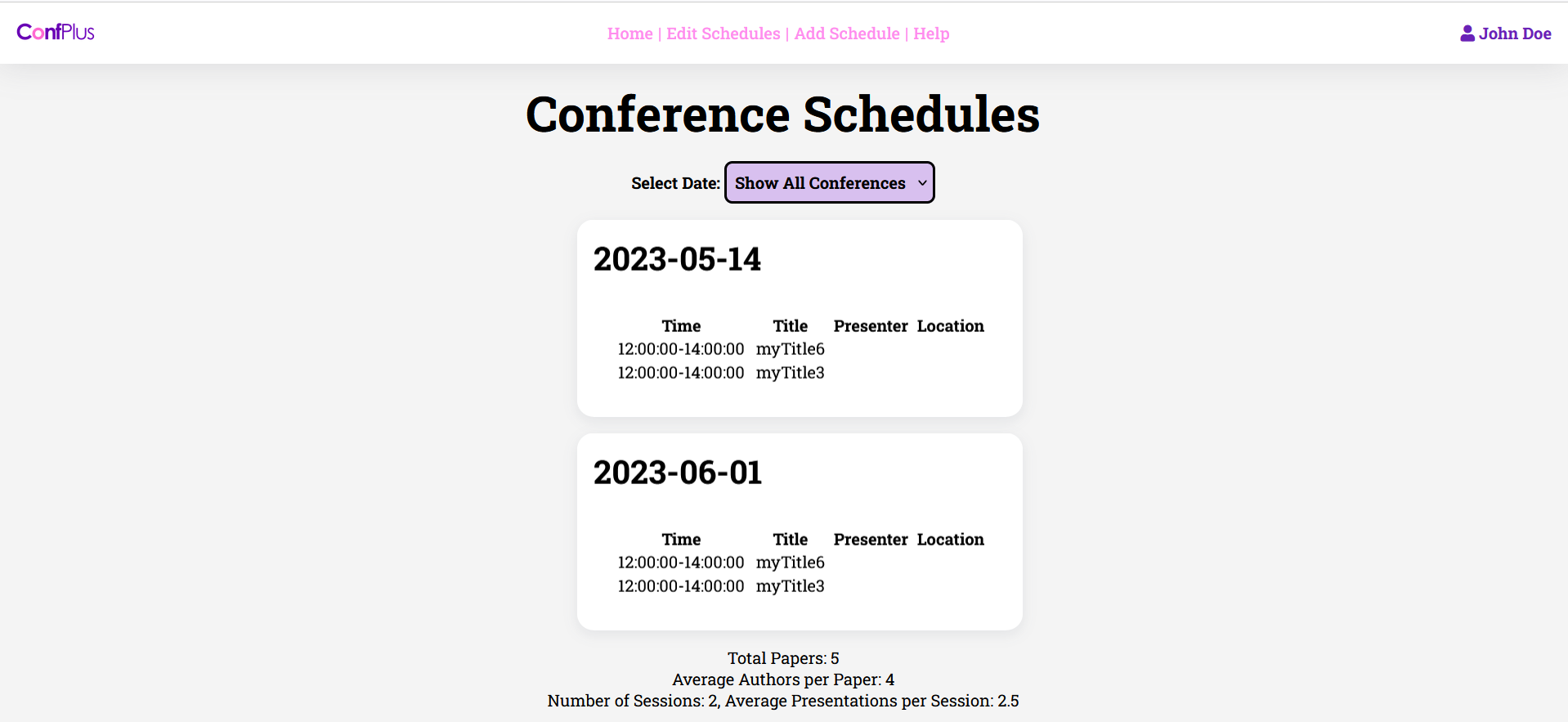
We also learned how to take a pure html css and js project and convert it to a functioning next.js application.

# Testing

# Custom Login

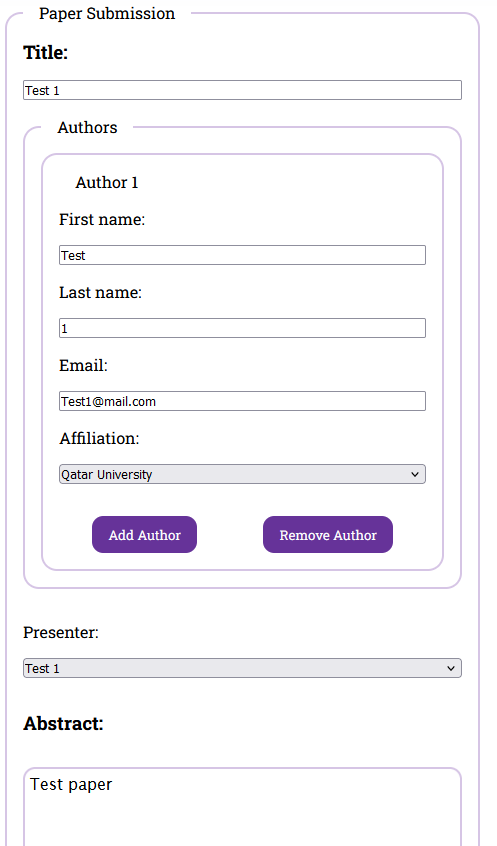
A screenshot of a computer

Description automatically generated with medium confidence

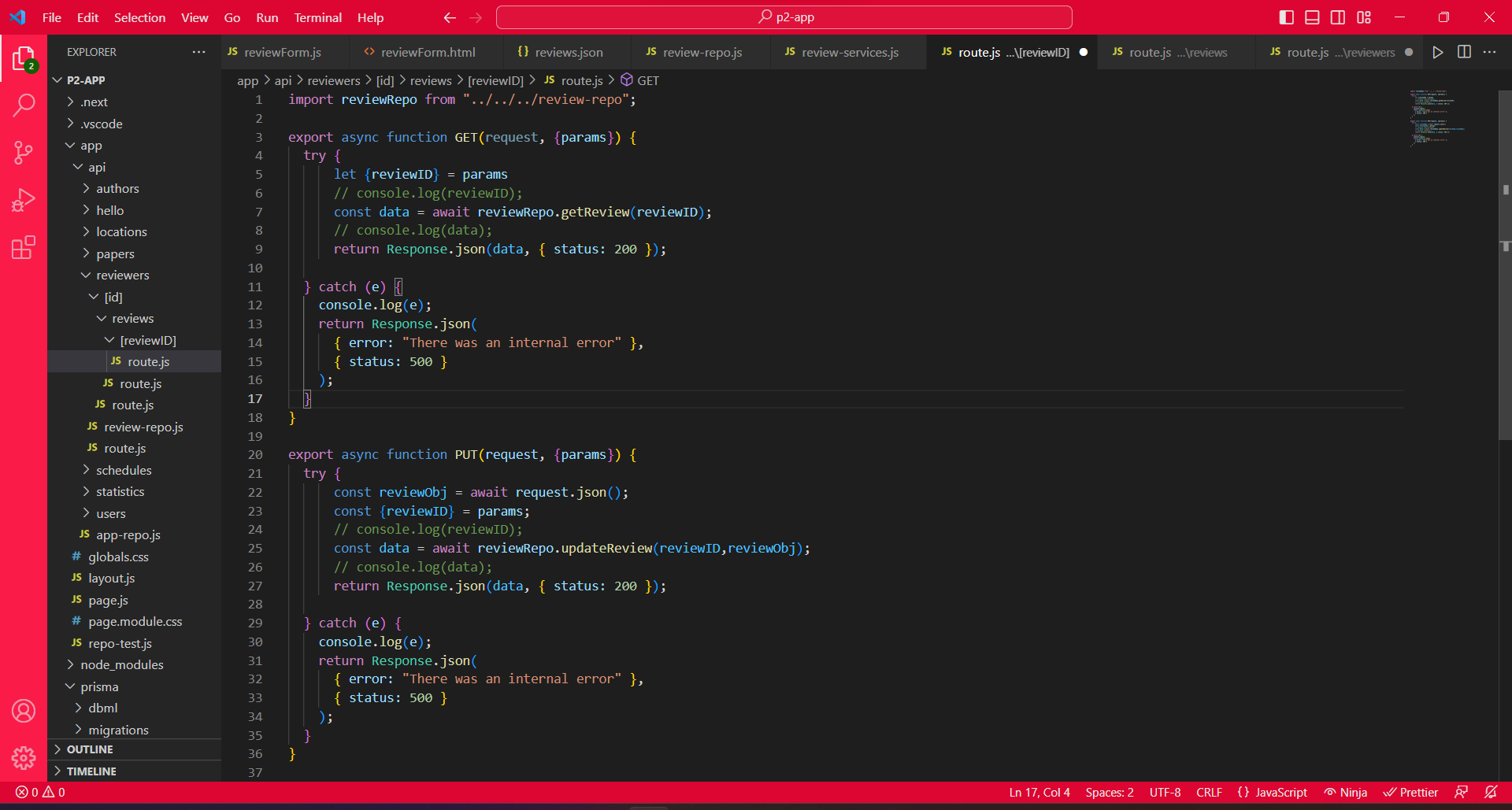
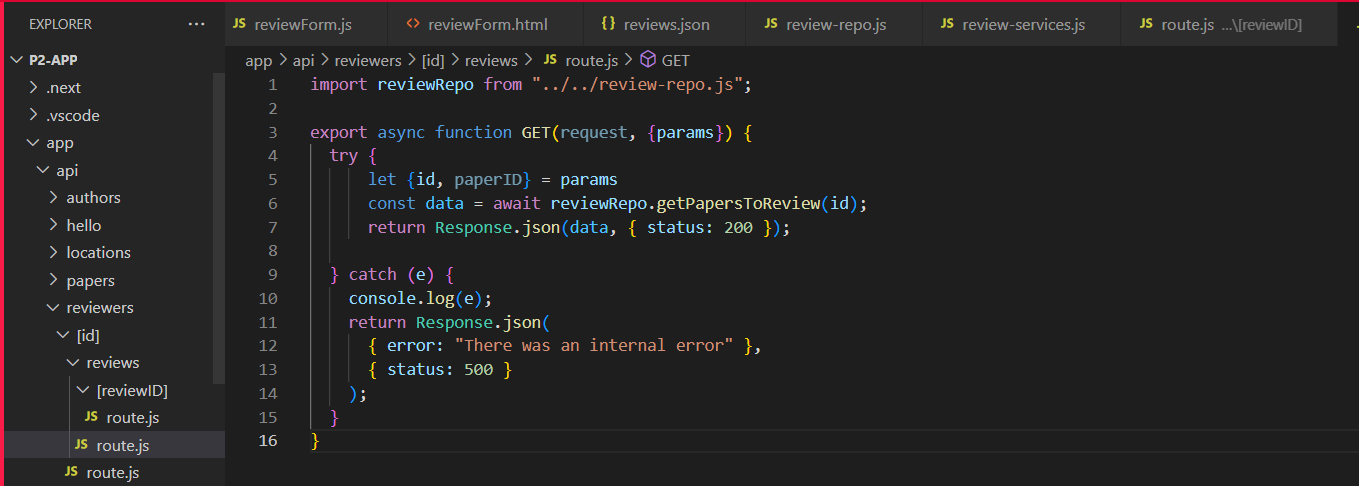


# Login using 2 Authentication Providers

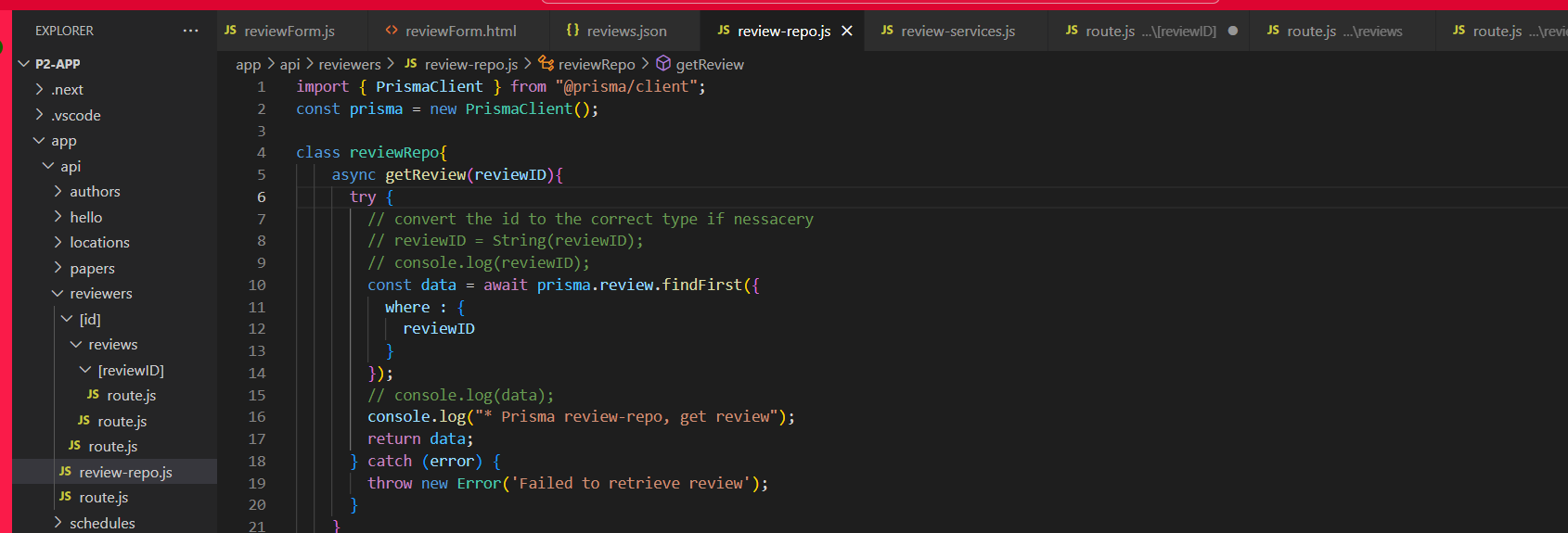
# Submit paper



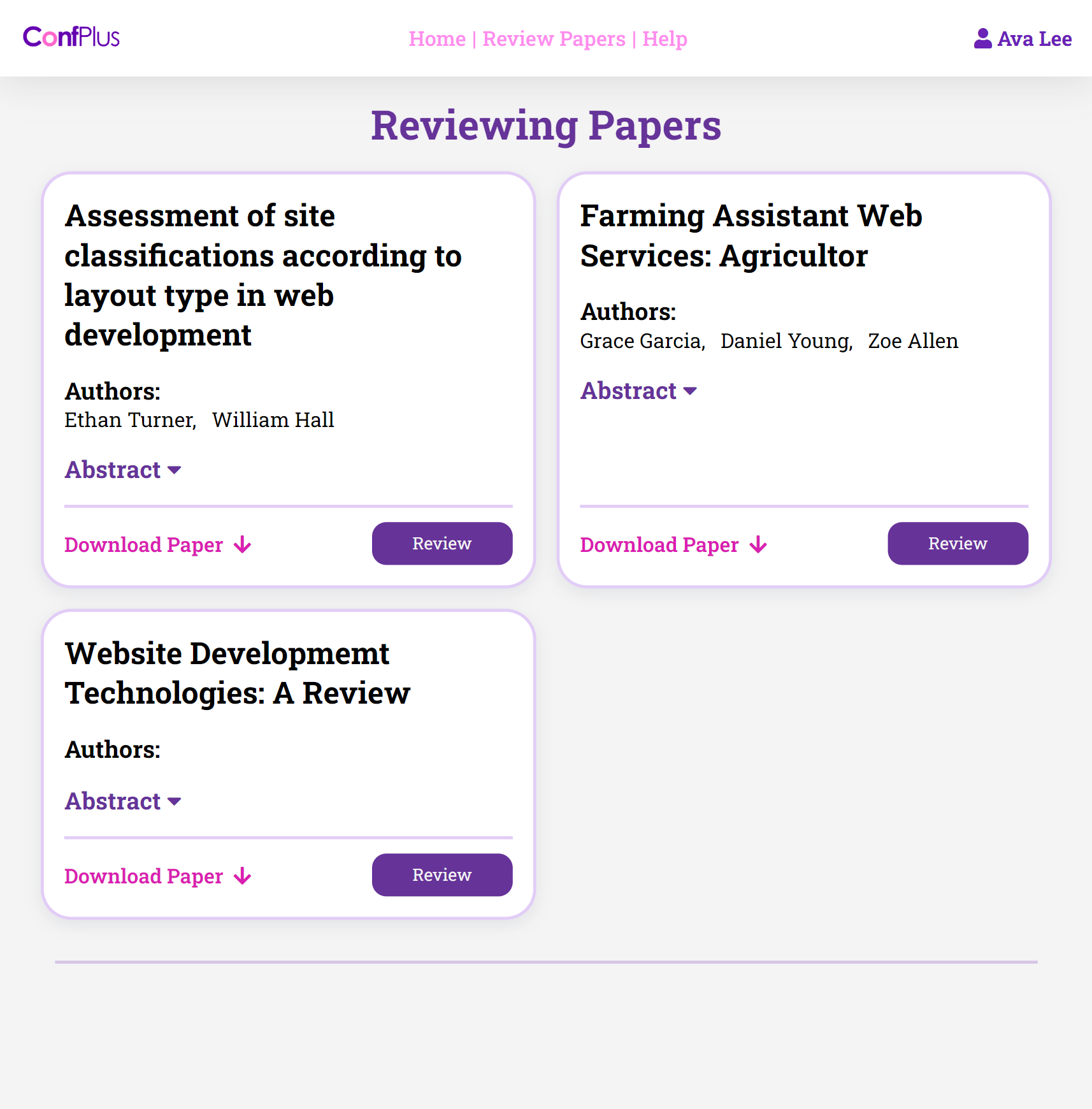
# Review paper

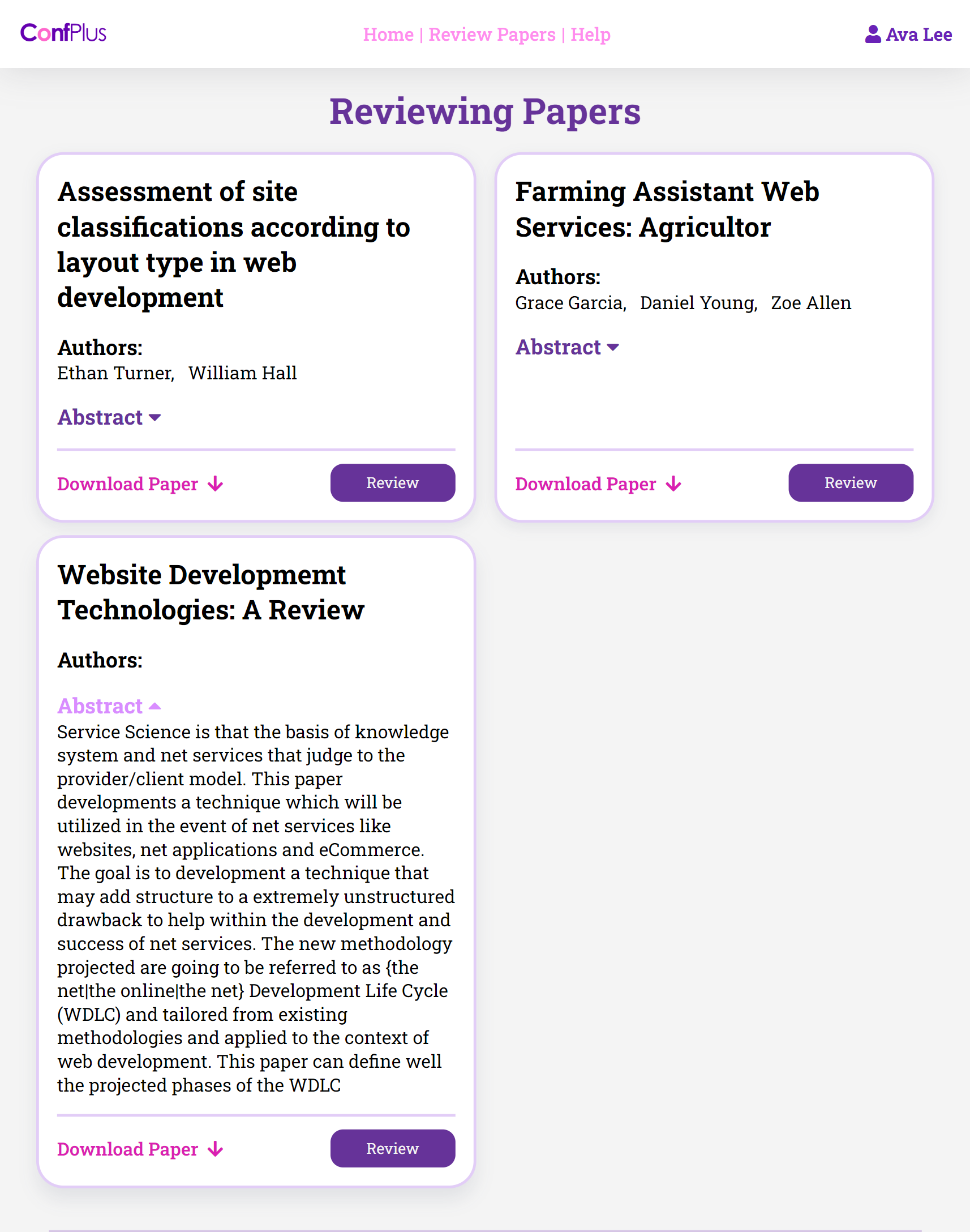


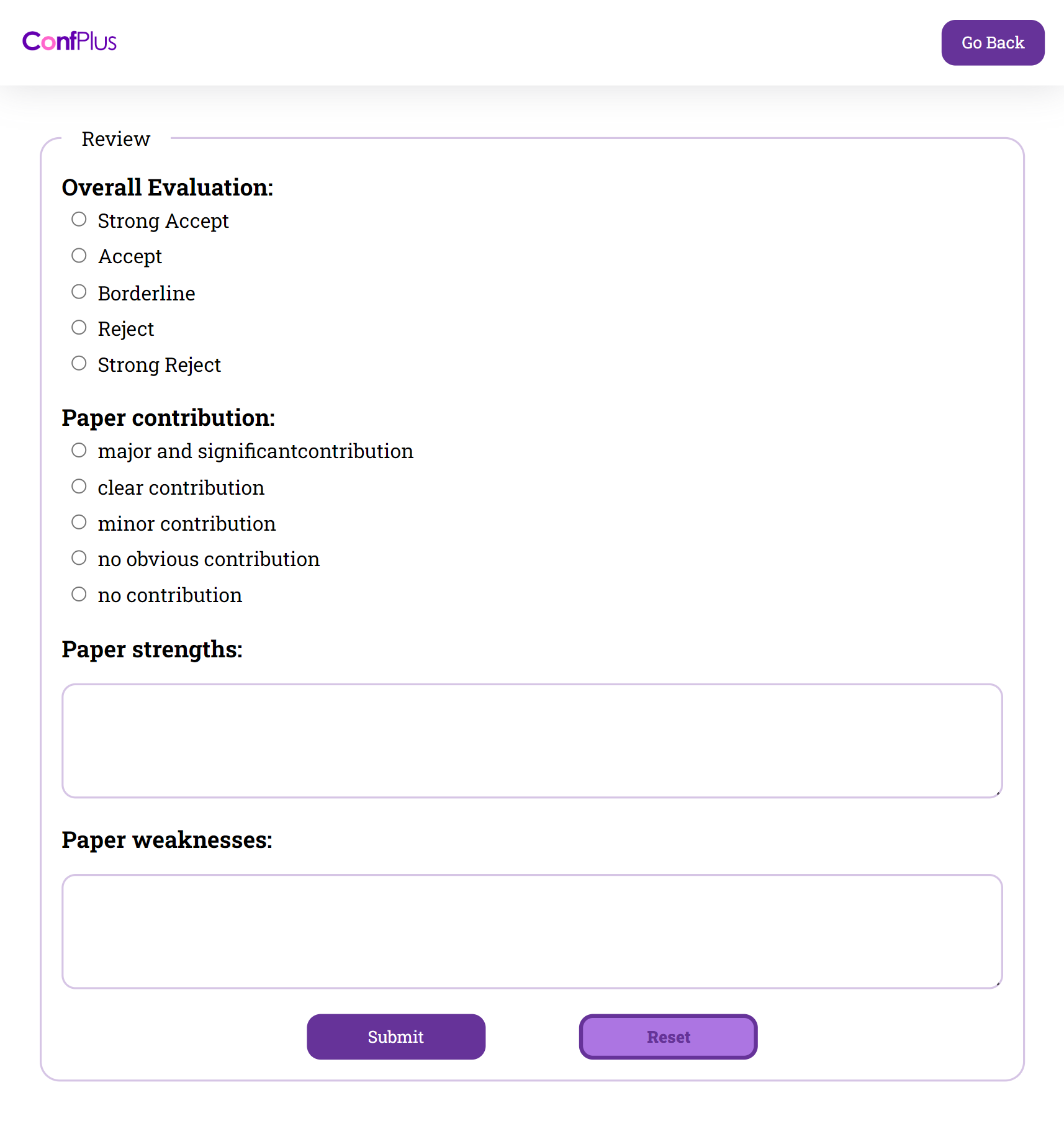
# 

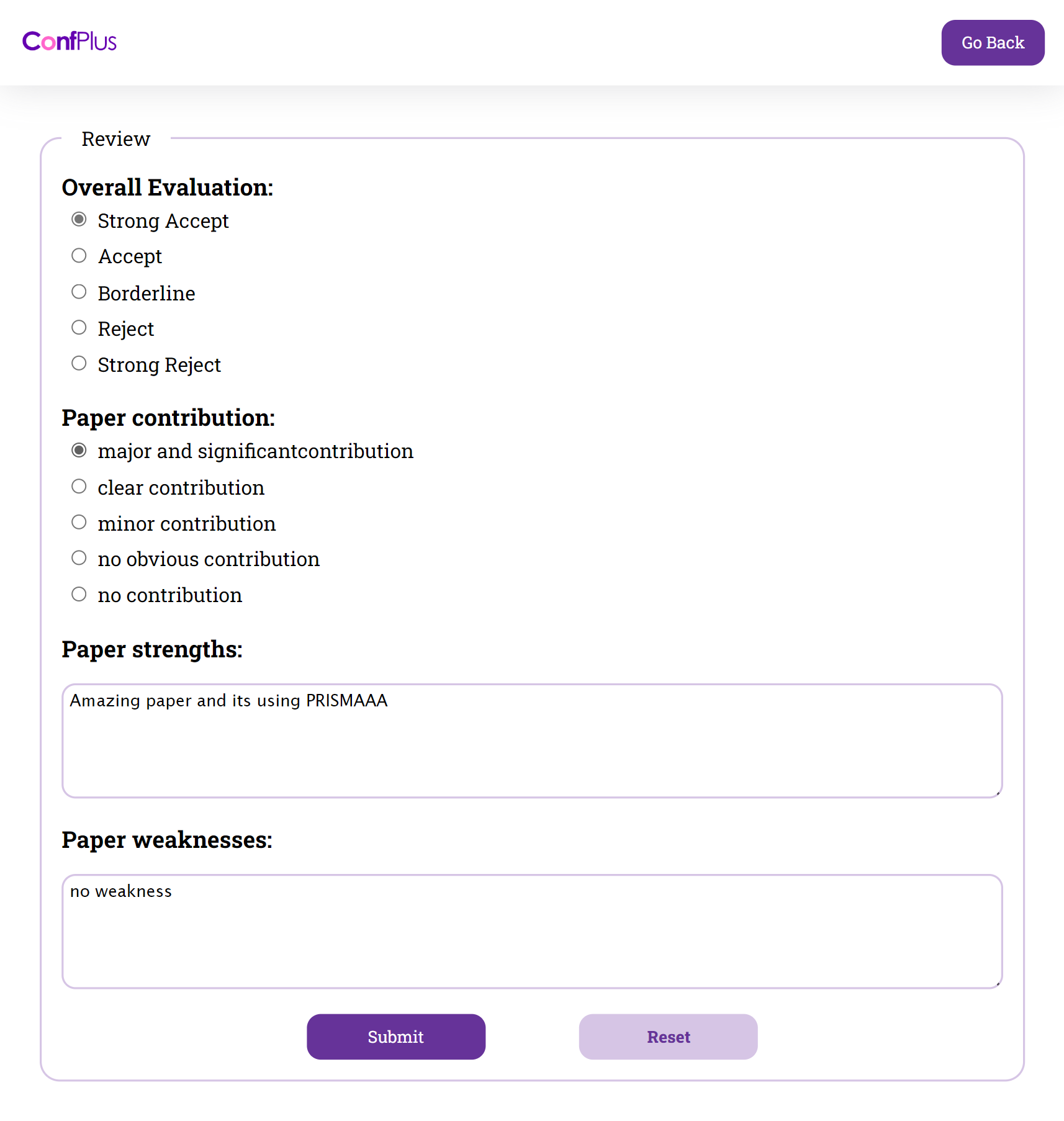


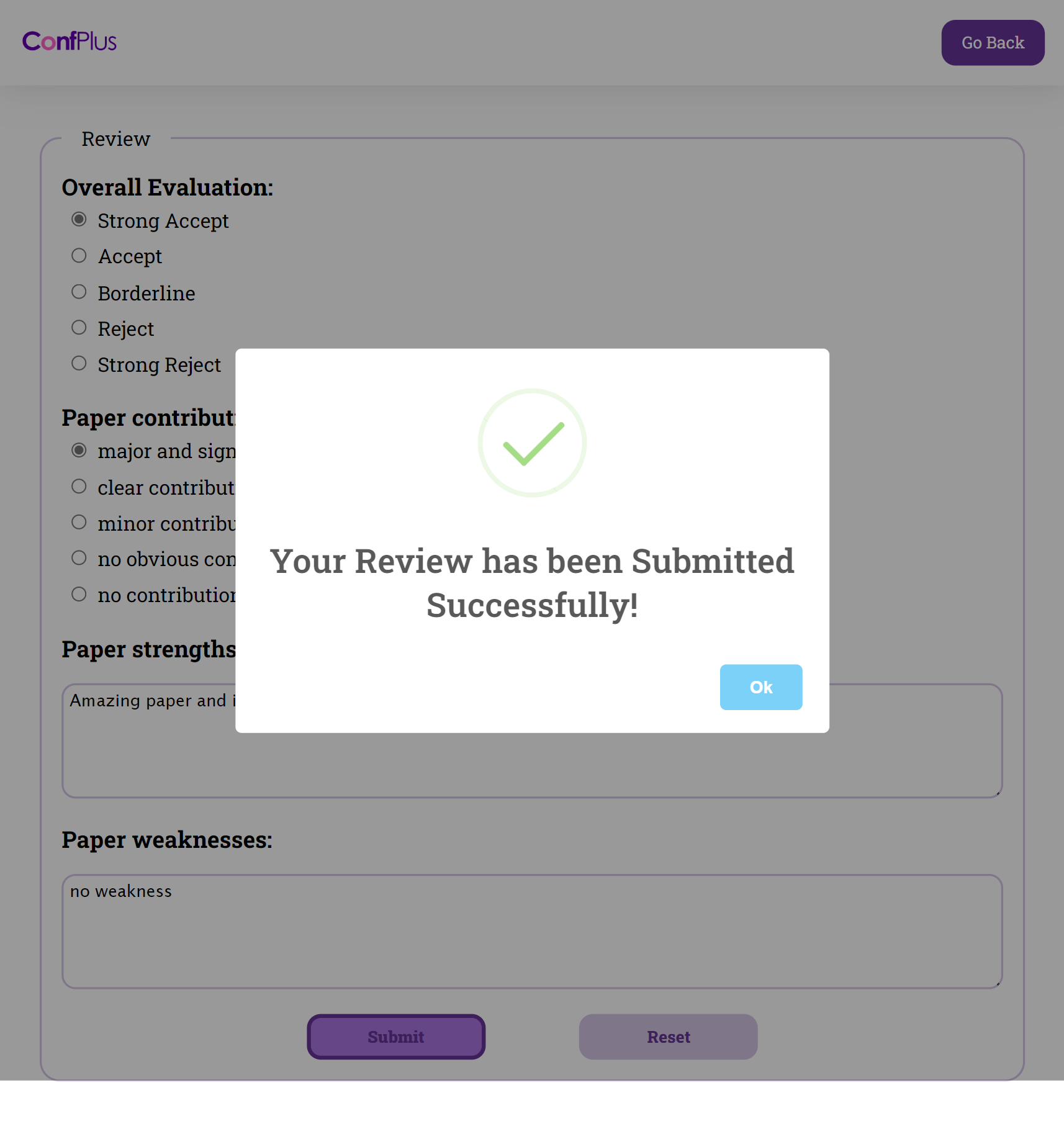
# 







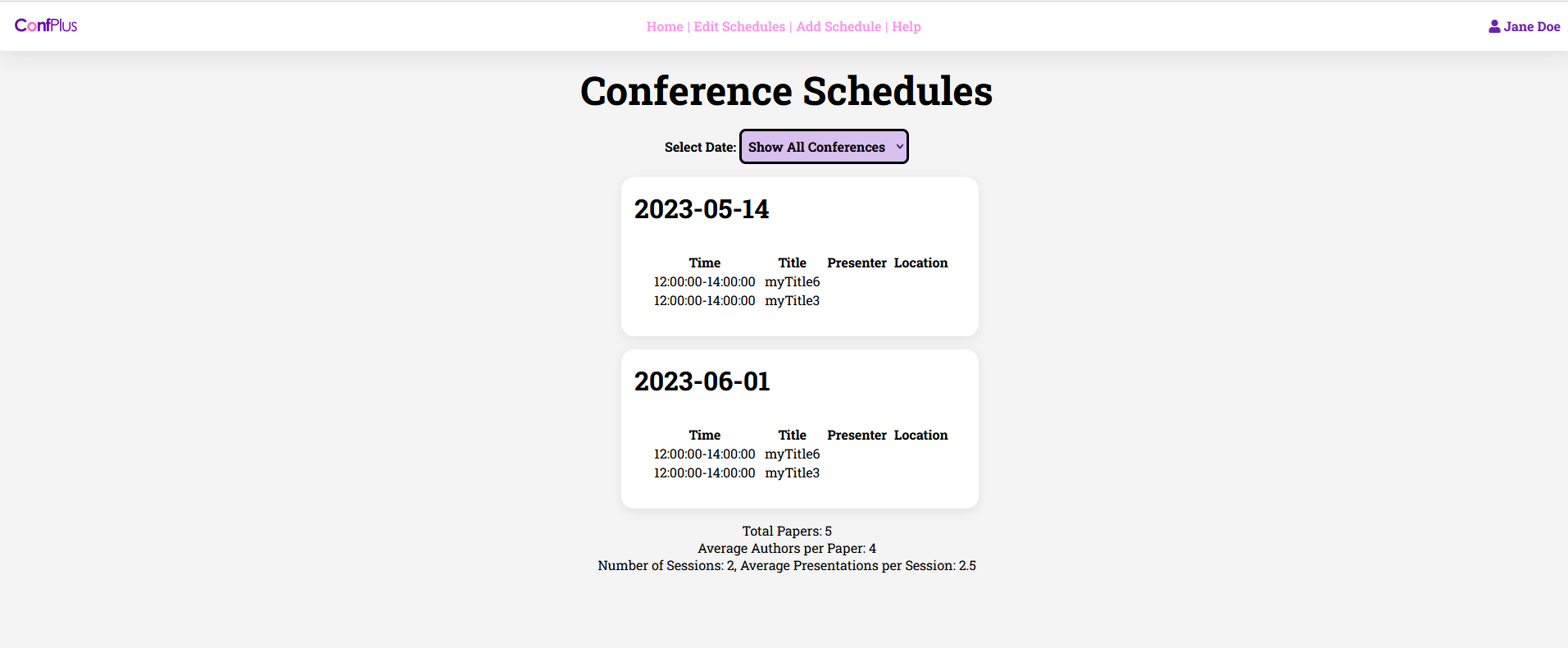




A screenshot of a computer

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 some of us worked on use case 1,2,3,4 while others were fixing issues with the database or queries or routing issues. We also worked on the data model, then one of us created the seed and checked after him to troubleshoot the seed with our prisma schema model. Each member took his part from phase 1 and changed it to prisma.

|  |  |  |
| --- | --- | --- |
| **Name** | **Contribution** | **Task** |
| Mohamed Dhia Abdaoui | 25% | Prisma schema model, use case 1 fixes, use case 2 fixes |
| Ridhwan Athaullah | 25% | Use case 1, use case 2 , use case 4 |
| Mohamad Allaham | 25% | Fixes with the prisma schema model. Fixes with the navbar, use case 3, fixes to the statistics |
| Saoud Ali Al Khelaifi | 25% | Helped with the installation of next.js, helped with use case 1, statistics functions. |