# Master Team Project Fall 2023

## Fulda Hochschule Digital Bibliotheca

*Team 3 - Dev Dragons*
Team Lead: Parsa Rashidikia - parsa.rashidikia@informatik.hs-fulda.de
Back-end Lead: Rahul Patil
Front-end Lead: MD Monoarul Islam
Github Master: Hauva Vali
Fullstack Developer: Amar Sharma

## Milestone 4

*23/11/2023*

**Revision 1**

## Table of Contents

## 1. Product Summary - FHDB (Fulda Hochschule Digital Bibliotheca)

In today's fast-paced digital world, the use of physical media is becoming less and less popular; universities are seeking innovative solutions to enhance communication, collaboration, and resource sharing within their academic communities. To address these needs, we are proud to introduce our FHDB (Fulda Hochschule Digital Bibliotheca), a cutting-edge platform that facilitates seamless buying, selling, and sharing of digital media, including videos, images, audio files, and documents, exclusively designed for faculty and students.

**Key Features:**

- Upload your own digital media to share your work with fellow students
- Search for media that matches your taste and budget
- Keep track of your purchased media and download them at any given time
- Manage your shared media
- Keep in touch directly using the chat system with fellow creators

**What makes us unique?**

- Share your opinion on a product by participating in dedicated group chats
- Feel safe to share your precious creations as they will be watermarked

**Application URL:** https://teamthreegdsdfulda.westeurope.cloudapp.azure.com/

## 2. Usability Test Plan

- **Test Objective:**

As a platform focused on buying, selling, and sharing media, the importance of media creation in the application is of utmost significance, hence the upload functionality being the core of the platform. This vital organ of the application must be easy to use, covering all the users' needs, as well as optimized in performance metrics, therefore, it is the perfect candidate to be the objective of our Usability Test.

- **Test Background and Setup:**

To ensure that the results are based only on the experience with the upload functionality, additional setup had been conducted before the start of the actual test. The application was hosted on Microsoft Azure Cloud platform, as well as a test user created only for this purpose.

Since a usability test is a way of understanding customer behavior and patterns and improving the product based on those factors, we aimed to have non-technical individuals as our testers. As a result, we kindly asked some of our fellow students to perform the tests. A laptop was provided to them, the test user was already logged in to the platform and the testers began their procedure from the dashboard page.

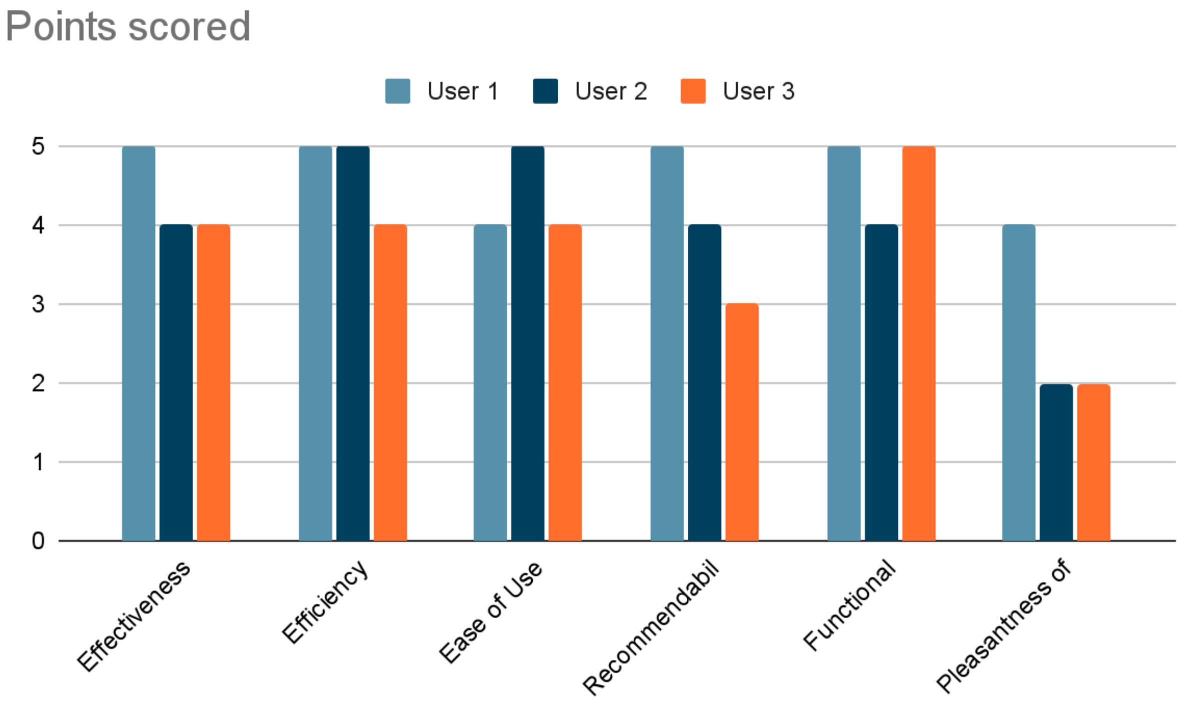
URL of the System Under Test: https://teamthreegdsdfulda.westeurope.cloudapp.azure.com/

In this test we asked our testers to measure the following factors on a scale of 1 (being the worst) to 5 (being the best):

A. Effectiveness
B. Efficiency
C. Ease of Use
D. Recommendability
E. Functional Satisfaction
F. Pleasantness of the Interface

After briefly describing the platform to the testers, we provided a detailed description of our content upload functionality, what data should be provided, and how to go through the upload process. And asked them to share their experience with us apart from scoring. In addition to that, during the process of testing, a few bugs have been discovered.

The results are shown in the table below:



## 3. QA Test Plan

**Test Objective:** Uploading media

**HW/SW Setup:** Application Deployed on the cloud, Test user logged in.

*URL: https://teamthreegdsdfulda.westeurope.cloudapp.azure.com/upload-content*

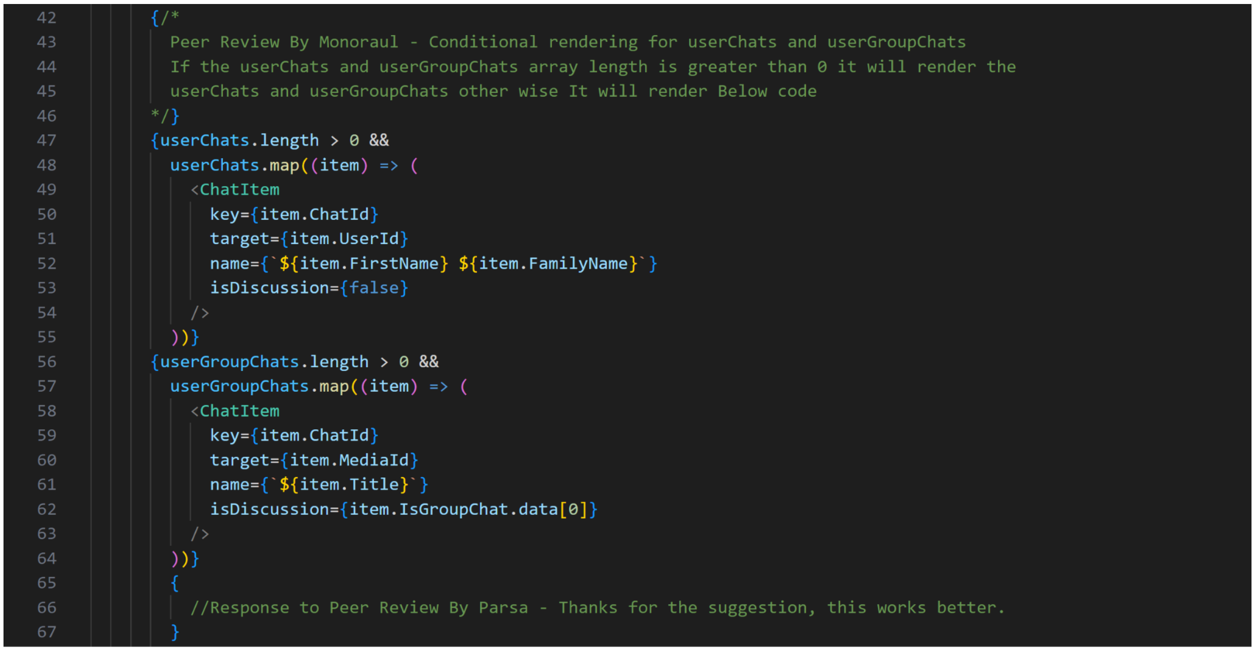
**Feature To Be Tested:** Media Upload

**QA Test Plan:**

| Test Cases | | | | |
|---|---|---|---|---|
| ID | Description | Test Steps | Expected Result | Status |
| TC01 | Upload media with actual media, should create successfully | Filling in Title, Description, Price, MediaType, and providing file for demo and the original file(s) | Media Created successfully and awaiting administrator's approval | PASS |
| TC02 | Upload Media without a demo file, should produce error | Filling in Title, Description, Price, MediaType, and providing the original file(s) without the demo file | Error produced, No file uploaded | PASS |
| TC03 | Upload Media, providing wrong file with respect to the chosen MediaType | Filling in Title, Description, Price, MediaType as Image, and providing the demo file and original file(s) as Document(s) | Error Produced, No file uploaded | PASS |
| TC04 | Upload Media, Providing everything except the Title | Filling in, Description, Price, MediaType, and providing file for demo and the original file(s) | No request should be sent to the backend, validation error should be shown to the user in UI | PASS |

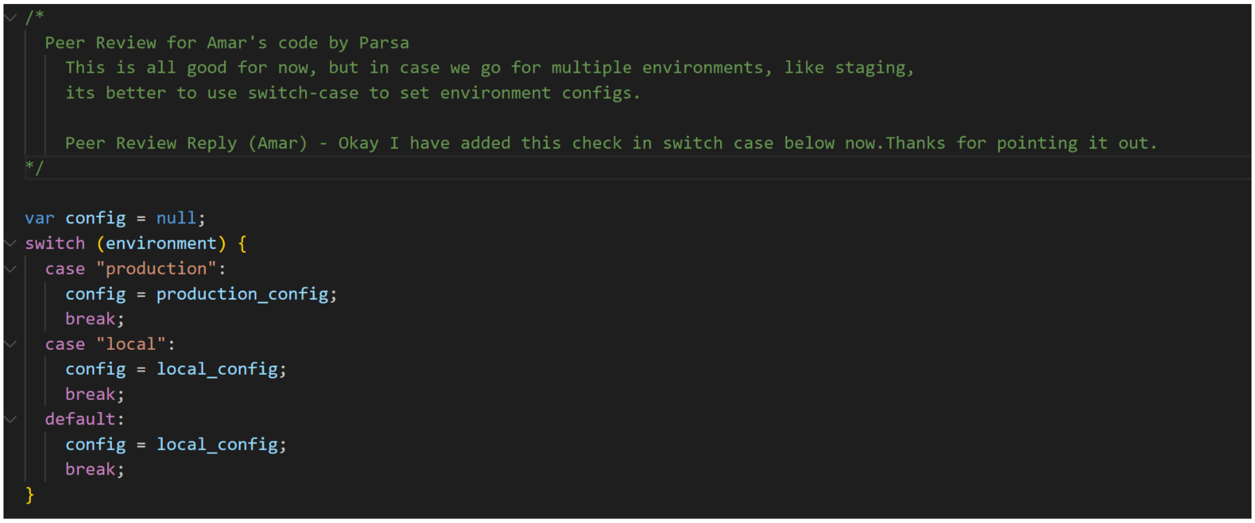
## 4. Code Review

Peer Review on chatList.jsx:



```jsx
{/*
  Peer Review By Monoraul - Conditional rendering for userChats and userGroupChats
  If the userChats and userGroupChats array length is greater than 0 it will render the
  userChats and userGroupChats other wise It will render Below code
*/}
{userChats.length > 0 &&
  userChats.map((item) => (
    <ChatItem
      key={item.ChatId}
      target={item.UserId}
      name={`${item.FirstName} ${item.FamilyName}`}
      isDiscussion={false}
    />
  ))}
{userGroupChats.length > 0 &&
  userGroupChats.map((item) => (
    <ChatItem
      key={item.ChatId}
      target={item.MediaId}
      name={`${item.Title}`}
      isDiscussion={item.IsGroupChat.data[0]}
    />
  ))}
  {
    //Response to Peer Review By Parsa - Thanks for the suggestion, this works better.
  }
```

Peer Review on connection.js:

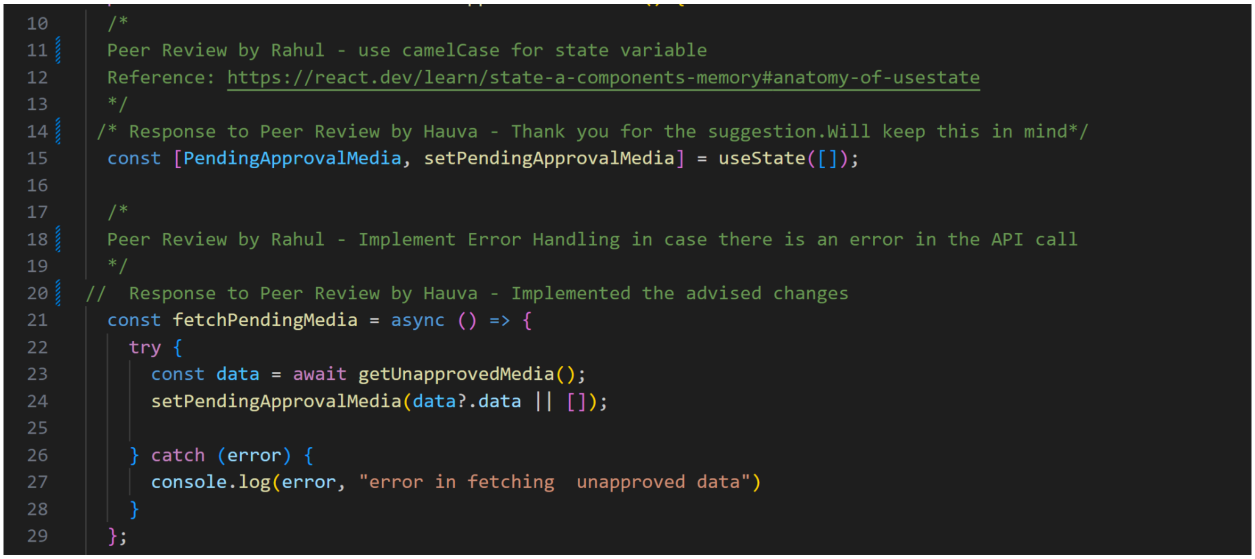


```js
/*
  Peer Review for Amar's code by Parsa
    This is all good for now, but in case we go for multiple environments, like staging,
    its better to use switch-case to set environment configs.

    Peer Review Reply (Amar) - Okay I have added this check in switch case below now.Thanks for pointing it out.
*/

var config = null;
switch (environment) {
  case "production":
    config = production_config;
    break;
  case "local":
    config = local_config;
    break;
  default:
    config = local_config;
    break;
}
```

Peer Review on admin-approval-dashboard.jsx

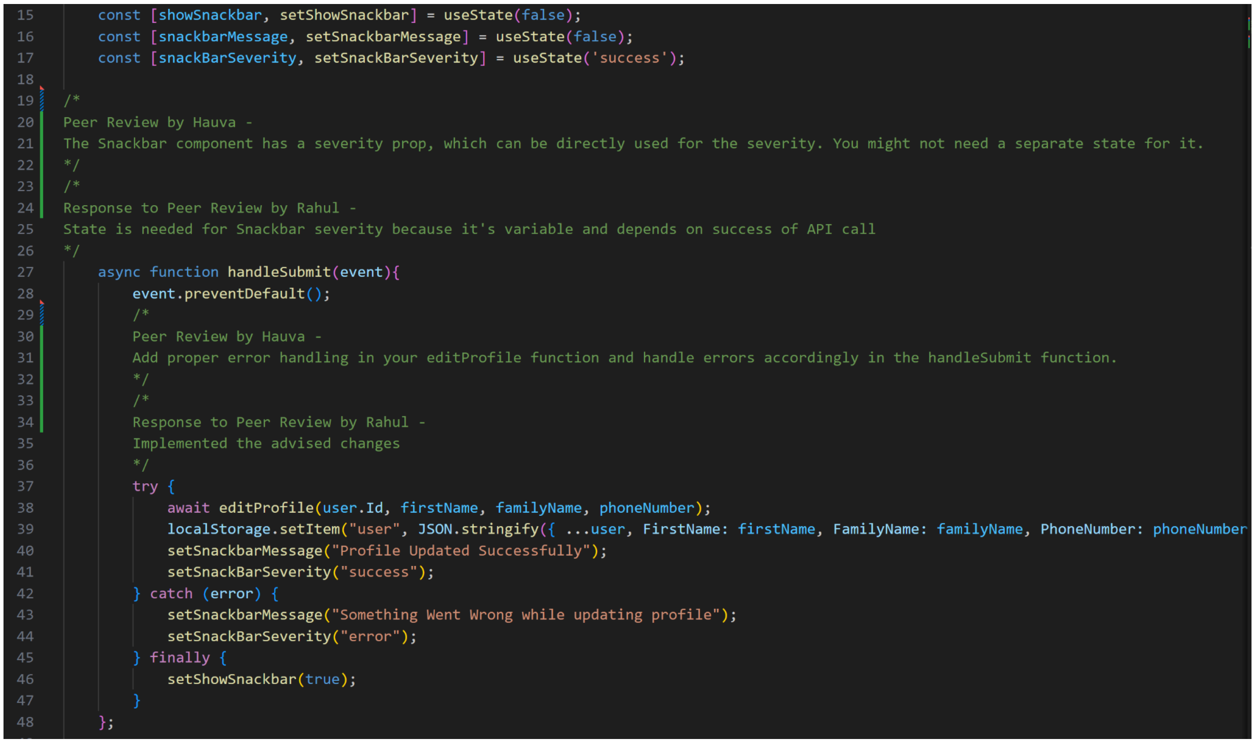


```
  /*
  Peer Review by Rahul - use camelCase for state variable
  Reference: https://react.dev/learn/state-a-components-memory#anatomy-of-usestate
  */
 /* Response to Peer Review by Hauva - Thank you for the suggestion.Will keep this in mind*/
  const [PendingApprovalMedia, setPendingApprovalMedia] = useState([]);

  /*
  Peer Review by Rahul - Implement Error Handling in case there is an error in the API call
  */
 //  Response to Peer Review by Hauva - Implemented the advised changes
  const fetchPendingMedia = async () => {
    try {
      const data = await getUnapprovedMedia();
      setPendingApprovalMedia(data?.data || []);

    } catch (error) {
      console.log(error, "error in fetching  unapproved data")
    }
  };
```

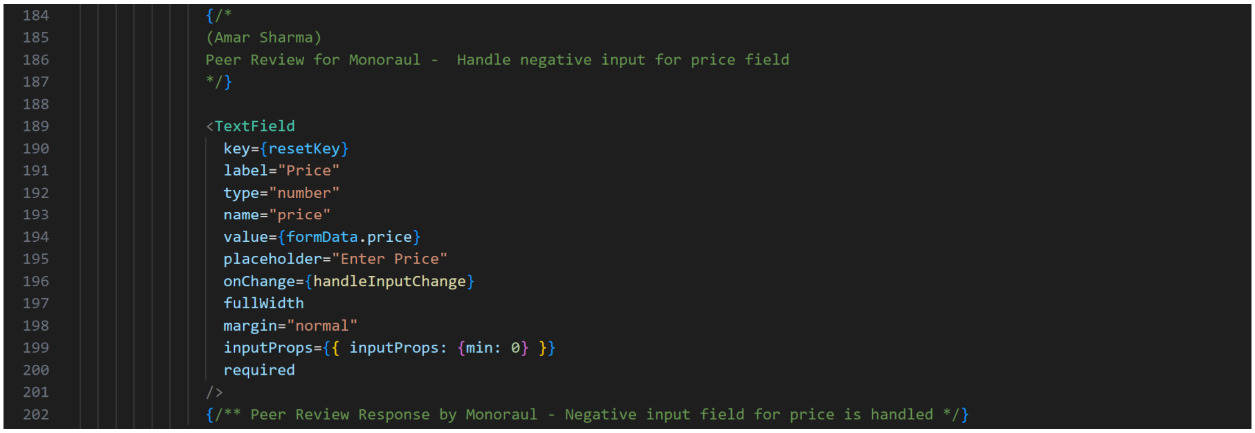
Peer Review on edit-profile.jsx:



```
    const [showSnackbar, setShowSnackbar] = useState(false);
    const [snackbarMessage, setSnackbarMessage] = useState(false);
    const [snackBarSeverity, setSnackBarSeverity] = useState('success');

/*
Peer Review by Hauva -
The Snackbar component has a severity prop, which can be directly used for the severity. You might not need a separate state for it.
*/
/*
Response to Peer Review by Rahul -
State is needed for Snackbar severity because it's variable and depends on success of API call
*/
    async function handleSubmit(event){
        event.preventDefault();
        /*
        Peer Review by Hauva -
        Add proper error handling in your editProfile function and handle errors accordingly in the handleSubmit function.
        */
        /*
        Response to Peer Review by Rahul -
        Implemented the advised changes
        */
        try {
            await editProfile(user.Id, firstName, familyName, phoneNumber);
            localStorage.setItem("user", JSON.stringify({ ...user, FirstName: firstName, FamilyName: familyName, PhoneNumber: phoneNumber
            setSnackbarMessage("Profile Updated Successfully");
            setSnackBarSeverity("success");
        } catch (error) {
            setSnackbarMessage("Something Went Wrong while updating profile");
            setSnackBarSeverity("error");
        } finally {
            setShowSnackbar(true);
        }
    };
```

Peer Review on upload-content.jsx:



```
{/*
(Amar Sharma)
Peer Review for Monoraul -  Handle negative input for price field
*/}

<TextField
  key={resetKey}
  label="Price"
  type="number"
  name="price"
  value={formData.price}
  placeholder="Enter Price"
  onChange={handleInputChange}
  fullWidth
  margin="normal"
  inputProps={{ inputProps: {min: 0} }}
  required
/>
{/** Peer Review Response by Monoraul - Negative input field for price is handled */}
```

## 5. Security Best Practices Self-Check

**Protected assets:**

- Database: The database is protected with a secure password to be resistant to brute force direct attacks to the database. The queries are also parameterized to help protect the database against SQL Injection attacks.
- Host: The server hosting the application is protected against ssh reverse shell and man-in-the-middle attacks by removing password authentication and using a 4096-bit SSH RSA key for authentication.
- Content: The media uploaded by the user are stored safely in Firebase storage, as well as images being watermarked for copyright reasons. In addition to that, each uploaded media must be approved by the administrators before being made public to ensure that no inappropriate media threatens the safety of the platform.
- Password: Passwords are encrypted and stored in the database. In case of database leaks, it is almost impossible to reverse the hash and see the actual passwords.
- Input Validation: Most of the input data entered by the user are validated on the frontend as well as the backend to help protect against man-in-the-middle and denial-of-service attacks.

## 6. Adherence to original non-functional specs

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in Milestone 0. Application delivery shall be from chosen cloud server DONE

2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers DONE

3. All or selected application functions must render well on mobile devices DONE

4. Data shall be stored in the database on the team's deployment cloud server. DONE

5. Full resolution free media shall be downloadable directly, and full resolution media for selling shall be obtained after contacting the seller/owner. DONE

6. No more than 50 concurrent users shall be accessing the application at any time. DONE

7. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users. DONE

8. The language used shall be English (no localization needed). DONE

9. Application shall be very easy to use and intuitive. DONE

10. Application should follow established architecture patterns. DONE

11. Application code and its repository shall be easy to inspect and maintain. DONE

12. No email clients shall be allowed. DONE

13. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. DONE

14. Site security: basic best practices shall be applied (as covered in the class) for main data items. DONE

15. Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today DONE