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 2

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:

1. **User-Friendly Interface**: An intuitive and easy-to-navigate interface ensures that both faculty and students can effortlessly buy, sell, and share digital media, promoting a seamless user experience.
2. **Digital Media Exchange:** Users can upload, buy, and sell a wide variety of digital content, including videos, images, audio files, and documents. This feature streamlines the acquisition of educational materials and promotes knowledge sharing.
3. **Communication Tools**: Our application facilitates real-time communication between users, allowing for collaboration, inquiries, and discussions, all within a secure and controlled environment.
4. **Reviews and Ratings:** Users can post reviews and ratings for purchased media, creating a transparent and reliable system for evaluating the quality of content, creating a sense of community and trust.

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 was conducted before the actual test. The application was hosted on the Microsoft Azure Cloud platform, as well as a test user was 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 users who tested the application are categorized in 3 age groups, detail of which can be found below:

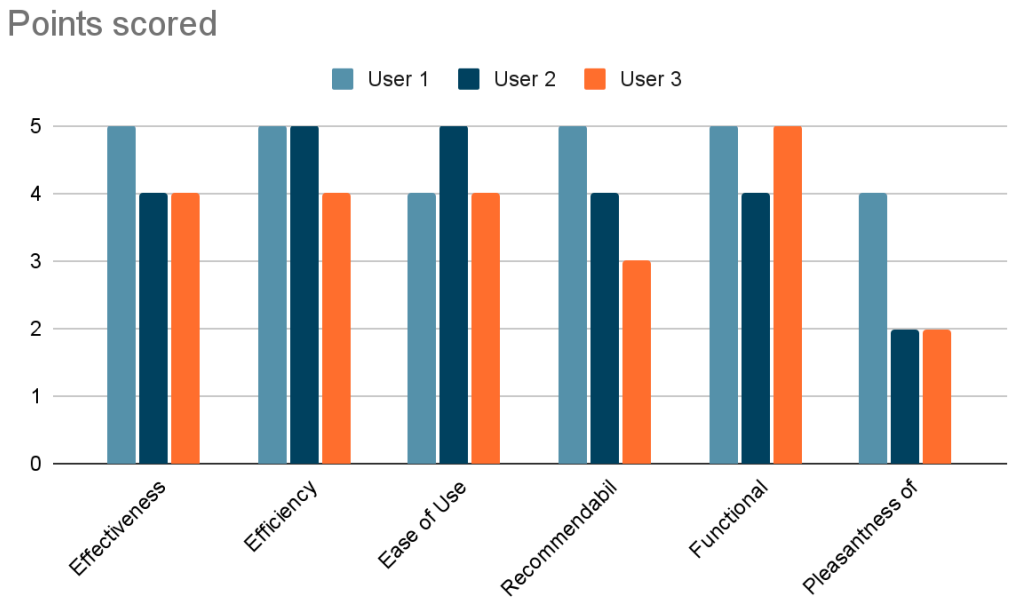
Age Group 1 (20 - 30):

User 1: Student in health sciences, 20 years old, no technical background.

User 2: Student in social sciences, 22 years old, mediocre technical background.

User 3: Student in social sciences, 26 years old, no technical background.

The results are shown in the table below:



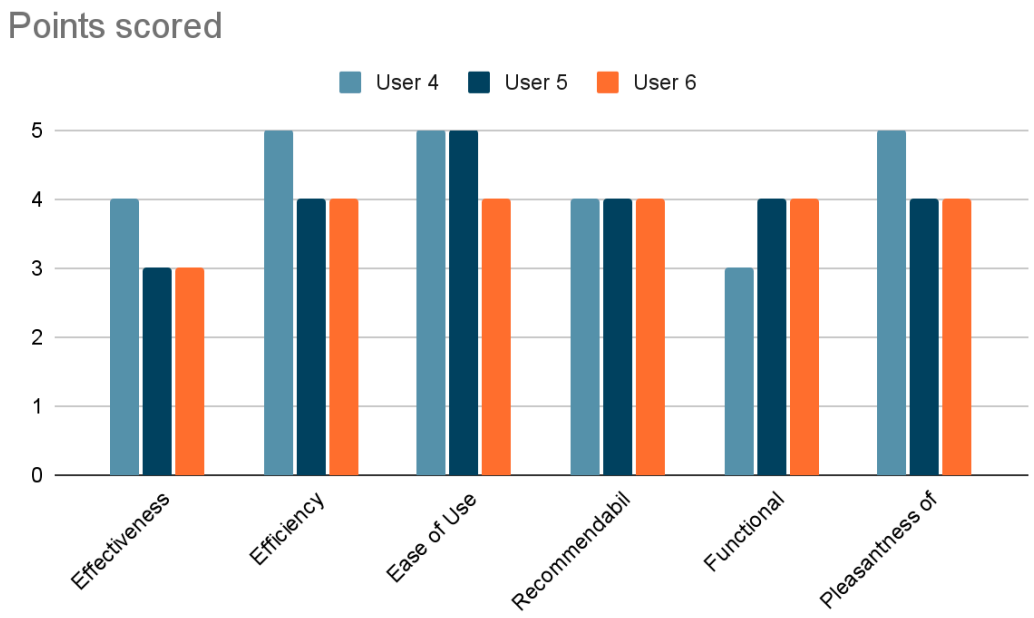
Age Group 2 (31 - 50):

User 4: Working Professional in Banking, 35 years old, no technical background.

User 5: Homemaker, 48 years old, no technical background.

User 6: Master's Student, 31 years old, technical background.

The results are shown in the table below:

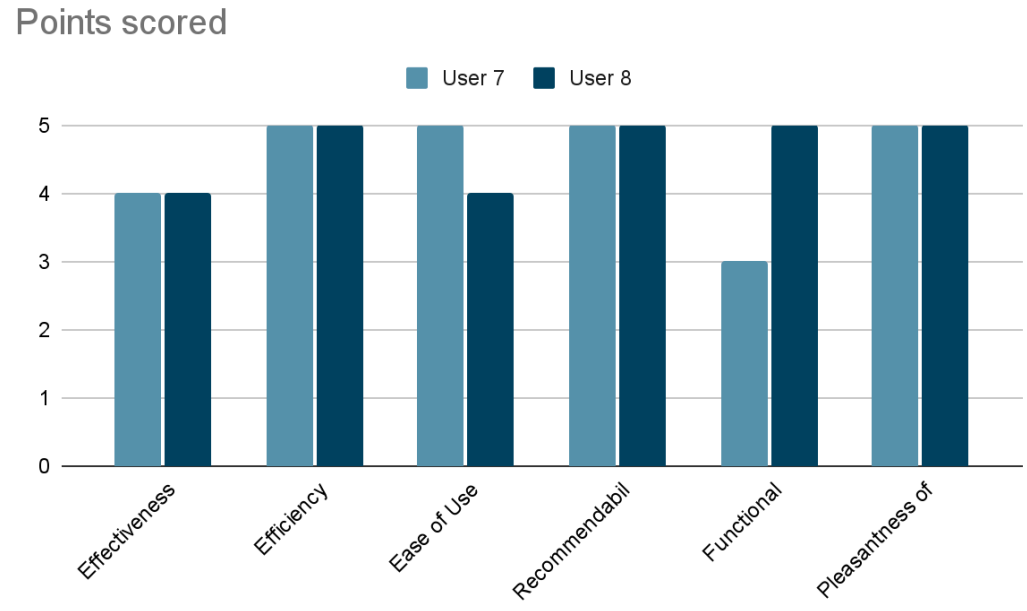


Age Group 3 (51 - 80):

User 7: Businessman, 55 years old, no technical background.

User 8: Retired Individual, 75 years old, no technical background.

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.

Browsers: Microsoft Edge 12 / Google Chrome 121

Operating Systems: Windows 11 / Ubuntu 22

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 concerning the chosen MediaType	Filling in Title, Description, Price, and 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 a file for the demo and	No request should be sent to the backend, validation error should be shown	PASS

		the original file(s)	to the user in UI	
TC05	Upload media with special characters in the Title field	Filling in Title with special characters, Description, Price, MediaType, and providing file for demo and the original file(s)	Media Created successfully and awaiting administrator's approval	PASS
TC06	Attempt to upload media with a negative price value	Filling in Title, Description, Price as -50, 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
TC07	Try to upload media without providing any information (blank form)	Leaving all fields blank and attempting to upload media	No request should be sent to the backend, validation errors should be shown for each required field in the UI	PASS
TC08	Upload media with a decimal price value	Filling in Title, Description, Price with a decimal value (e.g., 19.99), MediaType, and providing file for demo and the original file(s)	Media Created successfully and awaiting administrator's approval	PASS

4. Code Review

1. Peer review of Parsa by Monoraul

Source File : chatList.jsx

Link: [GdsdTeam3/react-app/src/sections/chat/chatList/chatList.jsx at development · hauvavali/GdsdTeam3 (github.com)]

Before:

```
/*
  Peer Review By Monoraul - Implemented try and catch block
  Here I have added try and catch block so that user does not stuck on the error page
  If there is any data it will load the data.
  Else it will show a error message : Error loading user chats
*/
{userChats.map((item) => (
  <ChatItem
    key={item.ChatId}
    target={item.UserId}
    name={`${item.FirstName} ${item.FamilyName}`}
    isDiscussion = {false}
  />
))}
{userGroupChats.map((item) => (
  <ChatItem
  key={item.ChatId}
  target={item.MediaId}
  name={`${item.Title}`}
  isDiscussion = {item.IsGroupChat.data[0]}
  />
))}
```

After :

```
{/*
  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.
  }
```

2. Peer review of Amar by Parsa

Source File : connection.js:

Link : [GdsdTeam3/backend/utils/connection.js at development · hauvavali/GdsdTeam3 (github.com)]

Before :

```
/*
  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.
*/
var config = environment == "production" ? production_config : local_config;

var connection = mysql.createConnection(config);
```

After :

```
/*
  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;
}
```

3. Peer review of Hauva by Rahul

Source File: admin-approval-dashboard.jsx:

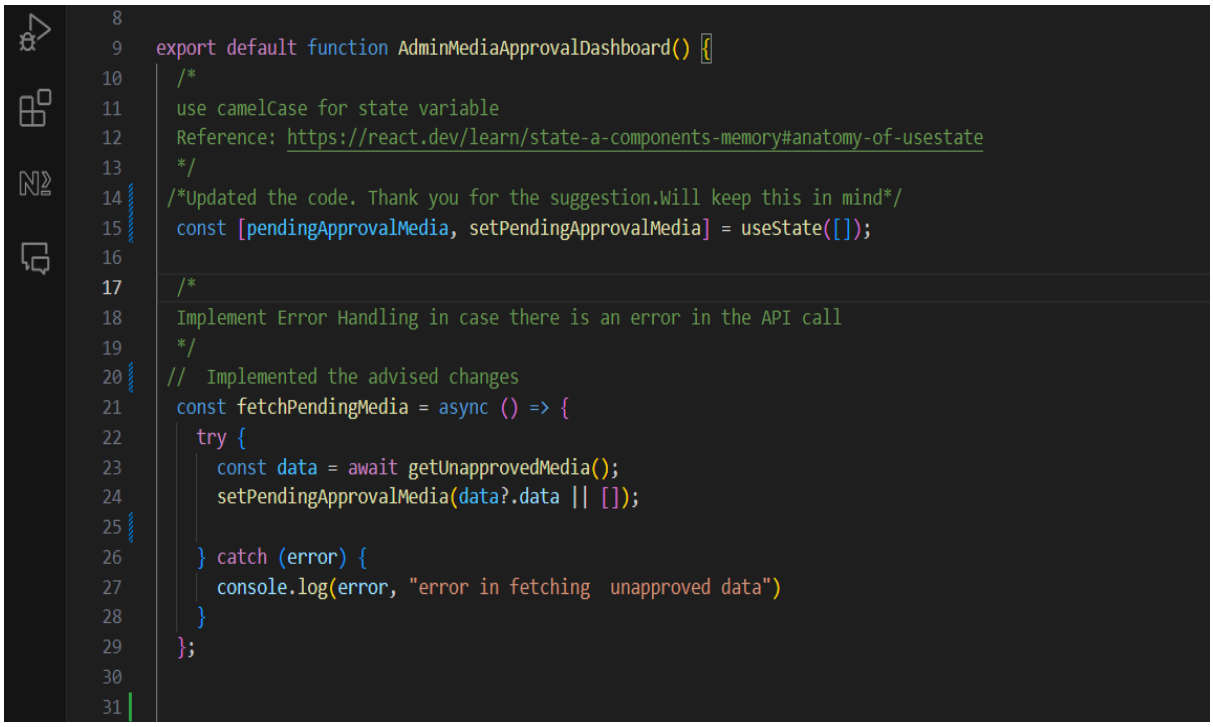
Link: [GdsdTeam3/react-app/src/pages/admin-approval-dashboard.jsx at development · hauvavali/GdsdTeam3 (github.com)]

Before:

```
export default function AdminMediaApprovalDashboard() {
  /*
  Peer Review by Rahul - use camelCase for state variable
  Reference: https://react.dev/learn/state-a-components-memory#anatomy-of-usestate
  */
  const [PendingApprovalMedia, setPendingApprovalMedia] = useState([]);

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

After:



```
export default function AdminMediaApprovalDashboard() {
  /*
  use camelCase for state variable
  Reference: https://react.dev/learn/state-a-components-memory#anatomy-of-usestate
  */
 /*Updated the code. Thank you for the suggestion.Will keep this in mind*/
  const [pendingApprovalMedia, setPendingApprovalMedia] = useState([]);

  /*
  Implement Error Handling in case there is an error in the API call
  */
 //  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")
    }
  };
```

4. Peer review of Rahul by Hauva

Source : Peer Review on edit-profile.jsx:

Link: [GdsdTeam3/react-app/src/pages/edit-profile.jsx at main · hauvavali/GdsdTeam3 (github.com)]

Before:

```
/*
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.
*/
    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.
        */
        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");

    };
```

After:

```
15      const [showSnackbar, setShowSnackbar] = useState(false);
16      const [snackbarMessage, setSnackbarMessage] = useState(false);
17      const [snackBarSeverity, setSnackBarSeverity] = useState('success');
18
19  /*
20  Peer Review by Hauva -
21  The Snackbar component has a severity prop, which can be directly used for the severity. You might not need a separate state for it.
22  */
23  /*
24  Response to Peer Review by Rahul -
25  State is needed for Snackbar severity because it's variable and depends on success of API call
26  */
27      async function handleSubmit(event){
28          event.preventDefault();
29          /*
30          Peer Review by Hauva -
31          Add proper error handling in your editProfile function and handle errors accordingly in the handleSubmit function.
32          */
33          /*
34          Response to Peer Review by Rahul -
35          Implemented the advised changes
36          */
37          try {
38              await editProfile(user.Id, firstName, familyName, phoneNumber);
39              localStorage.setItem("user", JSON.stringify({ ...user, FirstName: firstName, FamilyName: familyName, PhoneNumber: phoneNumber
40              setSnackbarMessage("Profile Updated Successfully");
41              setSnackBarSeverity("success");
42          } catch (error) {
43              setSnackbarMessage("Something Went Wrong while updating profile");
44              setSnackBarSeverity("error");
45          } finally {
46              setShowSnackbar(true);
47          }
48      };
```

5. Peer Review of Monoraul by Amar

Source File: Peer Review on upload-content.jsx:

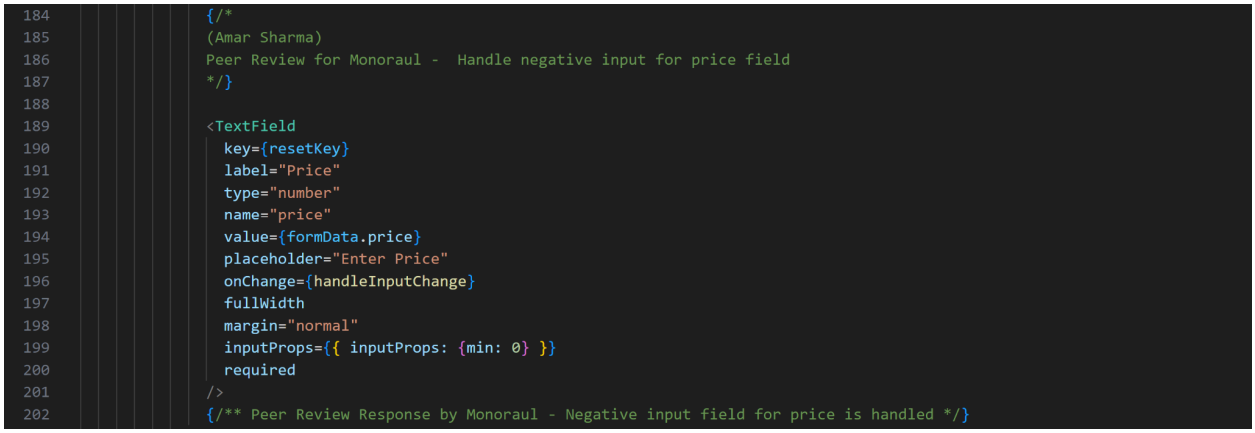
Link: [GdsdTeam3/react-app/src/pages/upload-content.jsx at development · hauvavali/GdsdTeam3 (github.com)]

Before:

```jsx
<div>
  {/*
  (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"
    required
  />
</div>
```

After:



```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 front end as well as the back end to help protect against man-in-the-middle and denial-of-service attacks.

6. Adherence to original non-functional specs

1. The 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 the chosen cloud server DONE

2. The 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. The 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. The application shall be very easy to use and intuitive. DONE

10. The 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. The application shall be media-rich (images, video etc.). Media formats shall be standard as used in the market today DONE