|  |  |  |
| --- | --- | --- |
|  | | |
|  | **CMPS 350 Project Phase 2 – Conference Management System (ConfPlus)**  **(15% of the course grade)** | |
| **Group Id:** | | G? |
| **Group Members:** | | Abdulla Al-malki (202009135)  Ahmed Deef (201606478)  Mohammed Al-Obaidly(201801987)  Youssef Ahmed (202107162)  **Emails:** [aa2009135@qu.eu.qa](mailto:aa2009135@qu.eu.qa)  [ya2107162@qu.edu.qa](mailto:ya2107162@qu.edu.qa)  [ad1606478@qu.edu.qa](mailto:ad1606478@qu.edu.qa)  [ma1801987@qu.edu.qa](mailto:ma1801987@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 | 50% | We designed the database to complete the missing functionalities from phase 1 by adding status attributes to papers, reviews, sessions. However, it was done on server-side only.  The review page though has the new functionality showing reviewed papers.  Instead of static html, the application ui is fully from react and user friendly. It is mobile friendly as all pages’ layouts are responsive  We used tailwind css for styling instead of vanilla css |  |
| Design and implement the Data Model.  Clarity of data entities, their attributes and relations (in Prisma and the conceptual model (the diagram)) | 10 | 100% | The design is fully implemented |  |
| Init DB: populate the database with the data from the json files. | 5 | %100 | Database gets initially populated with seed.js |  |
| Repository Implementation to read/write data from the database | 10 | %80 | Almost all CRUD methods for entities were created. |  |
| 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 | %50 | Statistics page was not done but 2/3 APIs are ready. The sessions statistic is not complete  All queries were from the database |  |
| **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 | 90% |  |  |
| **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

Give details and proof about:

* What was missing in your previous phase

We were lacking better user experience as our first implementation was too abstract. Authors couldn’t check their submitted papers for example.

* Your improvements + proofs

We made the repo take data from database instead of Json files.

The web UI is implemented through react.

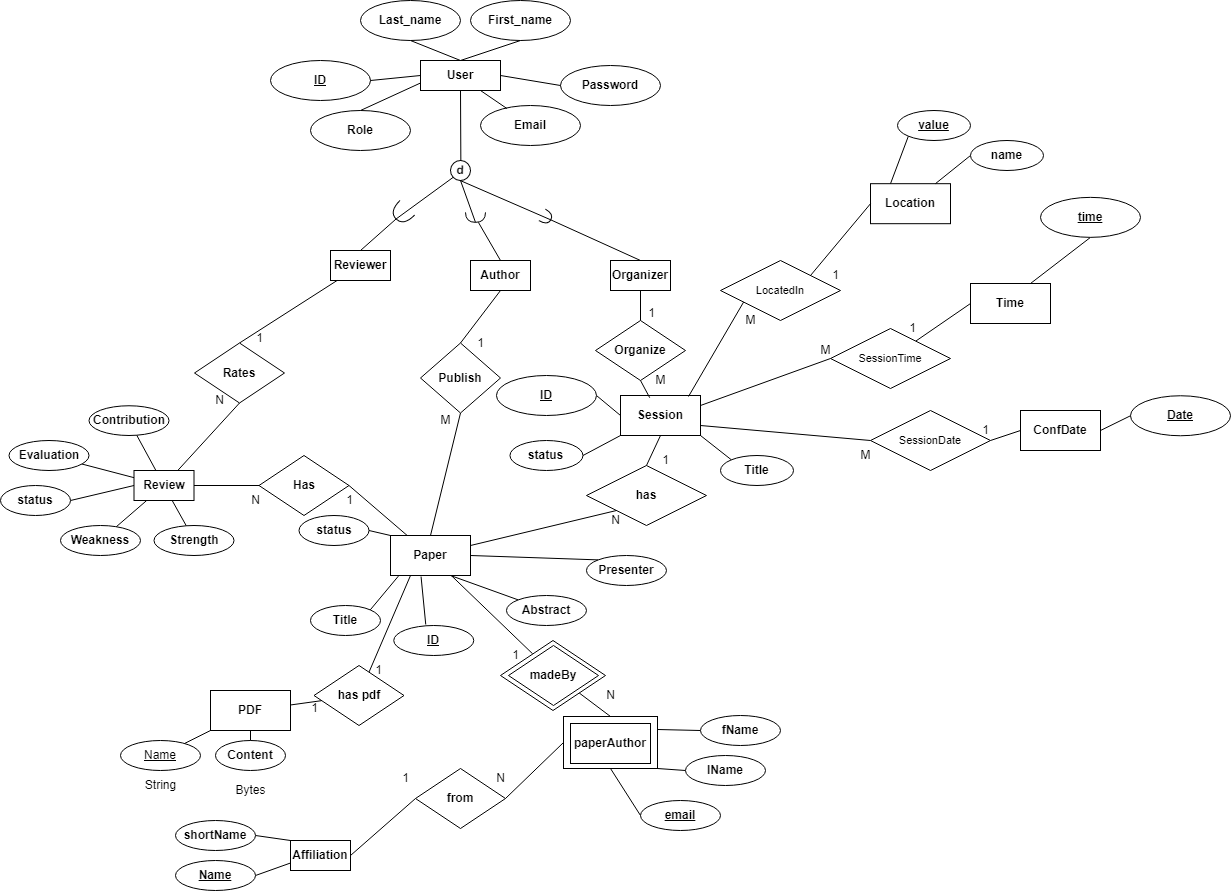
Extra data to entities to categorize them based on status.

* What was not improved and still missing
* The full UI. Even though the backend is almost complete, the frontend is about half complete. The pages are there and responsive, but they’re missing some fetches and cards

|  |  |  |
| --- | --- | --- |
| Page | Caching strategy | Reasoning |
| Login | No-store | User data could change and we need to check that whenever we login |
| Home | No-store, revalidate | New conference dates could be added in the future and papers  Sessions may get revalidated every minute to stay up to date with the latest sessions |
| Staff/ | No-store, force-cache | More papers could be published later or sessions or reviews  But some data like locations can be stored in the cache with no issue |

# Data Model diagram

Conceptual Data diagrams, Prisma Model….

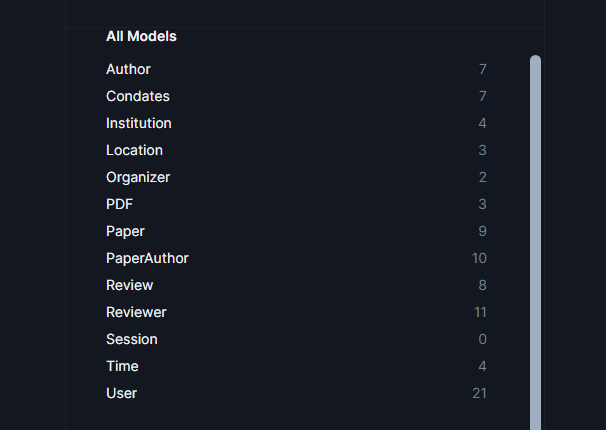


# Database population

Give details and proofs with screenshots how you have populated database, the current content ..

The schema was modeled by the diagram we made. We made a superclass User with subclasses based on role of the user. Each role has a specific relationship with the application entities.

The model was originally oriented on the json data files, as they will be used also to initialize the database.



# Database

Give details and proofs about how you have updated your application to use database. This applies to all use-cases.

We updated the application to use prisma queries in getting data instead of reading the json files immediately using fs library.

Detail your statistics page

Statistics was only made server-side

Give the list of your Prisma queries

Conference-dates:

* readDates: Accepts no inputs, returns all assignable dates for the conference.

Institutions:

* readInstitutions: Accepts no inputs, returns all assignable institutions for the conference.

Locations:

* readLocations: Accepts no inputs, returns all assignable locations for the conference.

Login:

* readUser: Accepts the email address and password for a user, returns the user in question if the email and password match.

Papers(/[id]):

* createPaper: Accepts an initial object for the paper (containing the parameters of title, abstract, author ID, paper authors, and the presenter), file name, and file content. Creates a database paper tuple based on the inputs in the paper object, as well as a PDF tuple based on the ID of the newly created paper, the file name, and the file content. The function also creates the paper author tuples included, as well as the two reviews needed. Returns the newly created paper.
* readPapers: Accepts a reviewer ID as well as a status. Returns a list of papers based on the values of the reviewer ID and status.
* readPaper: Accepts a paper ID as input. Returns the paper if it exists, and various error messages depending on the problem, including the lack of a paper.

Papers/[id]/download:

* readPdf: Accepts a paper ID as input. Returns the PDF object associated with the paper if it exists, as well as the content of said object as a PDF file.

Paper/[id]/review:

* readReviews: Accepts a reviewer ID and status as inputs. Returns a list of reviews based on the values of the reviewer ID and status.
* updateReview: Accepts a paper ID, reviewer ID, modification object, and review ID as inputs. Returns the updated review after updating the paper and reviews’ statuses as well based on the total evaluations of the paper.

Reports:

* NoOfPapers: Accepts no inputs. Returns a list of the different papers’ statuses as well as the number of papers per status.
* AverageAuthorsPerPaper: Accepts no inputs. Returns the average number of authors per paper. This is calculated by taking the list of authors of each paper, mapping the lengths of each list into a new array, finding the sum of the lengths, then dividing the sum by the total number of papers.
* NoOfConfereneSessions: Accepts no inputs. Returns the total number of sessions currently in the system.

Sessions:

* createSession: Accepts an object as an input. Creates a new session object then outputs said object.
* readSessions: Accepts a date as input. Returns the list of all sessions if the date is null or otherwise invalid, or the list of sessions on that date if the date is valid.
* readSession: Accepts a title as input. Returns the session with that title.
* updateSession: Accepts a title and object as inputs. Returns the session with that title after modification if it exists.
* deleteSession: Accepts a title as input. Returns the specified session after deleting it if it exists.

Times:

* readTimes: Accepts no inputs. Returns the list of all time slots specified for the conference.

Users:

* readUsers: Accepts a role as input. Returns a list of users based on the value of the role.
* readUser: Accepts an ID as input. Returns the user specified with that ID if it exists.

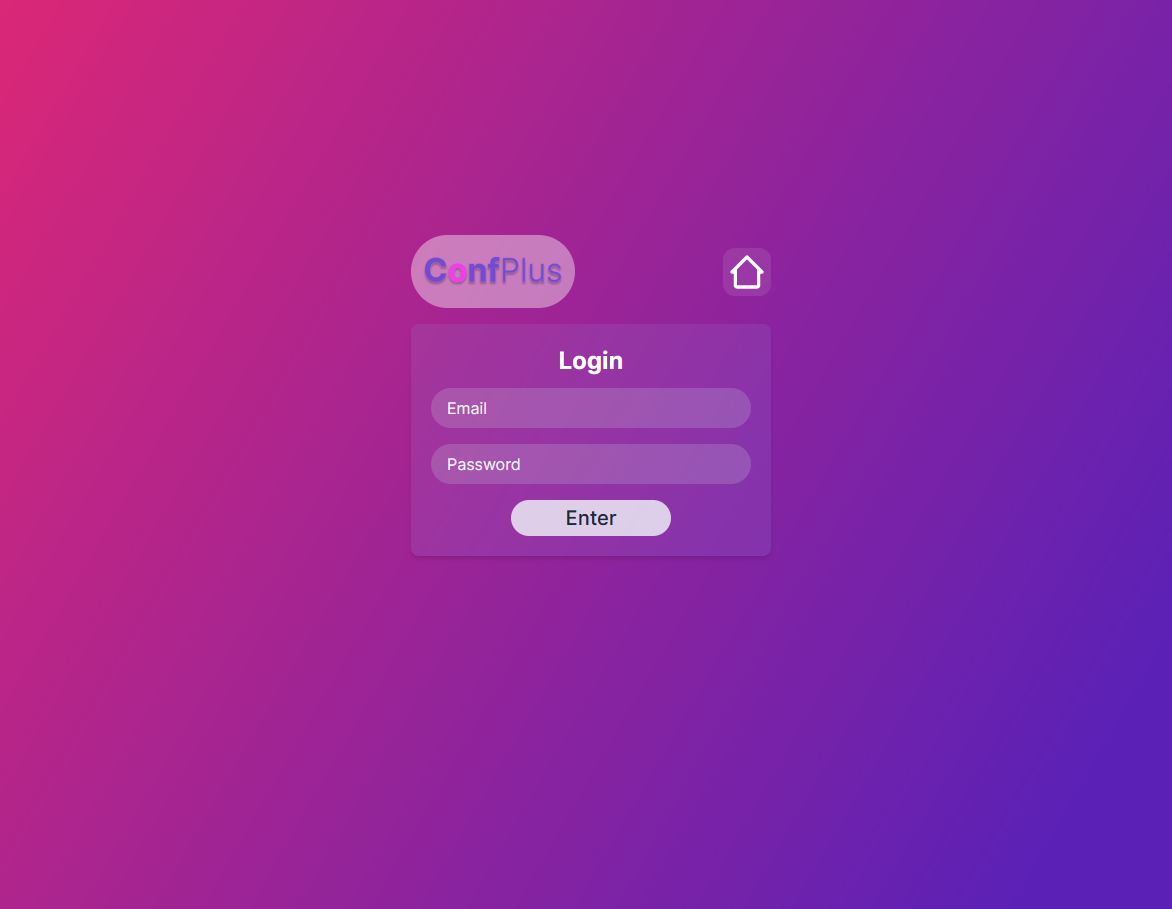
# 3 technical lessons learned from your submitted solution vs. the model solution

Technical Lessons Learned:

1. The first ERD does not need to be absolutely correct the first time. The advantage of prisma migrations is that we can always update our schema whenever we find we missed something or need to modify. At the same time, the rest of the team could work on the project using prisma client with little modifications needed later. The key here is saving time working on the project while the schema gets modified.
2. Cache options need to be taken care of. Some sensitive data could be needed to stay up to date and force-cache options will not make them work in the right way.
3. Initially starting UI with react saves time as layouts give the ability to reuse components shared by many pages.

# Testing

# Custom Login

A screenshot of a login screen

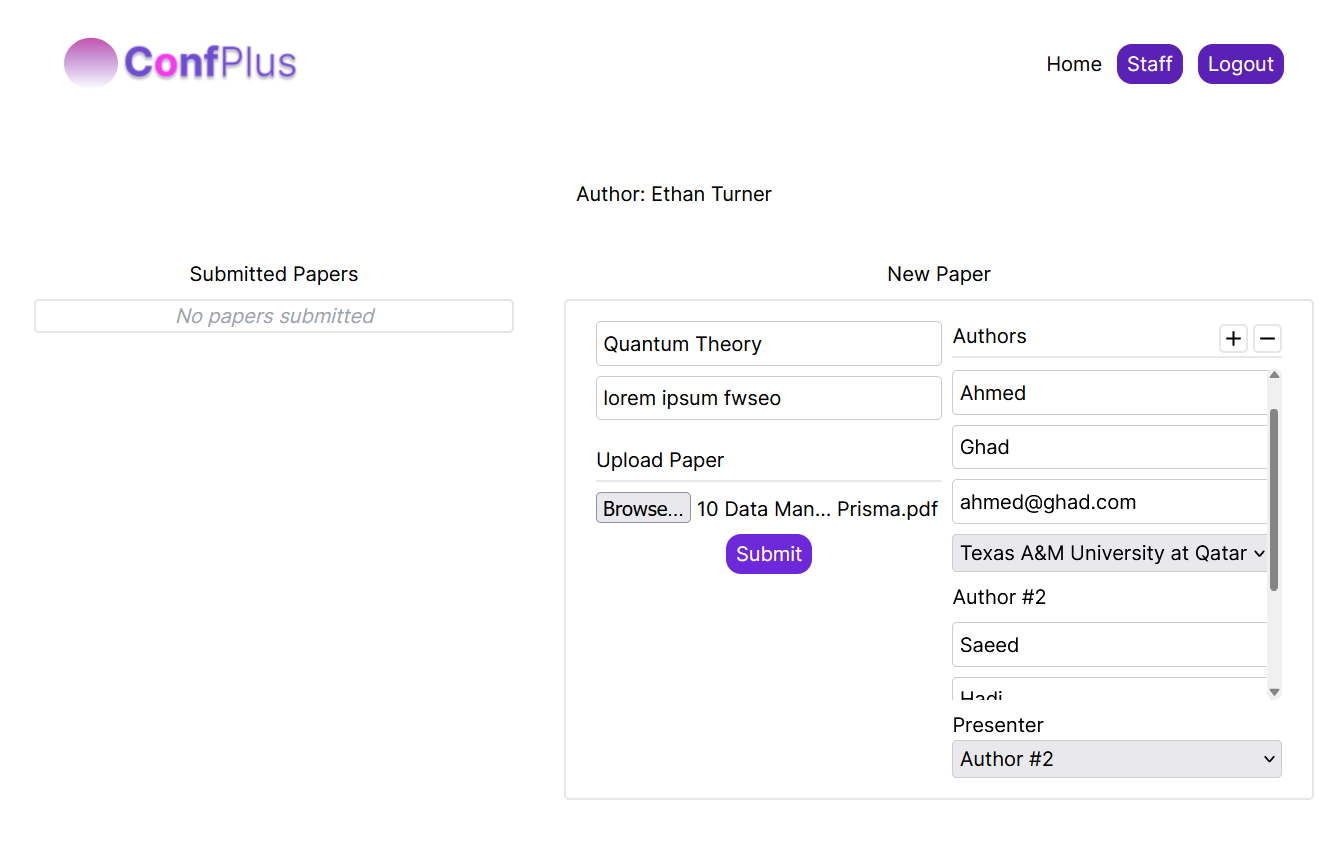
Description automatically generated with medium confidence

# Login using 2 Authentication Providers

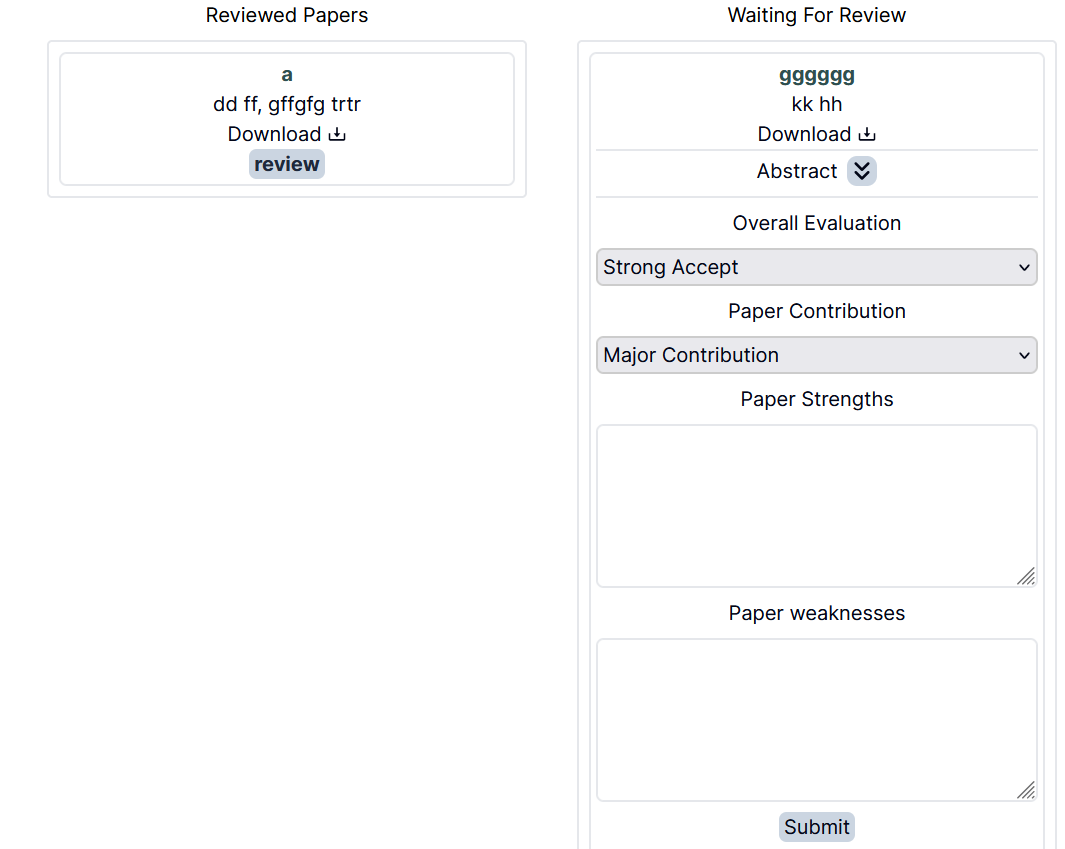
# Submit paper

A screenshot of a computer

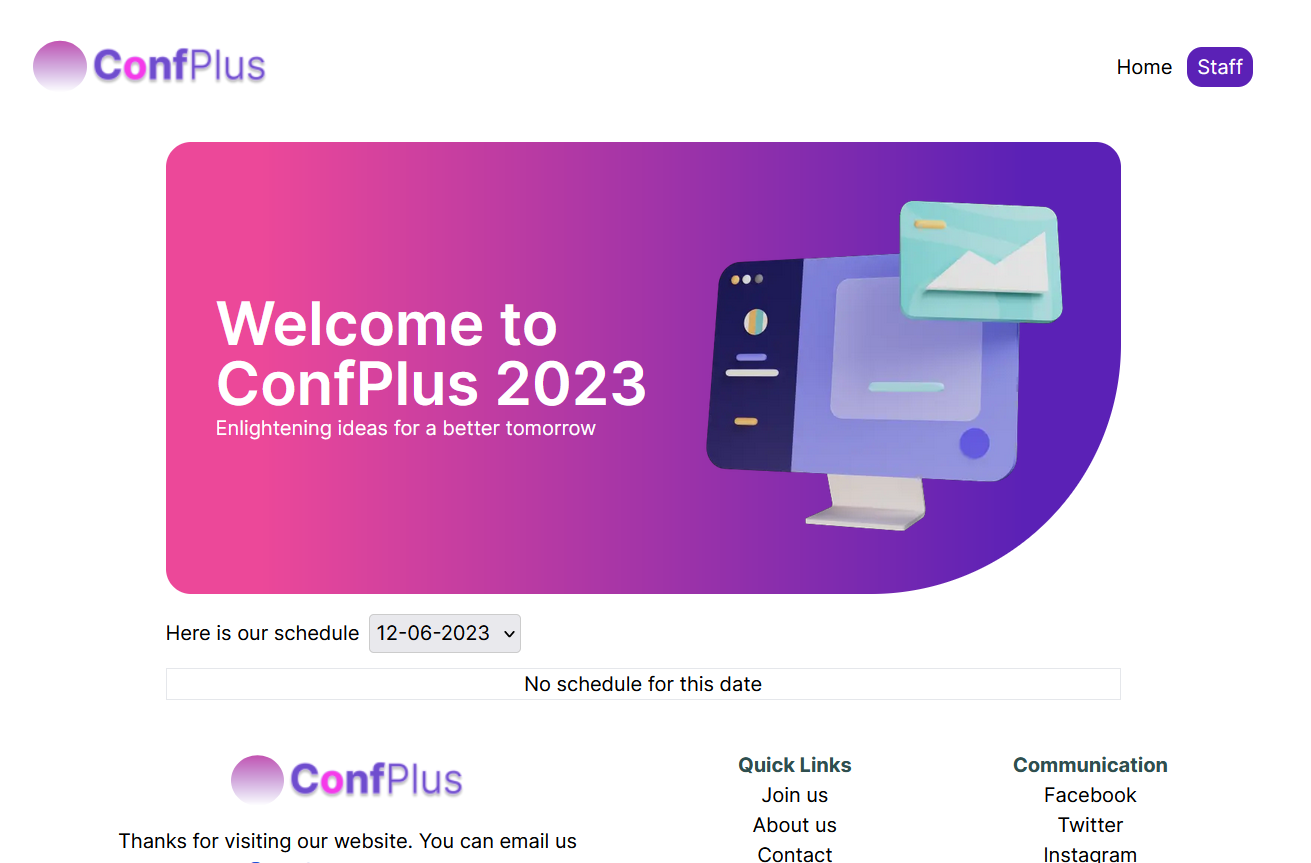
Description automatically generated with medium confidence



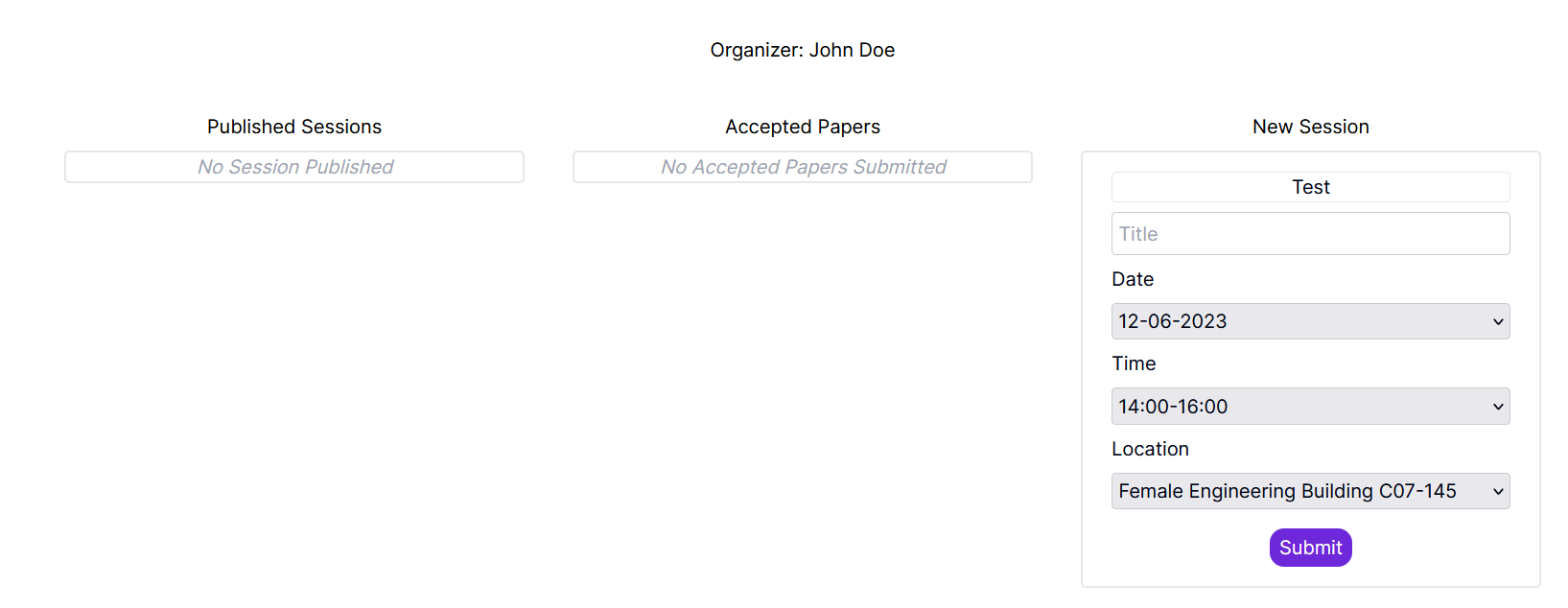
# Review paper



# Get conference schedule



# Edit conference schedule



# Conference Statistics Report

# Discussion of the project contribution of each team member

Contribution:

Abdulla 30%:

Frontend, actions, react, assisting.

Mohammed 25%:

Server-side, route, repo.

Youssef 25%:

Server-side, actions, seed, schema assistance.

Ahmed 20%:

ER diagram, schema model, seed assistance