Sri Lanka Institute of Information Technology

B.Sc. Honors Degree in Information Technology

Specialized in Software Engineering

Final Examination
Year 3, Semester 1 (2021)

SE3040 – Application Frameworks

Group Project

Technical Report

Group ID: 2021S1_REG_WD_22

Group Name: Team MERNs

Student ID	Student Name
IT19129204	Jayasekara R.T.R
IT19147024	Kariyawasam K.G.S.S.K
IT19121352	Kudarachchi K.A.N.D
IT19126234	Jayasinghe S.L

React app	https://github.com/rukshan99/app-ICAF
REST API	https://github.com/rukshan99/api-ICAF

For Evaluations:

Marks Allocation	Leader	Member 2	Member 3	Member 4	
User Interfaces look professional and consistency - 15					
Features are comprehensive and user friendly - 15					
Implementation of the RESTful web services - 15					
Implementing database and Mongo collections - 10					
Implementation of unit tests - 10					
Coding Standards and quality - 5					
Deployed in the cloud - 5					
Maintaining online git repository - 5					
Technical Report (Group Mark) - 5					
User guide (Group Mark) - 5					
Presentation - 10					

Comments:

Brief description of all the functions (member wise)

Jayasekara R.T.R – IT19129204

User sign up

Any guest user can use the sign up form to register to the system. First everyone need to add the basic user details in the form. Then if the frontend validation passes, the registering user can select the user type. Based on the selected user type, the next part of the sign up form changes. If the user selects the type as “Attendee”, credit card details are needed to be entered for the conference attendance payment. If the user selects the type as “Researcher” or “Workshop Presenter”, relevant document needed to be uploaded. User can also view the uploaded document from the sign up form. After completing the sign up form, user can click on the “Sign up” button and if the signing up successful, will be redirected to their user account.

User account

User accounts differ based on the user role. If the user is an “Attendee”, the entered contact details can be viewed. If the user is a “Researcher”, the entered contact details and the document can be viewed. If needed the document can also be downloaded. The status of the document is also displayed in the user account. The status can be either “pending”, “approved” or “rejected”. If only the status is “approved”, the researcher can proceed for making the payment for the research paper presentation. After making the payment the status will be displayed as “paid”. If the user is a “Workshop Presenter”, the entered contact details and the document can be viewed. Same as before, If needed the document can also be downloaded and the status of the document is also displayed as either “pending”, “approved” or “rejected”.

Role based authentication

Any user who has previously signed up can sign in to the system using the sign in form. After entering the credentials, if they are validated, users are redirected to the user profile. Only the users with role “Admin” can access the admin dashboard. Only the users with “Admin” and “Reviewer” roles can view the uploaded research papers and workshop presentations. Only the users with “Admin” and “Editor” roles can view the approved and un-approved conference detail submissions. Users with “Attendee”, “Researcher” or “Workshop Presenter” roles can only access their respective user accounts and the public pages which can also be accessed by the guest users. The public pages include a download page, where anyone can download sample research papers, sample workshop proposals, the research paper template and the workshop proposal template.

Kariyawasam K.G.S.S.K – IT19147024

Admin Dashboard

An admin can look summary of the system and he can get some details about main categories. So, admin can observe the total of Researchers, Works Shops and Attendees. So, admin can get summary details of Research Papers. There is a count of Pending, Approvals and Rejects so admin can get summary of workshops. There is a count of pending, Approvals, Rejects

An admin can get all conferences list, all presentation list and all workshops were added by editor so admin can give his approval for correct conference of the year. Admin can get the Approved Conference only and can check the details quickly.

Approval Conference Details

An admin have to give the approval of right conference details so admin can get details after clicking the conferences and admin can get brief details of that conference after he want to check the more details admin can switch to the more details after clicking more details button then admin can check all the details and admin can give the approval of that conference by clicking publish button then status field switch to the “Approved”.

Kudarachchi K.A.N.D – IT19121352

Editor

Editor is an admin of the system. He can manage conferences. First he have to login as the editor. There is an editor panel in editor page. He can manage conference details using that panel. If he clicked add conference, create conference page appears and he can add a conference. If he clicked workshops & presentations, he can add presentations and workshops to the selected conferences. He can also view all the conferences, presentations and workshops. He can update conferences too.

Home page

This is the page that appears first when a user access the website. User can also view upcoming conference details there.

Jayasinghe S.L – IT19126234

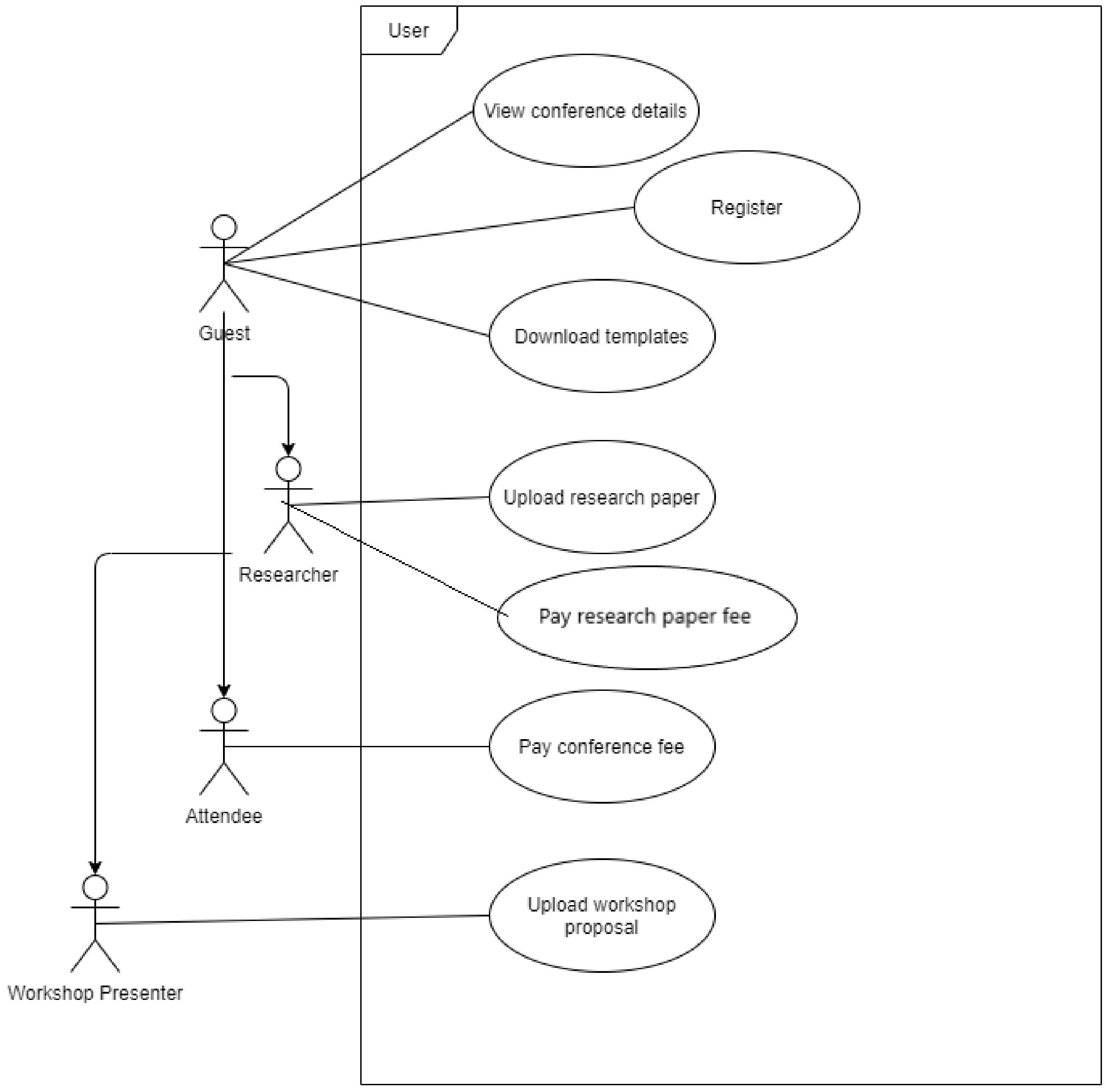
<u>View All Researchers and workshop presenters and Review a document</u>

The users with “Reviewer” roles can view list of user details which belongs to “Researcher” or “Workshop Presenter” roles and the uploaded research papers and workshop presentations and. Once reviewer click one user it will display the uploaded document details of that single user. Reviewer has ability to go through the uploaded document and grade it as “Accept” or “Reject”. If the reviewer click accept button the document status will update as “Accepted”, if reviewer click decline button the document status will update as “Rejected” and if not the document status will remains as “Pending”. Once reviewer Accept a research paper that particular researcher will be informed about the payment via sending an automatic email.

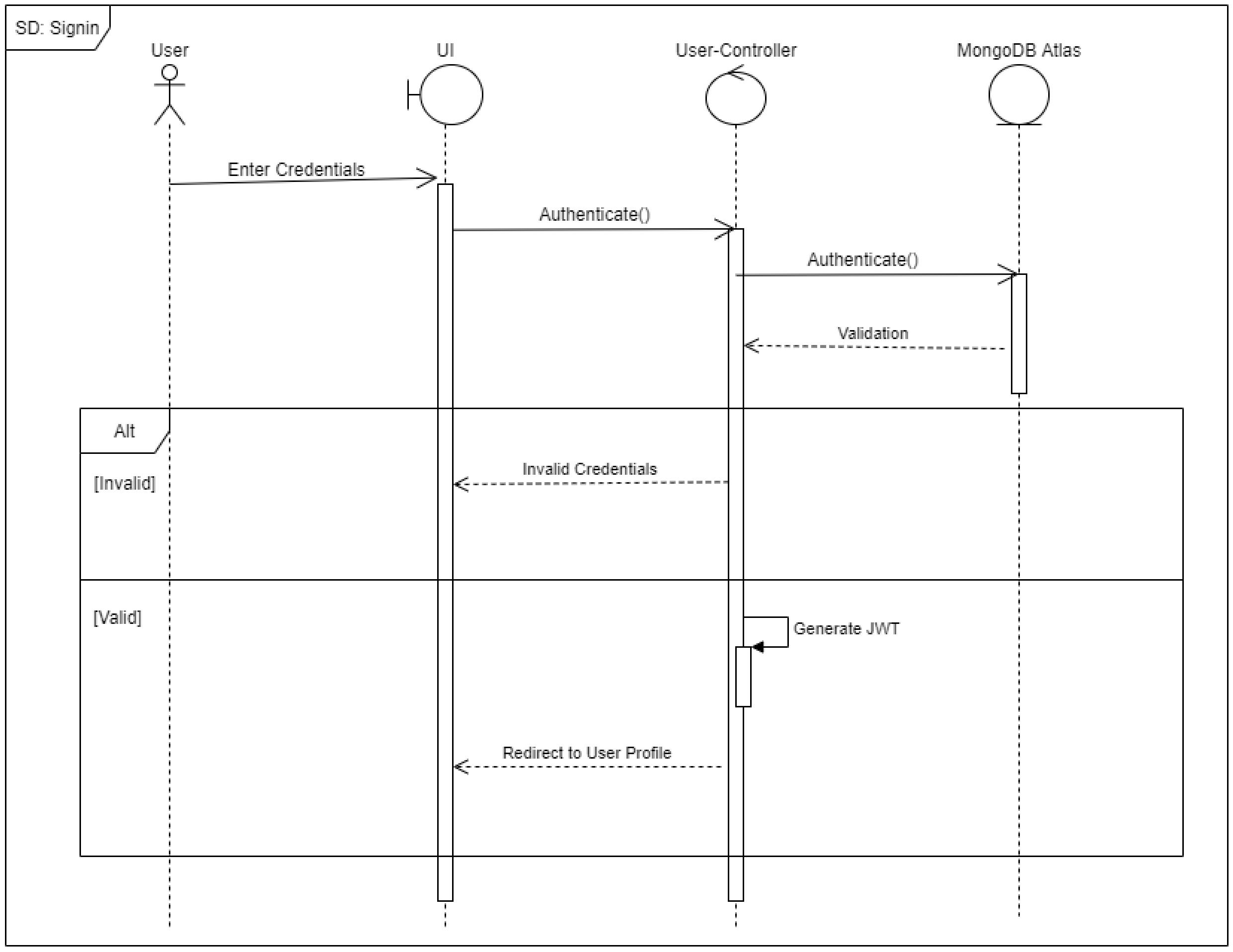
Use case Diagrams and the Component tree (member wise)

Jayasekara R.T.R – IT19129204

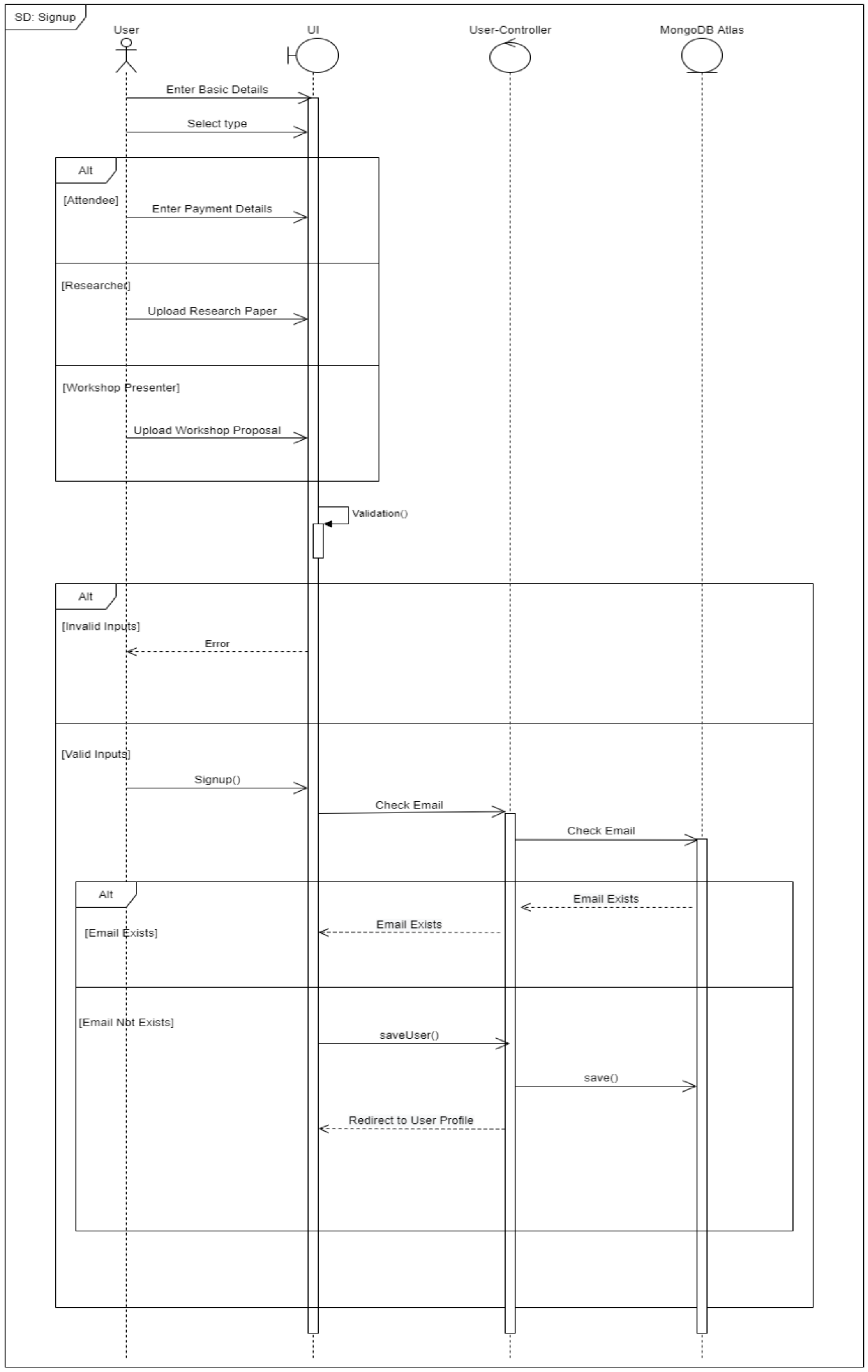
Authentication, authorization and user details handling use case



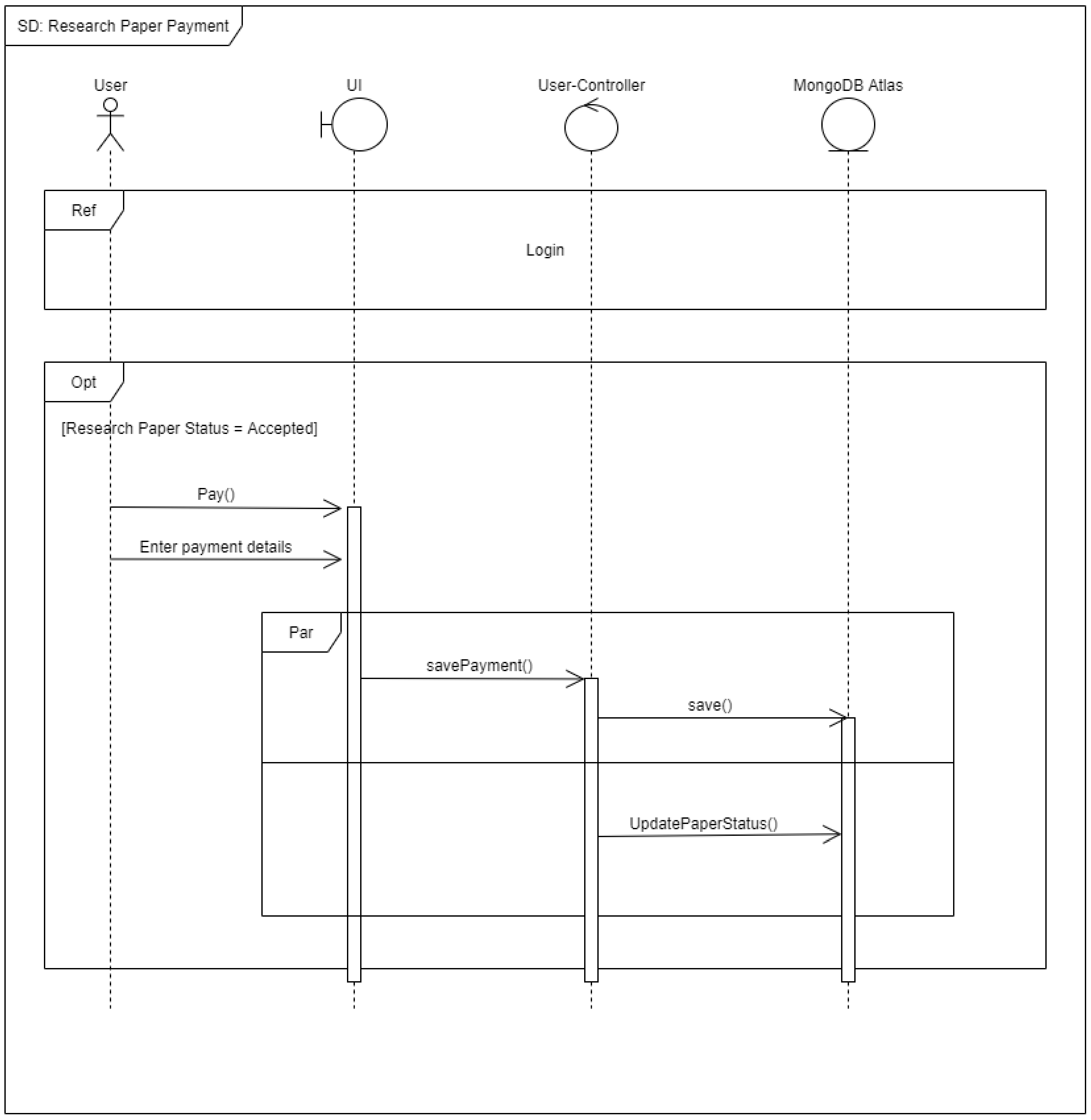
User Sign in


SD: Signin
User
UI
User-Controller
MongoDB Atlas
Enter Credentials
Authenticate()
Authenticate()
Validation
Alt
[Invalid]
Invalid Credentials
[Valid]
Generate JWT
Redirect to User Profile

User Sign up


SD: Signup
User
UI
User-Controller
MongoDB Atlas
Enter Basic Details
Select type
Alt
[Attendee]
Enter Payment Details
[Researcher]
Upload Research Paper
[Workshop Presenter]
Upload Workshop Proposal
Validation()
Alt
[Invalid Inputs]
Error
[Valid Inputs]
Signup()
Check Email
Check Email
Alt
Email Exists
[Email Exists]
Email Exists
[Email Not Exists]
saveUser()
save()
Redirect to User Profile

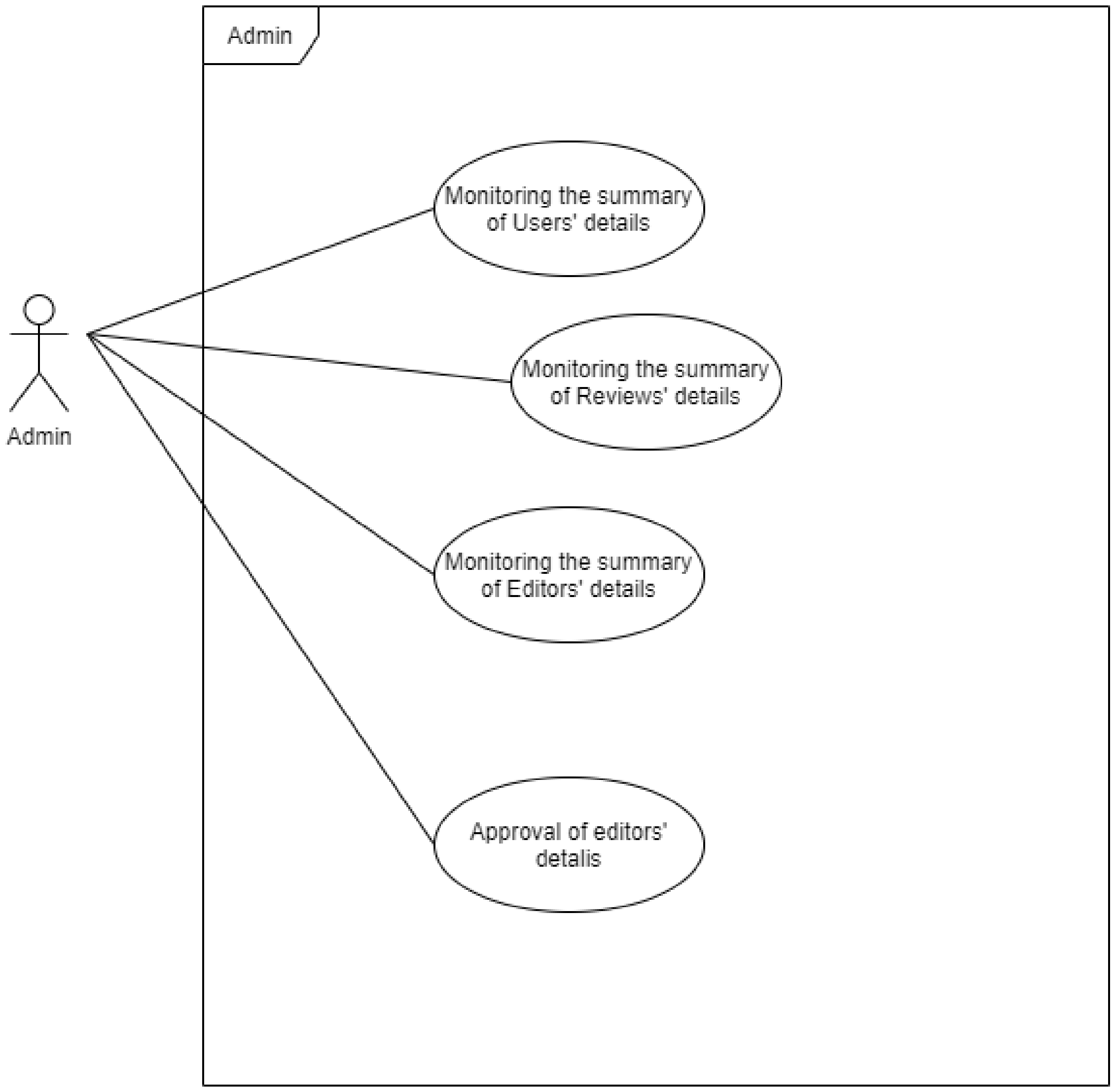
Research paper payment

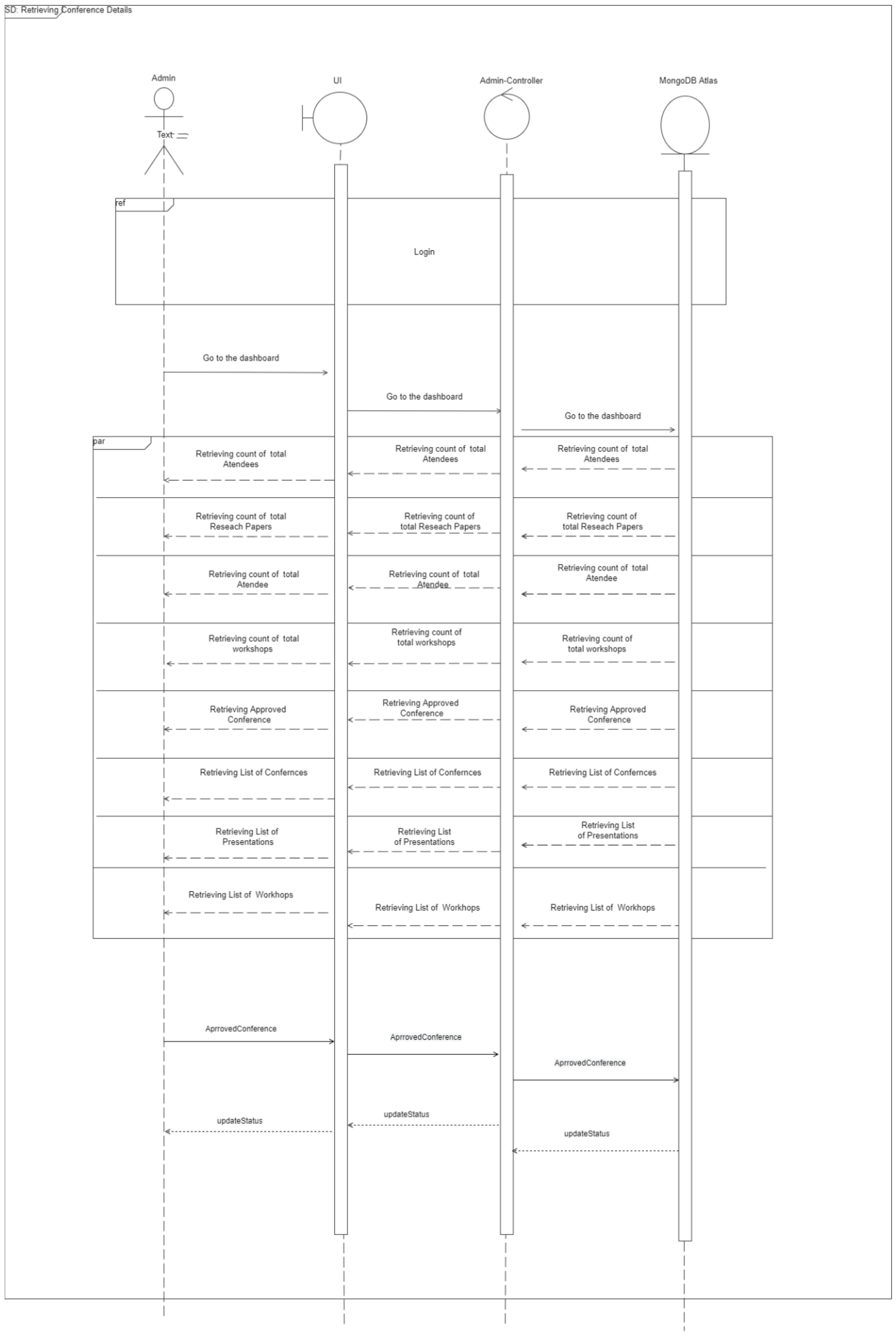


Component tree

- MainNavigation
  - BackDrop
  - SideDrawer
    - NavLinks
  - MainHeader
    - NavLinks
- Footer
- Downloads
  - Card
  - Card
  - Card
  - Card
- Profile
  - Card
    - LoadingSpinner
    - BankForm
      - ErrorModal
        - Modal
          - BackDrop
      - LoadingSpinner
- SignIn
  - ErrorModal
  - Card
    - LoadingSpinner
    - Input
    - Input
    - Input
    - DropdownButton
    - BankForm
      - ErrorModal
      - LoadingSpinner
    - DocumentUpload
      - FileBase
    - Button
    - Button

Kariyawasam K.G.S.S.K – IT19147024


Admin
Admin
Monitoring the summary of Users' details
Monitoring the summary of Reviews' details
Monitoring the summary of Editors' details
Approval of editors' detalis


SD: Retrieving Conference Details
Admin
UI
Admin-Controller
MongoDB Atlas
Text
ref
Login
Go to the dashboard
Go to the dashboard
Go to the dashboard
par
Retrieving count of total Atendees
Retrieving count of total Atendees
Retrieving count of total Atendees
Retrieving count of total Reseach Papers
Retrieving count of total Reseach Papers
Retrieving count of total Reseach Papers
Retrieving count of total Atendee
Retrieving count of total Atendee
Retrieving count of total Atendee
Retrieving count of total workshops
Retrieving count of total workshops
Retrieving count of total workshops
Retrieving Approved Conference
Retrieving Approved Conference
Retrieving Approved Conference
Retrieving List of Confernces
Retrieving List of Confernces
Retrieving List of Confernces
Retrieving List of Presentations
Retrieving List of Presentations
Retrieving List of Presentations
Retrieving List of Workhops
Retrieving List of Workhops
Retrieving List of Workhops
AprrovedConference
AprrovedConference
AprrovedConference
updateStatus
updateStatus
updateStatus

Component tree

- Dashboard
  - LoadingSpinner

- ConferenceDetails
  - loadingSpinner

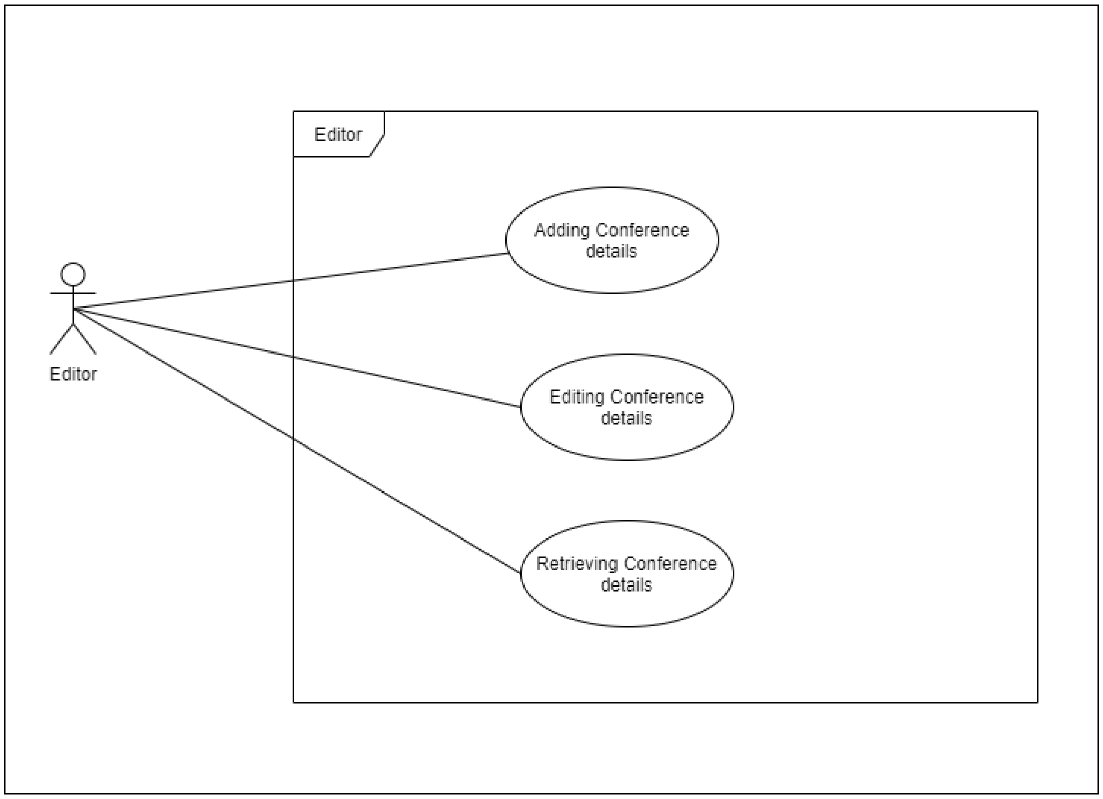
- ApprovedConferenceDetails
  - Card
    - loadingSpinner

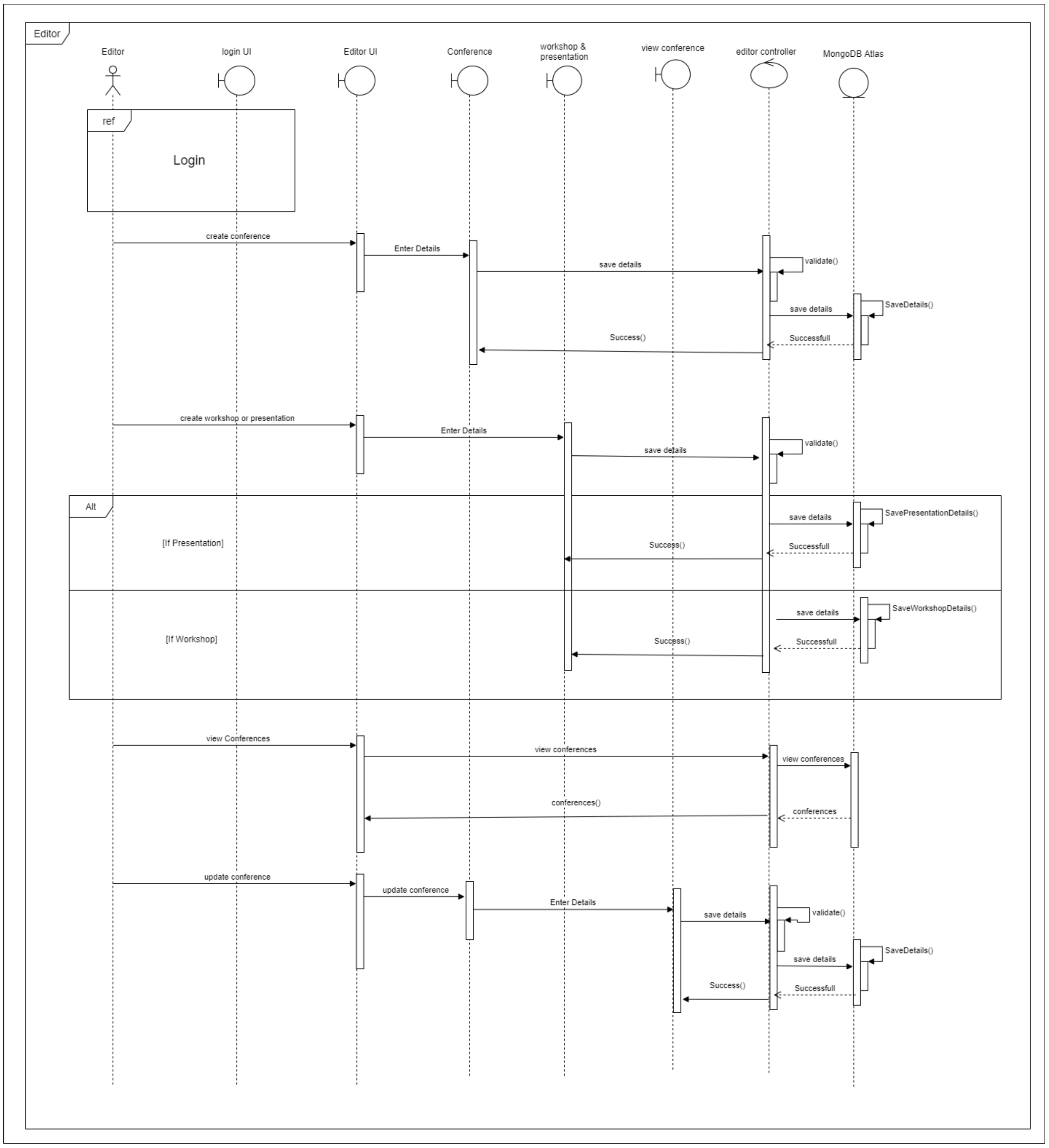
- PresentationsDetails
  - Card
    - loadingSpinner

- WorkshopsDetails
  - Card
    - loadingSpinner

Kudarachchi K.A.N.D – IT19121352

Managing conference details


Editor
Adding Conference details
Editing Conference details
Retrieving Conference details
Editor

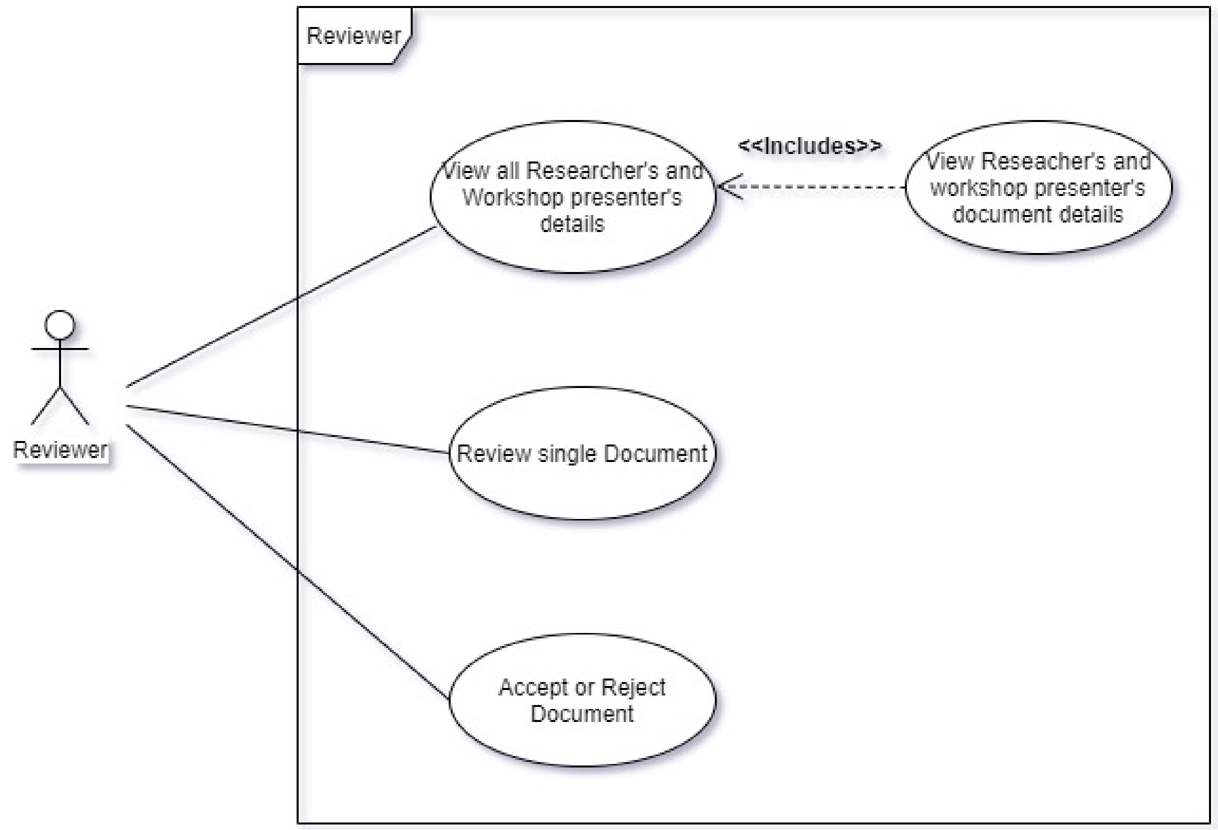

Editor
Editor
login UI
Editor UI
Conference
workshop & presentation
view conference
editor controller
MongoDB Atlas
ref
Login
create conference
Enter Details
save details
validate()
save details
SaveDetails()
Successfull
Success()
create workshop or presentation
Enter Details
save details
validate()
Alt
[If Presentation]
save details
SavePresentationDetails()
Success()
Successfull
[If Workshop]
save details
SaveWorkshopDetails()
Success()
Successfull
view Conferences
view conferences
view conferences
conferences()
conferences
update conference
update conference
Enter Details
save details
validate()
save details
SaveDetails()
Success()
Successfull

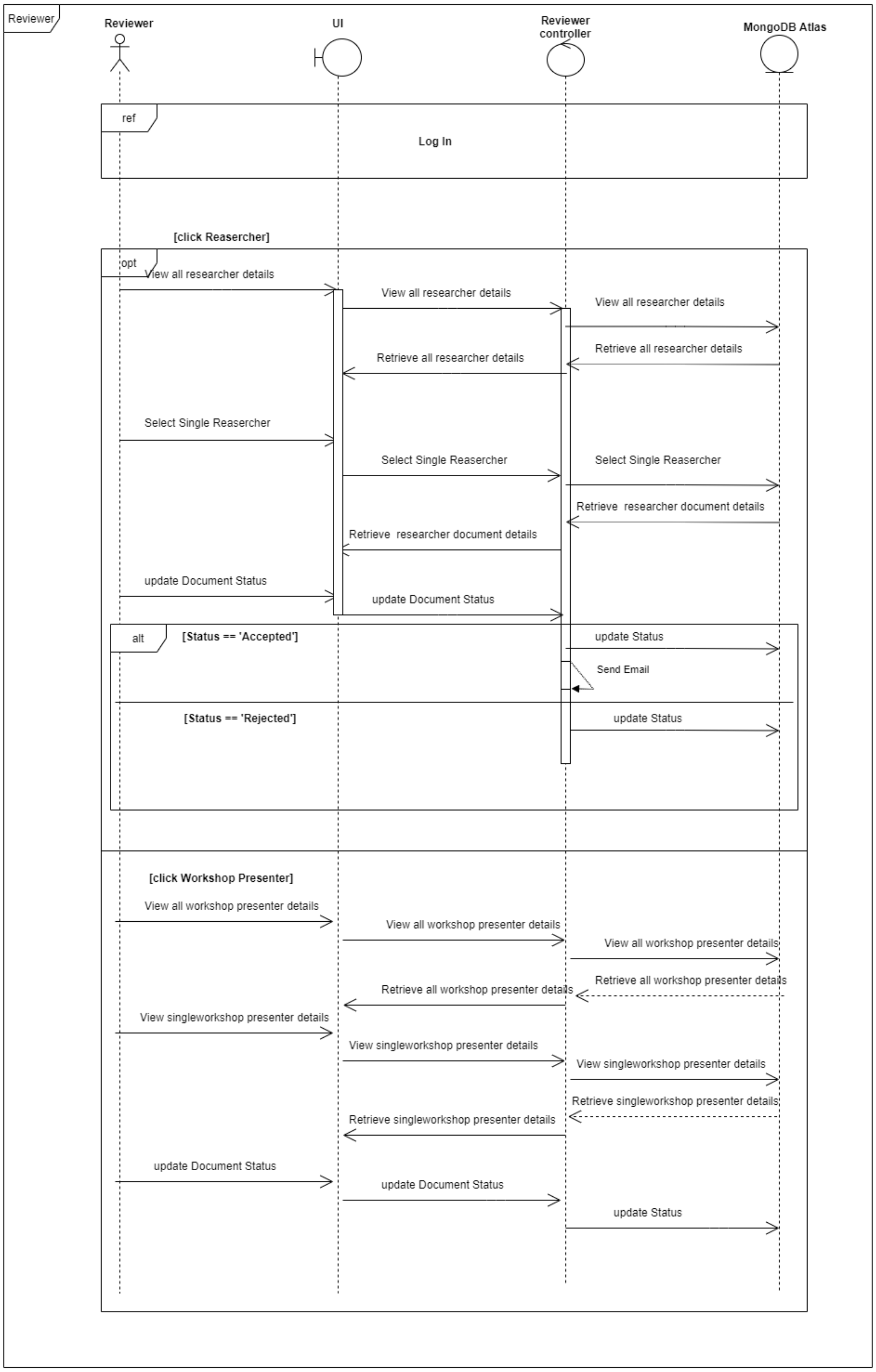
Component Tree

- Home
  - NavLinks
- Panel
  - NavLinks
- Conference
  - Loading-Spinner
  - Error-modal
- Presentation
  - Select
  - Loading-Spinner
  - Error-modal
- UpdateConference
  - Loading-Spinner
  - Error-modal
- ViewConferences
  - NavLinks
- ViewPresentations

Jayasinghe S.L – IT19126234

Researcher’s and workshop presenter’s document approval handling


Reviewer
View all Researcher's and Workshop presenter's details
<<Includes>>
View Reseacher's and workshop presenter's document details
Reviewer
Review single Document
Accept or Reject Document


Reviewer
Reviewer
UI
Reviewer controller
MongoDB Atlas
ref
Log In
[click Reasercher]
opt
View all researcher details
View all researcher details
View all researcher details
Retrieve all researcher details
Retrieve all researcher details
Select Single Reasercher
Select Single Reasercher
Select Single Reasercher
Retrieve researcher document details
Retrieve researcher document details
update Document Status
update Document Status
alt
[Status == 'Accepted']
update Status
Send Email
[Status == 'Rejected']
update Status
[click Workshop Presenter]
View all workshop presenter details
View all workshop presenter details
View all workshop presenter details
Retrieve all workshop presenter details
Retrieve all workshop presenter details
View singleworkshop presenter details
View singleworkshop presenter details
View singleworkshop presenter details
Retrieve singleworkshop presenter details
Retrieve singleworkshop presenter details
update Document Status
update Document Status
update Status

Component tree

- Reserchers
    - Card

        - LoadingSpinner
        - Button
        - Button

- Workshop Presenters
    - Card

        - LoadingSpinner
        - Button
        - Button

Usage of Rest APIs (member wise)

Jayasekara R.T.R – IT19129204

POST – http://localhost:4000/api/v1/users/auth

```
const authenticate = async ({ email, password }) => {
    let user = null;
    try{
        user = await User.findOne({ email: email });
      } catch(err) {
        const error = new HttpError(
          'Something went wrong, could not find user.',
          500
        );
        return error;
      }

      const match = await bcrypt.compare(password, user.password);
      if(match) {
        const token = jwt.sign({ sub: user.id, role: user.role }, config.secret);
        const { password, ...userWithoutPassword } = user;
        return {
            ...userWithoutPassword,
            token
        };
      } else {
        return res.json({success: false, message: 'passwords do not match'});
      }
}
```

This is the controller function for authenticating a user. First the email is checked with the database to verify the availability of the user. If the user is not registered to the system, an HTTP error is returned with the status code of 500. If the user is available, the hashed password is compared with the entered password using “compare” method of bcrypt. If the passwords are matching. A JSON Web Token (JWT) is created with the user details without the password and the token is returned with the response to the frontend. If the password do not match, a success failure massage is returned with the response.

POST - http://localhost:4000/api/v1/users/signup

```
const saveUser = async (req, res, next) => {
    console.log('Adding the user');
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        console.log(errors);
        return next(new HttpError('Invalid inputs! Please check again.', 422));
    }

    const { name, email, password, role, document, paymentForm } = req.body;

    let existingUser;
    try{
      existingUser = await User.findOne({ email: email});
    } catch(err) {
      const error = new HttpError(
        'Something went wrong, could not sign up.',
        500
      );
      return next(error);
    }

    if(existingUser) {
        const error = new HttpError(
          'User already exists, please sign in.',
          422
        );
        return next(error);
      }

    const hashedPassword = await bcrypt.hash(password, 10);
    console.log(hashedPassword);
    const newUser = new User({
        userid: uuid(),
        name,
        email,
        password: hashedPassword,
        role,
        document: document || {},
        payments: []
    });

    let newPayment = null;
    if(paymentForm) {
        newPayment = new Payment({
            payment_id: uuid(),
            amount: paymentForm.amount,
            paymentDate: new Date(),
            cardDetails: {
                cardNo: paymentForm.cardNo,
                expDate: paymentForm.expDate,
                cvv: paymentForm.cvv
            },
            userid: ''
        })
    }
    try {
        const session = await mongoose.startSession();
        session.startTransaction();
        if(paymentForm) {
            newUser.payments.push(newPayment);
            newPayment.userid = newUser._id;
            await newPayment.save({ session: session });
        }
        await newUser.save({ session: session });
        await session.commitTransaction();
    } catch (err) {
        const error = new HttpError(
            'Error occured while saving details. Please try again.',
            500
        );
        return next(error);
    }

    res.status(201).json({ User: saveUser });
};
```

This is the controller function for registering a user in the database. Initially, the request is validated for any errors. If there are error in the request, an error message is returned. Then the user entered data is extracted from the request body using object destructuring. After that the email is checked with the database data to verify that it has not been used by another user previously. If a user exists with that email, a HTTP error is returned with the status code of 422. If the email is unique, next the entered password is encrypted using bcrypt. After that, an object of the mongoose “User” schema is created with the relevant data. Next, an object of the mongoose “Payment” schema should be created based on the user role. It can be easily checked with the “paymentForm” data in the request body. If there are data in the “paymentForm” the “Payment” schema object is created. Then a mongoose transaction is started with a mongoose session and all the data is saved to the database according to the relationship between the two schemas. If there is an error while saving data, an aero is returned with the status code of 500.

POST - http://localhost:4000/api/v1/users/pay

```
const savePayment = async (req, res, next) => {

  const { userid, paymentForm } = req.body;
  let existingUser = null;
  try {
    existingUser = await User.findOne({ _id: userid});
  } catch(err) {
    const error = new HttpError(
      'Something went wrong, could not save details.',
      500
    );
    return next(error);
  }
  if(existingUser) {
    const newPayment = new Payment({
      payment_id: uuid(),
      amount: paymentForm.amount,
      paymentDate: new Date(),
      cardDetails: {
          cardNo: paymentForm.cardNo,
          expDate: paymentForm.expDate,
          cvv: paymentForm.cvv
      },
      userid
    })

    try {
      const session = await mongoose.startSession();
      session.startTransaction();
      await newPayment.save({ session: session });
      await User.updateOne({_id: userid}, { $push: { payments: newPayment._id } }
,(err, res) => {});
      await session.commitTransaction();
    } catch (err) {
      const error = new HttpError(
          'Error occured while saving details. Please try again.',
          500
      );
      return next(error);
    }
    let updatedUser = null;
    try {
      updatedUser = await User.findOne({ _id: userid});
    } catch(err) {
      const error = new HttpError(
        'Something went wrong..',
        500
    );
      return next(error);
    }
    res.status(201).json({ "user": updatedUser });
  }

}
```

This is the controller function for saving payment details. This is used in the use case of the payment done by a researcher when the submitted research paper is approved by a reviewer. First the required information is extracted from the request body by using object destructuring. Then, the database is checked for the existing user using the mongoose “findOne” method with the user id. After the user is found, an object of “Payment” mongoose schema is created. After that a mongoose transaction is started with a mongoose session and the payment details are saved in the database. Also the relevant user’s payment array is updated with the new payment id.

Below are the API end points to download templates and sample documents of the conference.

GET - http://localhost:4000/api/v1/users/downloads/template-research-paper

GET - http://localhost:4000/api/v1/users/downloads/template-workshop-proposal

GET - http://localhost:4000/api/v1/users/downloads/sample-research-paper

GET - http://localhost:4000/api/v1/users/downloads/sample-workshop-proposal

Kariyawasam K.G.S.S.K – IT19147024

GET- http://localhost:4000/api/v1/admin/count

```
const countRole = async (req, res) => {

        let userList = null;
        try{
            userList = await Admin.find({}, function(err, result) {
                if (err) {
                  console.log(err);
                } else {

                }
              });
        } catch(err) {
            return err;
        }

        let totalroleAttendee = 0;
        let totalroleResearcher = 0;
        let totalroleWorkshopPresenter = 0;
        let totalRejectedResearchPapers = 0;
        let totalAcceptedResearchPapers = 0;
        let totalPendingResearchPapers = 0;
        let totalAcceptedWorkshop = 0;
        let totalRejectedWorkshop = 0;
        let totalPendingWorkshop = 0;

        userList.map(user => {
            if(user.role === 'Attendee') {
                totalroleAttendee++;
            }
            else if(user.role === 'Researcher') {
                totalroleResearcher++;
                if(user.document.docStatus === 'Rejected') {
                    totalRejectedResearchPapers++;
                }
                else if(user.document.docStatus === 'Accepted') {
                    totalAcceptedResearchPapers++;
                }
                else if(user.document.docStatus === 'Pending') {
                    totalPendingResearchPapers++;
                }
            }
            else if(user.role === 'Workshop Presenter'){
                totalroleWorkshopPresenter++;
                if(user.document.docStatus === 'Rejected') {
                    totalRejectedWorkshop++;
                }
                else if(user.document.docStatus === 'Accepted') {
                    totalAcceptedWorkshop++;
                }
                else if(user.document.docStatus === 'Pending') {
                    totalPendingWorkshop++;
                }
            }

        });



        res.status(200).send({
            totalroleAttendee: totalroleAttendee,
            totalroleResearcher: totalroleResearcher,
            totalroleWorkshopPresenter: totalroleWorkshopPresenter,
            totalAcceptedResearchPapers: totalAcceptedResearchPapers,
            totalRejectedResearchPapers: totalRejectedResearchPapers,
            totalPendingResearchPapers: totalPendingResearchPapers,
            totalRejectedWorkshop: totalRejectedWorkshop,
            totalAcceptedWorkshop: totalAcceptedWorkshop,
            totalPendingWorkshop: totalPendingWorkshop
        });

}
```

Using this end point system can get the count of Attendees, Researchers, workshop presenters so system can get total pending, Rejects, Approvals of research papers and Work shops.

GET- http://localhost:4000/api/v1/admin/

```
const getAllConference = async (req, res) => {
    await Conference.find()
    .then(data => {
        res.status(200).send({ data: data});
    })
    .catch(error => {
        res.status(500).send({ error: error.message});
    })
}
```

Using this end point system can get the all details of conference details.

GET- http://localhost:4000/api/v1/admin/:id

```
const getOneConference = async (req, res) =>{
    const id = req.params.id;

    Conference.findById(id)
    .then(data => {
        if (!data)
            res.status(404).send({message: "not found Conference id " + id});
            else res.send(data);
    })
    .catch(err => {
        res
          .status(500)
          .send({ message: "Error retrieving Conference with id=" + id });
      });
}
```

Using this end point system can get only one conference details.

PUT- http://localhost:4000/api/v1/admin/:id

```
const updateconference = async (req, res) => {
    if (!req.body) {
      return res.status(400).send({
        message: "Data to update can not be empty!"
      });
    }

    const id = req.params.id;

    Conference.findByIdAndUpdate(id, req.body, { useFindAndModify: false })
      .then(data => {
        if (!data) {
          res.status(404).send({
            message: `Cannot update Conference with id=${id}. Maybe Conference wa
s not found!`
          });
        } else res.send({ message: "Conference was updated successfully." });
      })
      .catch(err => {
        res.status(500).send({
          message: "Error updating Conference with id=" + id
        });
      });
  };
```

Using this end point system can update the status without updating other fields.

GET- http://localhost:4000/api/v1/admin/presentation/all

```
const getAllPresentations = async (req, res) => {
    await Presentation.find()
    .then(data => {
        res.status(200).send({ data: data});
    })
    .catch(error => {
        res.status(500).send({ error: error.message});
    })
  }
```

Using this end point system can get the all presentation details to frontend.

GET- http://localhost:4000/api/v1/admin/workshop/all

```
const getAllWorkshops = async (req, res) => {
    await Workshop.find({})
    .then(data => {
        res.status(200).send({ data: data});
    })
    .catch(error => {
        res.status(500).send({ error: error.message});
    })
  }
```

Using this endpoint system can get all workshops to frontend

GET- http://localhost:4000/api/v1/admin/approved/one

```
  const findPublishedConference = (req, res) => {
      Conference.find({status: true})
      .then(data => {
        res.status(200).send({ data: data});
      })
      .catch(err => {
        res.status(500).send({
          message:
            err.message || "Some error occurred while retrieving Conferences."
        });
    });
}
```

Using this endpoint system can get only the approved conference to endpoint

GET- http://localhost:4000/api/v1/admin/presentations/:id

```
const getPresentationsForConference = async (req, res) => {
    if (req.params && req.params.id){
        await Conference.findById(req.params.id)
        .populate('presentation', 'topic description starttime endtime presenter'
)
        .then(data => {
            res.status(200).send({ data: data.presentation});
        })
        .catch(error => {
            res.status(500).send({ error: error.message});
        });
    }
}
```

Using this endpoint system can get presentations details related conference to frontend.

GET- http://localhost:4000/api/v1/admin/workshops/:id

```
const getWorkshopForConference = async (req, res) => {
    if (req.params && req.params.id){
        await Conference.findById(req.params.id)
        .populate('workshop', 'topic description starttime endtime presenter')
        .then(data => {
            res.status(200).send({ data: data.workshop});
        })
        .catch(error => {
            res.status(500).send({ error: error.message});
        });
    }
}
```

Using this endpoint system can get workshops details related conference to frontend.

Kudarachchi K.A.N.D – IT19121352

POST – http://localhost:4000/api/v1/editor/conference

```
const addingConference = async (req, res, next) => {
  if (req.body) {
    const workshop = new conference(req.body);
    await workshop.save()
      .then(data => {
        res.status(200).send({ data: data });
      })
      .catch(error => {
        res.status(500).send({ error: error.message });
      });
  }
}
```

This is the controller function for saving conference details to the database. Initially, the request is validated for any errors. If there are error in the request, an error message is returned. Then the user entered data is extracted from the request body.

POST – http://localhost:4000/api/v1/editor/presentation

```
const createPresentation = async (req, res) => {
    if (req.body) {
        const { conference } = req.body;

        const presentation = new Presentation(req.body);

        try {
            const data = await presentation.save();
            await Conference.updateOne({ _id: conference }, { $addToSet: { presen
tation: presentation._id } }, (err, res) => { });
            res.status(200).send({ data: data });
        } catch (error) {
            console.error(error)
        }


    }
}
```

This is the controller function for saving presentation details to the database. Initially, the request is validated for any errors. If there are error in the request, an error message is returned. Then the user entered data is extracted from the request body using object destructuring.

POST – http://localhost:4000/api/v1/editor/workshop

```
const createWorkshop = async (req, res) => {
    if (req.body) {
        const { conference } = req.body;

        const workshop = new Workshop(req.body);

        try {
            const data = await workshop.save();
            await Conference.updateOne({ _id: conference }, { $addToSet: { worksh
op: workshop._id } }, (err, res) => { });
            res.status(200).send({ data: data });
        } catch (error) {
            console.error(error)
        }


    }
```

This is the controller function for saving workshop details to the database. Initially, the request is validated for any errors. If there are error in the request, an error message is returned. Then the user entered data is extracted from the request body using object destructuring.

GET - http://localhost:4000/api/v1/editor/conference/

```
const getAllConference = async (req, res) => {
  await conference.find({})
    .then(data => {
      res.status(200).send({ data: data });
    })
    .catch(error => {
      res.status(500).send({ error: error.message });
    });
}
```

This is the controller function for retrieving all conference details from the database. If there are error, an error message is returned.

GET - http://localhost:4000/api/v1/editor/conference/:id

```
const getSingleConference = async (req, res) => {
  await conference.findById(req.params.id)
    .then(data => {
      res.status(200).send({ data: data });
    })
    .catch(error => {
      res.status(500).send({ error: error.message });
    });
}
```

This is the controller function for retrieving a single conference detail from the database. If there are error, an error message is returned.

GET - http://localhost:4000/api/v1/editor/presentation/:id

```
const getPresentationForConference = async (req, res) => {
  if (req.params && req.params.id) {
    await conference.findById(req.params.id)
      .populate('presentation', 'topic description starttime endtime presenter')
      .then(data => {
        res.status(200).send({ presentation: data.presentation });
      })
      .catch(error => {
        res.status(500).send({ error: error.message });
      });
  }
}
```

This is the controller function for retrieving presentation details according to a conference from the database. If there are error, an error message is returned.

GET - http://localhost:4000/api/v1/editor/presentation/single/:id

```
const getSinglePresentation = async (req, res) => {
    await Presentation.findById(req.params.id)
    .then(data => {
      res.status(200).send({ data: data });
    })
    .catch(error => {
      res.status(500).send({ error: error.message });
    });
  }
```

This is the controller function for retrieving a single presentation detail from the database. If there are error, an error message is returned.

GET - http://localhost:4000/api/v1/editor/workshop/:id

```
const getWorkshopForConference = async (req, res) => {
  if (req.params && req.params.id) {
    await conference.findById(req.params.id)
      .populate('workshop', 'topic description starttime endtime presenter')
      .then(data => {
        res.status(200).send({ workshop: data.workshop });
      })
      .catch(error => {
        res.status(500).send({ error: error.message });
      });
  }
}
```

This is the controller function for retrieving workshop details according to a conference from the database. If there are error, an error message is returned.

GET - http://localhost:4000/api/v1/editor/workshop /single/:id

```
const getSingleWorkshop = async (req, res) => {
    await Workshop.findById(req.params.id)
    .then(data => {
      res.status(200).send({ data: data });
    })
    .catch(error => {
      res.status(500).send({ error: error.message });
    });
  }
```

This is the controller function for retrieving a single workshop detail from the database. If there are error, an error message is returned.

PUT - http://localhost:4000/api/v1/editor/conference/:id

```
const updateConference = async (req, res) => {
  console.log(req.body)
  if(!req.body){
    return res.status(400).send({
      message: "Data to update can not be empty!"
    });
  }
    await conference.findByIdAndUpdate(req.params.id,req.body,{useFindAndModify :
 false})
      .then(data => {
        if(!data){
          res.status(400).send({ message: 'cannot update conference' });
        }else res.status(200).send({ data: data });
      })
      .catch(error => {
        res.status(500).send({ error: error.message });
      });

}
```

This is the controller function for updating a single conference detail to the database. Initially, the request is validated for any errors. If there are error in the request, an error message is returned. Then the user entered data is extracted from the request body.

Jayasinghe S.L – IT19126234

GET – http://localhost:4000/api/v1/reviewer/reserchers

```
exports.findAllReseachers = (req, res) => {
  User.find({ role: 'Researcher' })
    .then(data => {
      if (!data)
        res.status(404).send({ message: "Not found user with Reseacher " });
      else res.send({ data: data });
    })
    .catch(err => {
      res.status(500).send({ message: err.message || "Error retrieving
user with researcher role" });
    });
};
```

This is the function which retrieving all user details which are belongs to “Researcher” role by checking whether the user role is equal to “Researcher”. If there is no existing user with role Researcher error message will be displayed. If not all user details which are under the condition will be display to the reviewer.

GET – http://localhost:4000/api/v1/reviewer/presenters

```
exports.findAllWorkshopPresenters = (req, res) => {
  User.find({ role: 'Workshop Presenter' })
    .then(data => {
      if (!data)
        res.status(404).send({ message: "Not found user with workshop presenter "
 });
      else res.send({ data: data });
    })
    .catch(err => {
      res .status(500) .send({ message: "Error retrieving user details with works
hop presenter role" });
    });
};
```

This is the a function which retrieving all user details which are belongs to “Workshop Presenters” role by checking whether the user role is equal to “Workshop Presenters”. If there is no existing user with role Workshop Presenters error message will be displayed. If not all user details which are under the condition will be display to the reviewer.

GET – http://localhost:4000/api/v1/reviewer/user/:id

```
exports.getUserById = async (req, res) => {
  if (req.params && req.params.id) {
    await User.findById(req.params.id)
      .then(data => {
        res.status(200).send({ data: data.document });
      })
      .catch(error => {
        res.status(500).send({ error: error.message });
      });
  }
};
```

This function helps to retrieve a single user's document details by passing the user ID. First of all it will check the given user id is valid or not if it is valid document data will be find by given ID and retrieve the matching document details which are under given user ID.

PUT – http://localhost:4000/api/v1/reviewer/resercher/update/:id

```
exports.updateReasercherDocStatus = async (req, res) => {
  await User.findById(req.body.RequestID)
    .then(data => {
      User.updateOne({ _id: req.body.RequestID }, { payments:data.payments,userid
:data.userid,name:data.name,email:data.email,password:data.password,role:data.rol
e, document: { docName: data.document.docName, docData: data.document.docData, do
cStatus: req.body.Status } }, function (err, docs) {
        if (!err) {
          if(req.body.Status == "Accepted"){
            var transporter = nodemailer.createTransport({
              service: 'gmail',
              auth: {
                user: 'icafsliit9@gmail.com',
                pass: 'icaf12345*'
              }
            });
            transporter.use('compile',hbs({
                viewEngine:'express-handlebars',
                viewPath:'./views'
            }))
            var mailOptions = {
              from: 'icafsliit9@gmail.com',
              to:data.email,
              subject: 'Make your payment',
              text: 'Your Research paper has been approved.',
              template:'index'
            };
            transporter.sendMail(mailOptions, function(error, info){
              if (error) {
                console.log(error);
              } else {
                console.log('Email sent: ' + info.response);
              }
            });
          }
        } else {
          console.log("Error")
        }
      })
    })
    .catch(error => {
      console.log(error)
      res.status(500).send({ error: error.message });
    });
  };
```
This is an update function which helps to update the document status of a research paper by passing user ID. In here it is find out user by ID and update only the document States. If given status is “Accepted”, automatic email will be send to that user email to inform the payment details.

PUT – http://localhost:4000/api/v1/reviewer/presenter/update/:id

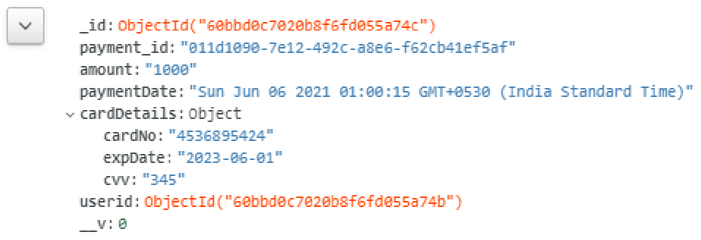
```
exports.updateReasercherDocStatus = async (req, res) => {
  await User.findById(req.body.RequestID)
    .then(data => {
      User.updateOne({ _id: req.body.RequestID }, { payments:data.payments,userid:data.userid,name:data.name,email:data.email,password:data.password,role:data.role, document: { docName: data.document.docName, docData: data.document.docData, docStatus: req.body.Status } }, function (err, docs) {
        if (!err) {
          console.log("updated successfully");
        } else {
          console.log("Error");
        }
      })
    })
    .catch(error => {
      console.log(error)
      res.status(500).send({ error: error.message });
    });
}
```

This is an update function which helps to update the document status of a Workshop presenter's document by passing user ID. In here it is find out user by ID and update only the document States.

Screen shots of Mongo DB Collections (member wise)

Jayasekara R.T.R – IT19129204

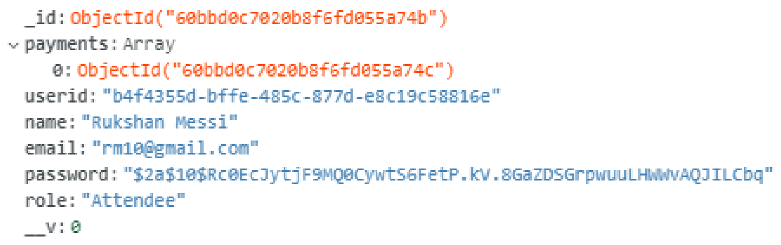
Payments collection sample document



```
_id: ObjectId("60bbd0c7020b8f6fd055a74c")
payment_id: "011d1090-7e12-492c-a8e6-f62cb41ef5af"
amount: "1000"
paymentDate: "Sun Jun 06 2021 01:00:15 GMT+0530 (India Standard Time)"
cardDetails: Object
   cardNo: "4536895424"
   expDate: "2023-06-01"
   cvv: "345"
userid: ObjectId("60bbd0c7020b8f6fd055a74b")
__v: 0
```

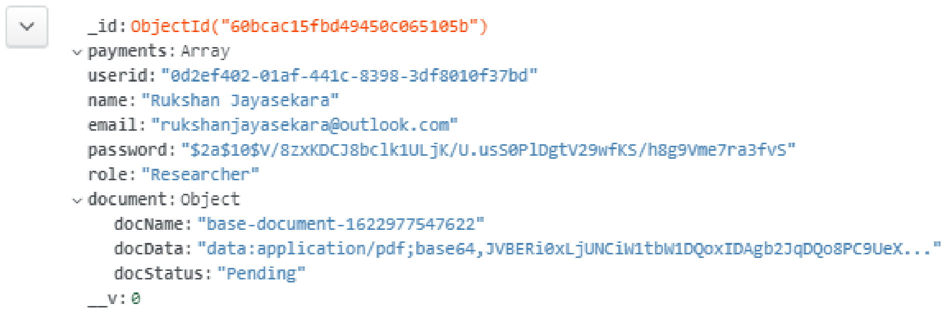
User collection sample documents

Attendee:



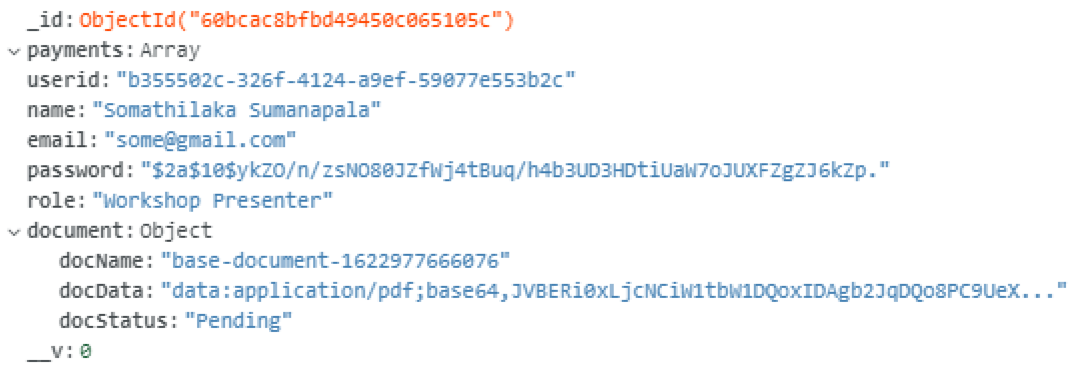
```
_id: ObjectId("60bbd0c7020b8f6fd055a74b")
payments: Array
   0: ObjectId("60bbd0c7020b8f6fd055a74c")
userid: "b4f4355d-bffe-485c-877d-e8c19c58816e"
name: "Rukshan Messi"
email: "rm10@gmail.com"
password: "$2a$10$Rc0EcJytjF9MQ0CywtS6FetP.kV.8GaZDSGrpwuuLHWWvAQJILCbq"
role: "Attendee"
__v: 0
```

Researcher:



```
_id: ObjectId("60bcac15fbd49450c065105b")
payments: Array
userid: "0d2ef402-01af-441c-8398-3df8010f37bd"
name: "Rukshan Jayasekara"
email: "rukshanjayasekara@outlook.com"
password: "$2a$10$V/8zxKDCJ8bclk1ULjK/U.usS0PlDgtV29wfKS/h8g9Vme7ra3fvS"
role: "Researcher"
document: Object
   docName: "base-document-1622977547622"
   docData: "data:application/pdf;base64,JVBERi0xLjUNCiW1tbW1DQoxIDAgb2JqDQo8PC9UeX..."
   docStatus: "Pending"
__v: 0
```

Workshop Presenter:

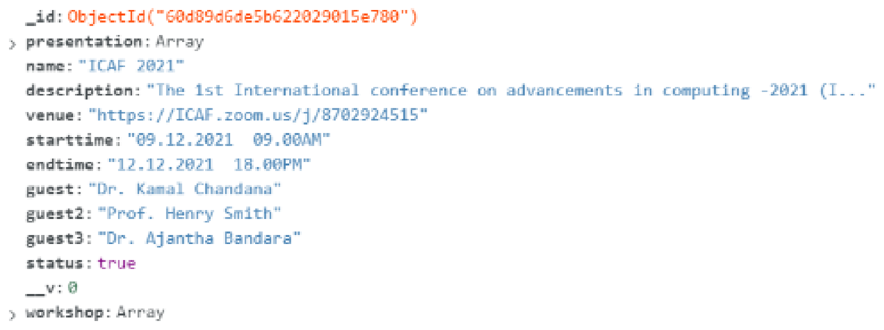


```
  _id: ObjectId("60bcac8bfbd49450c065105c")
v payments: Array
  userid: "b355502c-326f-4124-a9ef-59077e553b2c"
  name: "Somathilaka Sumanapala"
  email: "some@gmail.com"
  password: "$2a$10$ykZO/n/zsNO80JZfWj4tBuq/h4b3UD3HDtiUaW7oJUXFZgZJ6kZp."
  role: "Workshop Presenter"
v document: Object
     docName: "base-document-1622977666076"
     docData: "data:application/pdf;base64,JVBERi0xLjcNCiW1tbW1DQoxIDAgb2JqDQo8PC9UeX..."
     docStatus: "Pending"
  __v: 0
```

Kariyawasam K.G.S.S.K – IT19147024

Conference Collection

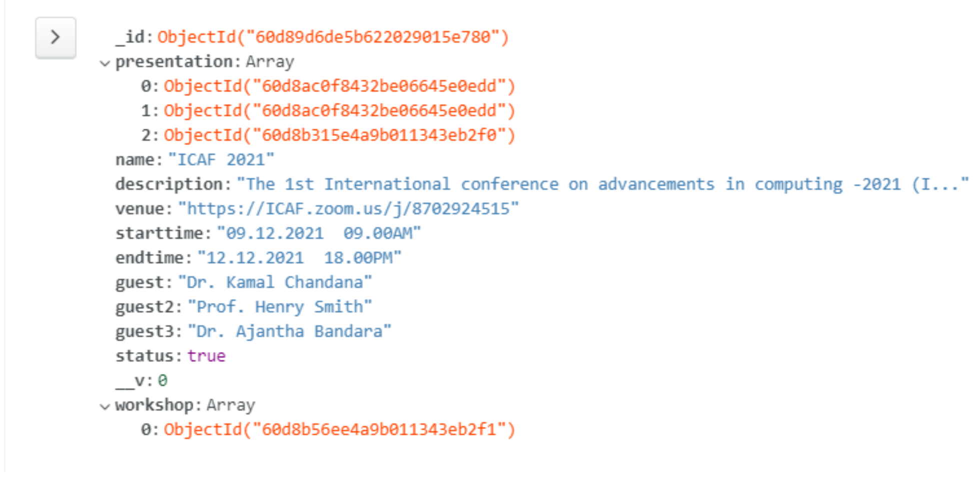
As an admin, he has to approve the Conference details so Admin can retrieve the details and after approving, have to update the status field “False” to “True”.



```
  _id: ObjectId("60d89d6de5b622029015e780")
> presentation: Array
  name: "ICAF 2021"
  description: "The 1st International conference on advancements in computing -2021 (I..."
  venue: "https://ICAF.zoom.us/j/8702924515"
  starttime: "09.12.2021  09.00AM"
  endtime: "12.12.2021  18.00PM"
  guest: "Dr. Kamal Chandana"
  guest2: "Prof. Henry Smith"
  guest3: "Dr. Ajantha Bandara"
  status: true
  __v: 0
> workshop: Array
```

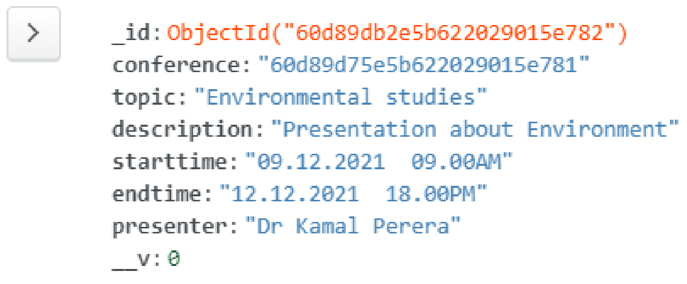
Kudarachchi K.A.N.D – IT19121352

Conferences Collection



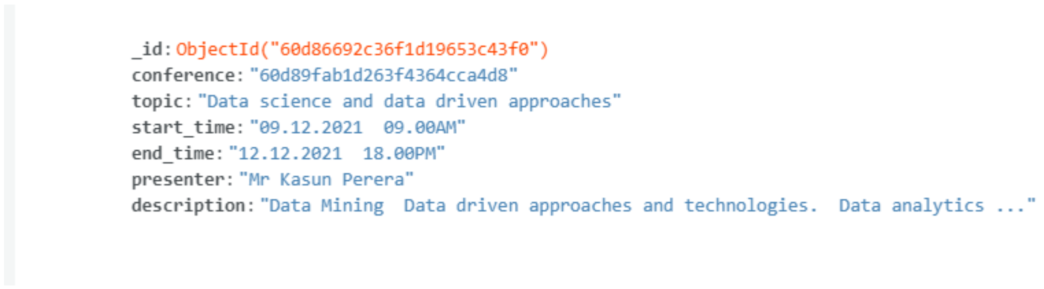
```
_id: ObjectId("60d89d6de5b622029015e780")
⌄ presentation: Array
    0: ObjectId("60d8ac0f8432be06645e0edd")
    1: ObjectId("60d8ac0f8432be06645e0edd")
    2: ObjectId("60d8b315e4a9b011343eb2f0")
  name: "ICAF 2021"
  description: "The 1st International conference on advancements in computing -2021 (I..."
  venue: "https://ICAF.zoom.us/j/8702924515"
  starttime: "09.12.2021  09.00AM"
  endtime: "12.12.2021  18.00PM"
  guest: "Dr. Kamal Chandana"
  guest2: "Prof. Henry Smith"
  guest3: "Dr. Ajantha Bandara"
  status: true
  __v: 0
⌄ workshop: Array
    0: ObjectId("60d8b56ee4a9b011343eb2f1")
```

Presentations Collection:



```
_id: ObjectId("60d89db2e5b622029015e782")
conference: "60d89d75e5b622029015e781"
topic: "Environmental studies"
description: "Presentation about Environment"
starttime: "09.12.2021  09.00AM"
endtime: "12.12.2021  18.00PM"
presenter: "Dr Kamal Perera"
__v: 0
```

Workshops Collection:

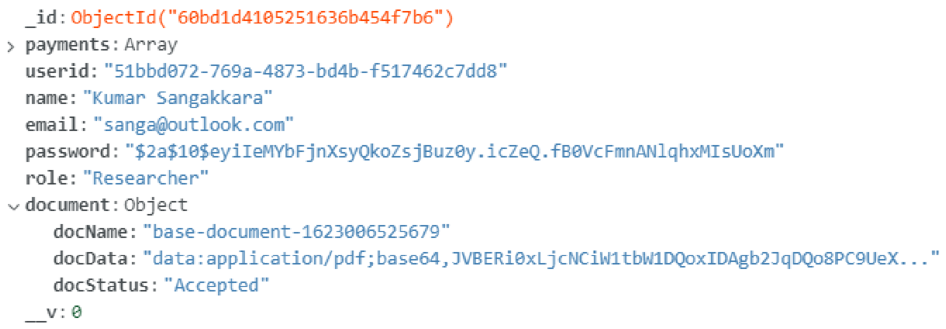


```
_id: ObjectId("60d86692c36f1d19653c43f0")
conference: "60d89fab1d263f4364cca4d8"
topic: "Data science and data driven approaches"
start_time: "09.12.2021  09.00AM"
end_time: "12.12.2021  18.00PM"
presenter: "Mr Kasun Perera"
description: "Data Mining  Data driven approaches and technologies.  Data analytics ..."
```

Jayasinghe S.L– IT19126234

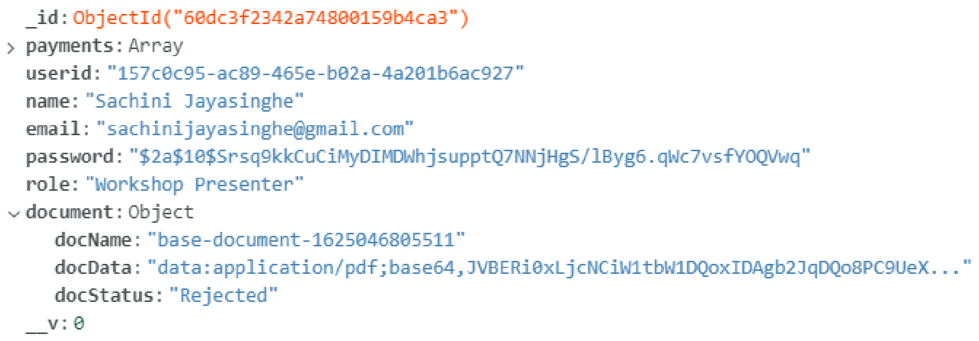
Update docStatus in document object

Researcher :



```
  _id: ObjectId("60bd1d4105251636b454f7b6")
> payments: Array
  userid: "51bbd072-769a-4873-bd4b-f517462c7dd8"
  name: "Kumar Sangakkara"
  email: "sanga@outlook.com"
  password: "$2a$10$eyiIeMYbFjnXsyQkoZsjBuz0y.icZeQ.fB0VcFmnANlqhxMIsUoXm"
  role: "Researcher"
v document: Object
    docName: "base-document-1623006525679"
    docData: "data:application/pdf;base64,JVBERi0xLjcNCiW1tbW1DQoxIDAgb2JqDQo8PC9UeX..."
    docStatus: "Accepted"
  __v: 0
```

Workshop Presenter:



```
  _id: ObjectId("60dc3f2342a74800159b4ca3")
> payments: Array
  userid: "157c0c95-ac89-465e-b02a-4a201b6ac927"
  name: "Sachini Jayasinghe"
  email: "sachinijayasinghe@gmail.com"
  password: "$2a$10$Srsq9kkCuCiMyDIMDWhjsupptQ7NNjHgS/lByg6.qWc7vsfYOQVwq"
  role: "Workshop Presenter"
v document: Object
    docName: "base-document-1625046805511"
    docData: "data:application/pdf;base64,JVBERi0xLjcNCiW1tbW1DQoxIDAgb2JqDQo8PC9UeX..."
    docStatus: "Rejected"
  __v: 0
```

Test Cases (member wise)

Jayasekara R.T.R – IT19129204

API Testing:

The testing is handled using Jest framework and MongoDB-memory-saver package. Jest provides methods to create, organize and run the tests. The mongodb-memory server creates a cluster which only exists in the device's main memory and is not physically stored to disk. Therefore, once the application terminates, the database instance will no longer exist.

Db-handler.js:

```
const mongoose = require("mongoose");
const { MongoMemoryServer } = require("mongodb-memory-server");

const mongod = new MongoMemoryServer();

module.exports.connect = async () => {
  const uri = await mongod.getConnectionString();

  const mongooseOpts = {
    useNewUrlParser: true,
    autoReconnect: true,
    reconnectTries: Number.MAX_VALUE,
    reconnectInterval: 1000,
  };

  await mongoose.connect(uri, mongooseOpts);
};

module.exports.closeDatabase = async () => {
  await mongoose.connection.dropDatabase();
  await mongoose.connection.close();
  await mongod.stop();
};

module.exports.clearDatabase = async () => {
  const collections = mongoose.connection.collections;

  for (const key in collections) {
    const collection = collections[key];
    await collection.deleteMany();
  }
};
```

This is a module to execute some basic operations of the in memory database.

Test case for saveUser():

```
describe("user ", () => {
  it("can be created correctly", async () => {
    expect(async () => await saveUser(req, res, next)).not.toThrow();
  });
});

const req = {
  body: {
    name: "Rukshan Jayasekara",
    email: "it19129204@my.sliit.lk",
    password: "qwerty123456",
    role: "Attendee",
    document: {},
    paymentForm: {
      cardNo: "1234567890",
      expDate: "03/04/2024",
      cvv: "123",
    },
  },
};
```

Test case for savePayment():

```
describe("payment ", () => {
  it("can be created correctly", async () => {
    expect(async () => await savePayment(req, res, next)).not.toThrow();
  });
});

const req = {
  body: {
    paymentForm: {
      cardNo: "1234567890",
      expDate: "03/04/2024",
      cvv: "123",
    },
    userid: mongoose.Types.ObjectId("55153a8014829a865bbf700d"),
  },
};
```

React App Testing:

React component testing and snapshot testing was done using Jest and Enzyme.

Component Testing: (App.test.js and User.test.js)

```
it("renders without crashing", () => {
    shallow(<App />);
  });
```

```
it("DocumentUpload renders without crashing", () => {
    shallow(<DocumentUpload />);
});

it("BankForm renders without crashing", () => {
    shallow(<BankForm />);
});

it("Signin renders without crashing", () => {
    shallow(<Signin />);
});
```

Snapshot Testing: (User.snapshot.test.js)

```
it("DocumentUpload renders correctly", () => {
    const tree = shallow(<DocumentUpload />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("BankForm renders without crashing", () => {
    const tree = shallow(<BankForm />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("Signin renders without crashing", () => {
    const tree = shallow(<Signin />);
    expect(toJson(tree)).toMatchSnapshot();
});
```

Kariyawasam K.G.S.S.K – IT19147024

React App Testing:

React component testing was done using Just and Enzyme

Component testing: (Admin-test.js)

```
it("Dashboard renders without crashing", () => {
    shallow(<Dashboard />);
});

it("Conference renders without crashing", () => {
    shallow(<Conference />);
});

it("Approved renders without crashing", () => {
    shallow(<Approved />);
});
```

Kudarachchi K.A.N.D – IT19121352

API Testing

Test Case for createConference()

```
describe("conference ", () => {
  it("can be created correctly", async () => {
    expect(async () => await addingConference(req, res, next)).not.toThrow();
  });
});

const req = {
  body: {
    name: "Environmental Studies",
    description: "About environment",
    venue: "sliit",
    starttime: "09.12.2021  09.00AM",
    endtime: "12.12.2021  18.00PM",
    guest: "Nimal Perera",
    guest2: "Sahan Perera",
    guest3: "Samantha Fernando",
    status: "false",
  },
};
```

Test case for createPresentation()

```
describe("presentation ", () => {
  it("can be created correctly", async () => {
    expect(async () => await createPresentation(req, res, next)).not.toThrow();
  });
});

const req = {
  body: {
    conference: "Environmental Studies",
    topic: "Introduce Environment",
    description: "nothing",
    starttime: "09.12.2021  09.00AM",
    endtime: "12.12.2021  18.00PM",
    presenter: "Saman perera"
  },
};
```

Test case for createWorkshop()

```
describe("workshop ", () => {
  it("can be created correctly", async () => {
    expect(async () => await createWorkshop(req, res, next)).not.toThrow();
  });
});

const req = {
  body: {
    conference: "Environmental Studies",
    topic: "environmental workshop",
    description: "nothing",
    starttime: "09.12.2021  09.00AM",
    endtime: "12.12.2021  18.00PM",
    presenter: "Sumana perera"
  },
};
```

React App Testing:

React component testing and snapshot testing was done using Jest and Enzyme.

Component Testing: (editor.test.js)

```
it("Panel renders without crashing", () => {
    shallow(<Panel />);
});

it("Editor renders without crashing", () => {
    shallow(<Editor />);
});

it("Presentation renders without crashing", () => {
    shallow(<Presentation />);
});

it("Presentation renders without crashing", () => {
    shallow(<ViewConference />);
});

it("Presentation renders without crashing", () => {
    shallow(<UpdateConference />);
});
```

Snapshot Testing: (editor.snapshot.test.js)

```
it("Panel renders correctly", () => {
    const tree = shallow(<Panel />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("Editor renders without crashing", () => {
    const tree = shallow(<Editor />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("Presentation renders without crashing", () => {
    const tree = shallow(<Presentation />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("ViewConference renders without crashing", () => {
    const tree = shallow(<ViewConference />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("updateConference renders without crashing", () => {
    const tree = shallow(<UpdateConference />);
    expect(toJson(tree)).toMatchSnapshot();
});
```

Jayasinghe S.L– IT19126234

React App Testing:

Component Testing:

Reviewer.test.js

```
it("DocumentView renders without crashing", () => {
    shallow(<DocumentView />);
});

it("Presenters renders without crashing", () => {
    shallow(<Presenters />);
});

it("Researchers renders without crashing", () => {
    shallow(<Researchers />);
});
```

Reviewer.snapshots.test.js

```
it("DocumentView renders correctly", () => {
    const tree = shallow(<DocumentView />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("Presenters renders without crashing", () => {
    const tree = shallow(<Presenters />);
    expect(toJson(tree)).toMatchSnapshot();
});

it("Researchers renders without crashing", () => {
    const tree = shallow(<Researchers />);
    expect(toJson(tree)).toMatchSnapshot();
});
```