

A PROJECT REPORT

on

B-Farm

Submitted by

Mr. SINGH SHANKAR SURENDRA

in partial fulfilment for the award of the degree

of

BACHELOR OF SCIENCE

in

COMPUTER SCIENCE

under the guidance of

Prof. FLOSIA SIMON

Department of Computer Science



**SIES (Nerul) College of Arts, Science and Commerce
(Autonomous)**

NAAC Re-Accredited (3rd Cycle) Grade 'A' CGPA 3.01/4.00

Sri Chandrasekarendra Saraswathy Vidyapuram, Plot 1-C, Sector V,
Nerul, Navi Mumbai – 400 706

(Sem VI)

(2023 – 2024)



**SIES (Nerul) College of Arts, Science and Commerce
(Autonomous)**

NAAC Re-Accredited (3rd Cycle) Grade 'A' CGPA 3.01/4.00

**Sri Chandrasekarendra Saraswathy Vidyapuram, Plot 1-C, Sector V,
Nerul, Navi Mumbai – 400 706**

Department of Computer Science

CERTIFICATE

This is to certify that Mr. **Singh Shankar Surendra** of T.Y.B.Sc (Sem-VI) class has satisfactorily completed the Project **B-Farm (render farm)**, to be submitted in the partial fulfilment for the award of **Bachelor of Science in Computer Science** during the academic year **2023 – 2024**.

Date of Submission:

**Project Guide
(Prof. Flosia Simon)**

**Head of Department
(Prof. Dr. Sheeja K.)**

College Seal

Signature of Examiner

DECLARATION

I, **Singh Shankar Surendra**, hereby declare that the project entitled “**B_Farm (render farm)**” submitted in the partial fulfilment for the award of Bachelor of Science in **Computer Science** during the academic year **2023 – 2024** is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

Signature of the Student:

Place:

Date:

ACKNOWLEDGMENT

I extend my heartfelt appreciation to all those who contributed to the realization of the Bfarm project.

I would like to thank all the Blender educators and content creators whose tutorials and insights have served as a source of inspiration and knowledge throughout my journey. Your dedication to sharing your expertise has been invaluable in my learning process.

I am grateful to my peers, mentors, and friends for their unwavering support and encouragement. Your feedback and guidance have been instrumental in shaping the Bfarm project and pushing me to strive for excellence.

Lastly, I acknowledge the collaborative efforts of my team members, whose dedication and hard work have been instrumental in the success of this project. Each member's unique contributions have contributed to the development and refinement of the Bfarm platform.

Thank you to everyone who has been a part of this remarkable journey.

TABLE OF CONTENT

Sr. No.	CONTENT	Page No.
1	TITLE	6
2	ABSTRACT	7
3	Introduction <ul style="list-style-type: none">• Purpose• Key feature overview	8
4	TOOLS AND TECHNOLOGIES	9
5	TARGET AUDIENCE	10
6	GANTT CHART	11
7	REQUIRMENT SPECIFICATION <ul style="list-style-type: none">a. System requirementb. Functional requirementc. Non-functional requirement	12 - 13
8	FEASIBILITY STUDY	14
9	SYSTEM DESIGN <ul style="list-style-type: none">a. High-Level Architecture:b. Component Diagram:c. Sequence Diagram - User Authentication:d. Deployment Diagram	15 - 17
10	IMPLEMENTATION <ul style="list-style-type: none">• Development setup• Folder structure• Code snippets	18 - 30
11	DEPLOYMENT	31
12	RESULTS	32 - 38
13	CONCLUSION AND FUTURE SCOPE	39 - 40
14	REFERENCES	41
15	PLAGIARISM REPORT	42

TITLE

It provides me a great opportunity to present this project on the topic

“Cloud Rendering Farm”

The name of my project is

“B - Farm”



ABSTRACT

The Bfarm project introduces a revolutionary solution to the time-consuming process of rendering Blender files by leveraging distributed cloud computing. This documentation provides a comprehensive overview of the Bfarm platform, detailing its architecture, functionality, and implementation process.

The primary objective of Bfarm is to address the challenges faced by Blender users in rendering large projects efficiently. By harnessing the combined power of multiple users' systems connected through a centralized middleware, Bfarm significantly reduces rendering time and enhances productivity for Blender artists and learners.

This documentation outlines the project's scope, methodology, and tools and technologies used in its development. It provides insights into the development process, including requirement analysis, system design, implementation, testing, and deployment phases.

Through the Bfarm platform, Blender users gain access to high-performance rendering capabilities, irrespective of their system specifications, fostering collaboration and resource-sharing within the Blender community. The documentation aims to serve as a guide for understanding and utilizing the Bfarm platform effectively to streamline the rendering process and enhance productivity in Blender projects.

INTRODUCTION

Bfarm emerges as a groundbreaking solution for Blender users seeking to optimize the rendering process. With meticulous attention to detail, Bfarm offers a sophisticated platform designed to streamline rendering tasks, empowering users with enhanced productivity and collaboration capabilities.

Purpose

The primary goal of Bfarm is to revolutionize rendering practices within the Blender community. By harnessing the collective computational power of distributed systems, Bfarm aims to significantly reduce rendering time while ensuring accessibility, efficiency, and reliability. With an emphasis on user experience and seamless integration, Bfarm strives to meet the evolving needs of Blender artists and learners.

Key Features Overview

- **Distributed Rendering:** Leveraging the combined power of multiple users' systems for faster rendering.
- **Seamless Integration:** Effortless integration with Blender software for a smooth rendering experience.
- **Collaborative Platform:** Fostering collaboration and resource-sharing within the Blender community through a centralized platform.
- **Enhanced Productivity:** Providing Blender users with access to high-performance rendering capabilities, thereby enhancing productivity and creativity.

TOOLS AND TECNOLOGIES

Bfarm is a cloud rendering farm application developed using a combination of technologies and tools to provide efficient rendering services. The following is a list of key tools and technologies utilized in the development of Bfarm:

1. Electron:

- Purpose: Framework for building cross-platform desktop applications using web technologies.
- Version: 29.1.6

2. Firebase:

- Purpose: Platform for building and managing web and mobile applications.
- Version: 8.10.0

3. Archiver:

- Purpose: Library for creating and extracting compressed files and archives.
- Version: 7.0.1

4. Node.js and npm:

- Purpose: Server-side JavaScript runtime and package management.
- Node.js Version: 21.0.0
- npm Version: 21.0.0

5. Electron Store:

- Purpose: Simplifies the storage of user preferences and other app data.
- Version: 8.2.0

6. Visual Studio Code:

- Purpose: Integrated development environment (IDE) for writing and managing code efficiently.

7. Nodemon:

- Purpose: Monitor changes in the application and automatically restart the server during development.
- Version: 3.1.0

8. Figma:

- Purpose: Design tool for creating user interface designs, system diagrams, and other graphical assets.

TARGET AUDIENCE

1. Developers:

- a. Individuals or teams involved in building, maintaining, or extending the Bfarm application.
- b. Those seeking to understand the architecture, code structure, and implementation details of Bfarm.

2. System Administrators:

- a. Professionals responsible for deploying, configuring, and managing Bfarm in production environments.
- b. Those interested in learning about system requirements, installation procedures, and server setup.

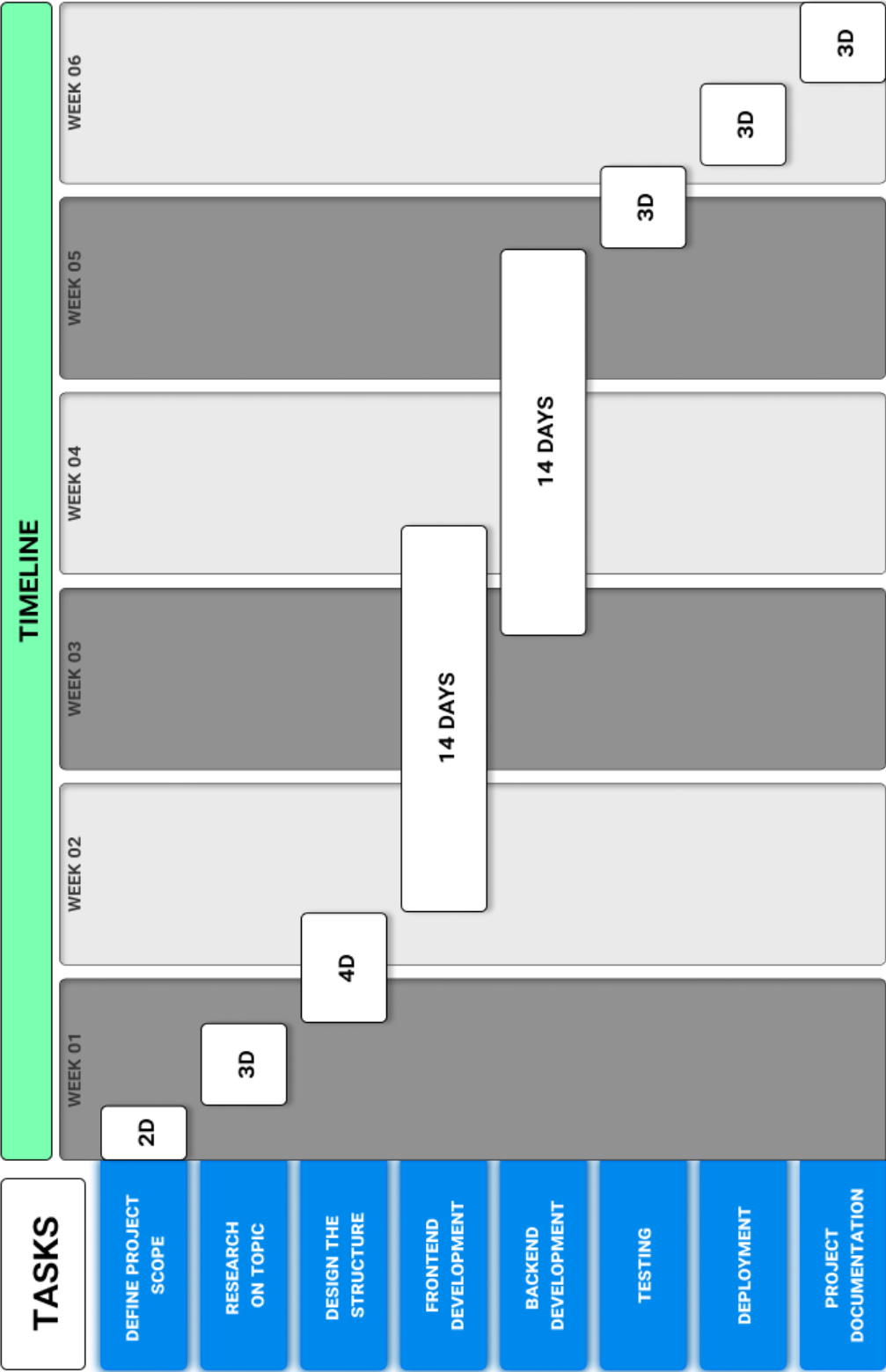
3. Blender Users:

- a. Artists, animators, and designers who utilize Blender for 3D modeling, animation, and rendering.
- b. Users looking to leverage Bfarm's cloud rendering capabilities to expedite their rendering workflows.

4. Technical Enthusiasts:

- a. Hobbyists or enthusiasts interested in exploring cloud-based rendering technologies and electron-based desktop applications.
- b. Individuals keen on understanding the underlying technologies and concepts employed in Bfarm.

GANTT CHART



REQUIREMENT SPECIFICATIONS

1. Backend Server:

- a. Node.js runtime environment with a minimum version of 14.x.

2. Frontend Client:

- a. Modern web browser with JavaScript enabled.
- b. Network connectivity for accessing the Bfarm service.

3. External Services:

- a. Firebase services including Firestore, Authentication, Storage and Analytics etc.

4. Development Tools:

- a. Code Editor (Visual Studio Code).
- b. npm (Node Package Manager) for managing project dependencies.
- c. Web browser (Chrome, Microsoft Edge, or Firefox).

Functional Requirements:

1. Single Page Application:

- a. Bfarm should function as a single-page application, reducing server and client bandwidth by loading content dynamically without refreshing the page.

2. Authentication and Authorization:

- a. Role-based Access: Different permissions and access levels for administrators, render node operators, and clients.
- b. Login/Logout System: Provides a seamless login and logout system with proper redirections.

3. Project Management:

- a. Create Project: Users can create new rendering projects, specifying parameters such as start and end frames, GPU usage, and rendering engine.
- b. Monitor Project: Users can monitor the status of their rendering projects, including running, completed, or stopped.

- c. **Stop Project:** Users can stop rendering projects in progress if needed.
- 4. **Rendering Control:**
 - a. **Start Rendering:** Users can initiate rendering for their projects, specifying the rendering path, GPU usage, and rendering engine.
 - b. **Stop Rendering:** Users can stop rendering for their projects if needed.
- 5. **Project Presentation:**
 - a. **View Projects:** Users can view a list of their rendering projects, including project name, status, start and end frames, GPU usage, and rendering engine.
 - b. **Project Details:** Users can view detailed information about each rendering project, including project ID, file name, total frames, rendered frames, status, and date.

Non-Functional Requirements:

- 1. **Performance:**
 - a. Bfarm should efficiently handle concurrent user requests, ensuring fast response times and smooth user experience.
 - b. Web pages should load quickly to provide a seamless browsing experience for users.
- 2. **Scalability:**
 - a. Design Bfarm to scale horizontally to accommodate a growing number of users and rendering projects.
 - b. Optimize database queries and rendering processes for efficient performance as the user base expands.
- 3. **Usability:**
 - a. The user interface should be intuitive and user-friendly, allowing users to navigate and interact with Bfarm easily.
 - b. Provide informative feedback and error messages to guide users in using the platform effectively.
- 4. **Reliability:**
 - a. Bfarm should be highly available and reliable, with minimal downtime for maintenance or upgrades.
 - b. Implement mechanisms for error handling and recovery to ensure uninterrupted service for users.

FEASIBILITY ANALYSIS

1. Technical Feasibility:

- a. **Software Requirements:** The project requires technologies like Electron for the desktop application, Node.js for the backend, and React for the frontend. These technologies are well-established and widely used, making it feasible to build the application.
- b. **Integration with External Services:** Integration with MongoDB Atlas for data storage and Render for hosting services is technically feasible and offers scalability options.
- c. **Development Resources:** Availability of developers with expertise in Electron, Node.js, and React is crucial. Adequate resources for development, testing, and deployment are necessary for technical feasibility.

2. Economic Feasibility:

- a. **Cost of Development:** Developing a rendering farm management application involves expenses related to software development, infrastructure setup, and ongoing maintenance.
- b. **Revenue Generation:** The project's revenue model, such as subscription-based pricing for users, can determine its economic feasibility. Analysis of potential revenue streams against development and operational costs is essential.

3. Operational Feasibility:

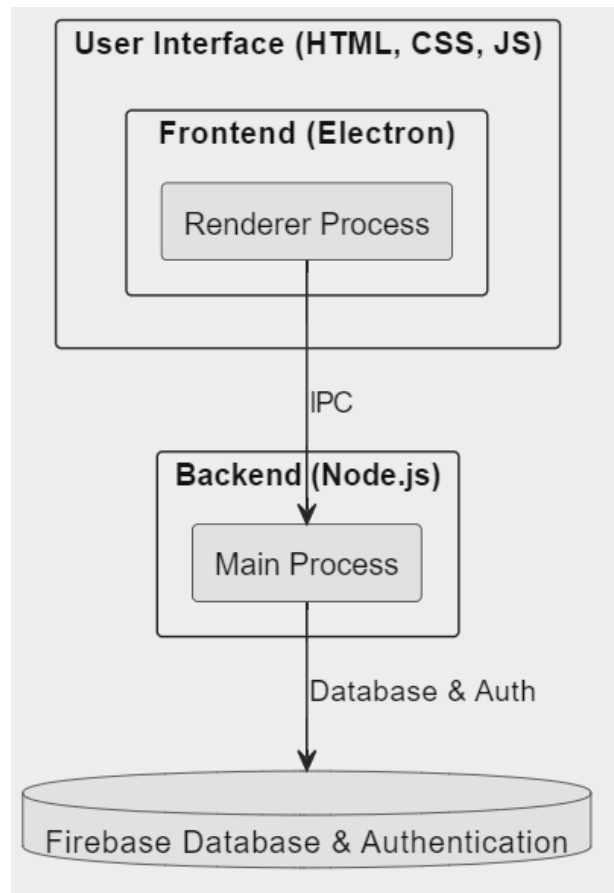
- a. **User Requirements:** Understanding the needs of rendering farm users (such as artists, animators, and filmmakers) and addressing their pain points through Bfarm is crucial for operational feasibility.
- b. **Ease of Use:** Bfarm should be user-friendly, with intuitive interfaces for managing rendering projects, monitoring progress, and accessing rendering resources.
- c. **Scalability:** The system should be designed to scale efficiently as the user base and rendering demands grow. Operational feasibility depends on the ability to handle increased workloads without significant performance degradation.

4. Legal and Regulatory Feasibility:

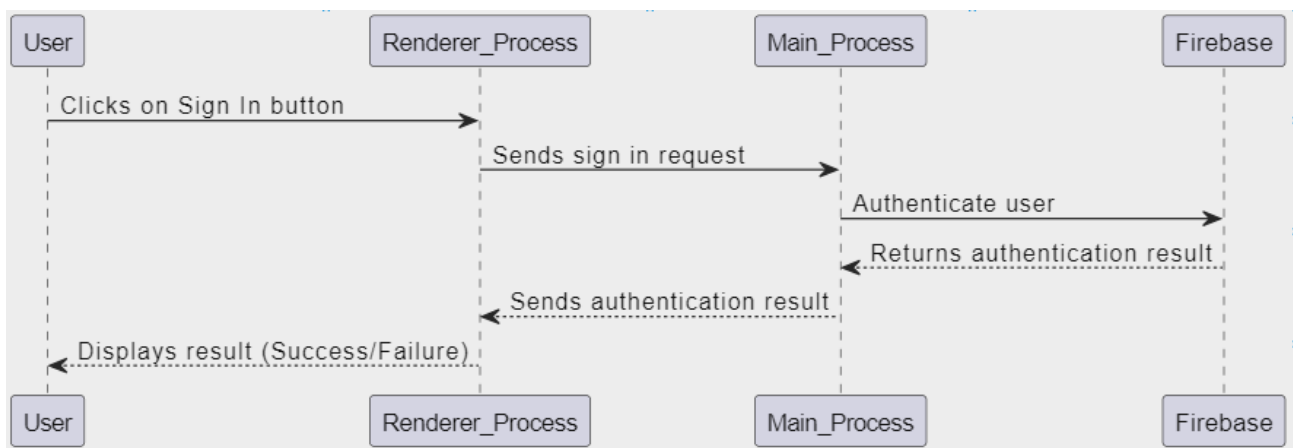
- a. **Compliance:** Ensuring compliance with relevant laws and regulations related to data privacy, intellectual property rights, and user agreements is essential.
- b. **Licensing:** Proper licensing of third-party software used in the project, along with adherence to open-source licensing requirements, is necessary to avoid legal issues.

SYSTEM DESIGN

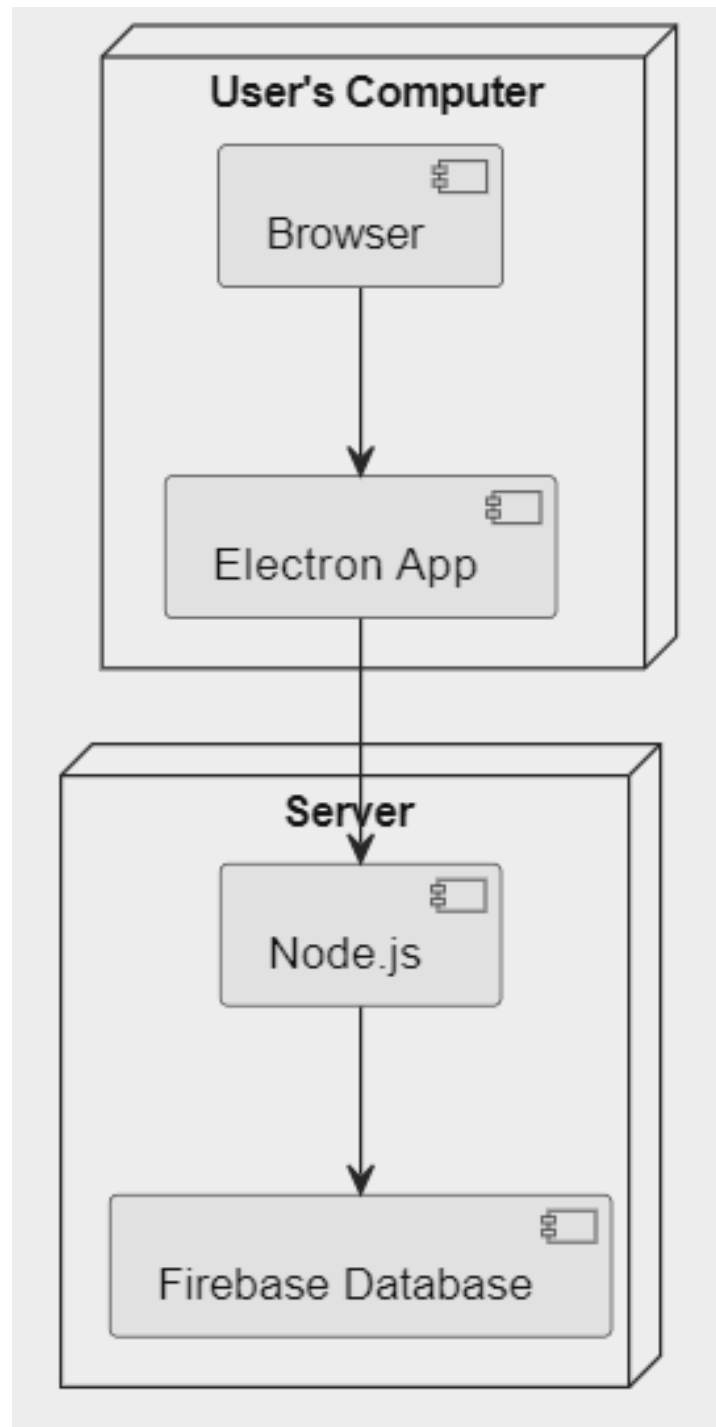
- **High-Level Architecture:**



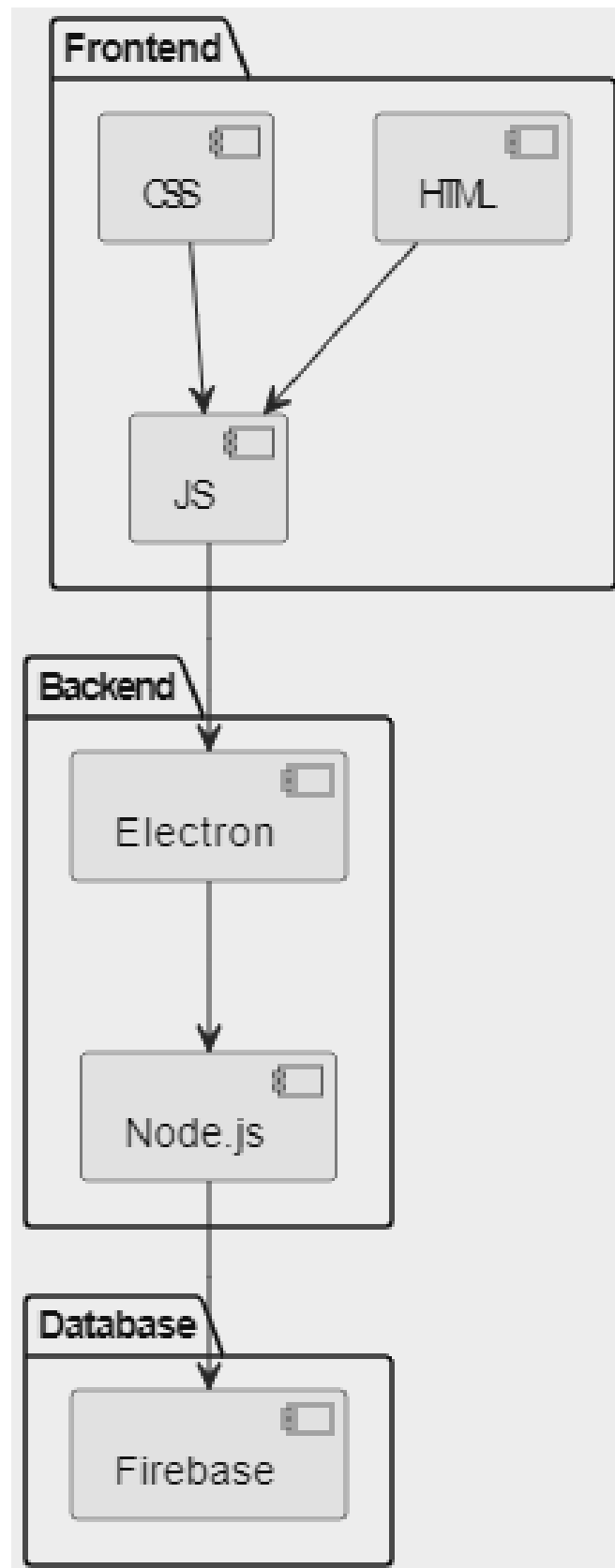
- **Sequence Diagram - User Authentication:**



- **Deployment Diagram**



- **Component Diagram:**



IMPLEMENTATION

1. Backend Development:

- a. Utilize Node.js runtime environment along with Electron for server-side development.
- b. Utilize Firebase SDK for integrating Firebase services like Firestore for database and Firebase Authentication for user authentication.

2. Frontend Development:

- a. Develop the user interface using HTML, CSS, and JavaScript.
- b. Implement responsive design to ensure compatibility across various devices and screen sizes.

3. Authentication and Authorization:

- a. Integrate Firebase Authentication to handle user authentication, including login, registration, and password reset functionalities.
- b. Implement role-based access control using Firebase Authentication custom claims or Firestore database to manage user roles and permissions.

4. Database Management:

- a. Utilize Firebase Firestore as the database for storing project data, user information, and other application data.
- b. Design the database schema to efficiently store and retrieve project details, user profiles, and authentication-related information.

5. File Upload and Rendering:

- a. Develop functionality for users to upload Blender project files using HTML file input elements.
- b. Implement rendering functionality using Blender software on the backend server.
- c. Utilize Firebase Cloud Storage for storing uploaded project files securely.

6. Real-time Updates:

- a. Utilize Firebase Realtime Database or Firestore's real-time listeners to provide real-time updates to users regarding rendering status changes, project uploads, etc.

7. User Profile Management:

- a. Develop features for users to manage their profiles, including updating personal information and profile pictures.
- b. Utilize Firebase Authentication to handle user profile management and authentication-related tasks.

8. Project Management:

- a. Create interfaces for users to view, edit, and delete their projects.
- b. Implement features for project status tracking, including rendering progress, completion status, etc., using Firebase Firestore.

9. Error Handling and Logging:

- a. Implement robust error handling mechanisms in both frontend and backend to gracefully handle exceptions and errors.
- b. Utilize Firebase Crashlytics or custom logging solutions to log relevant information for debugging and monitoring purposes.

10. Security Measures:

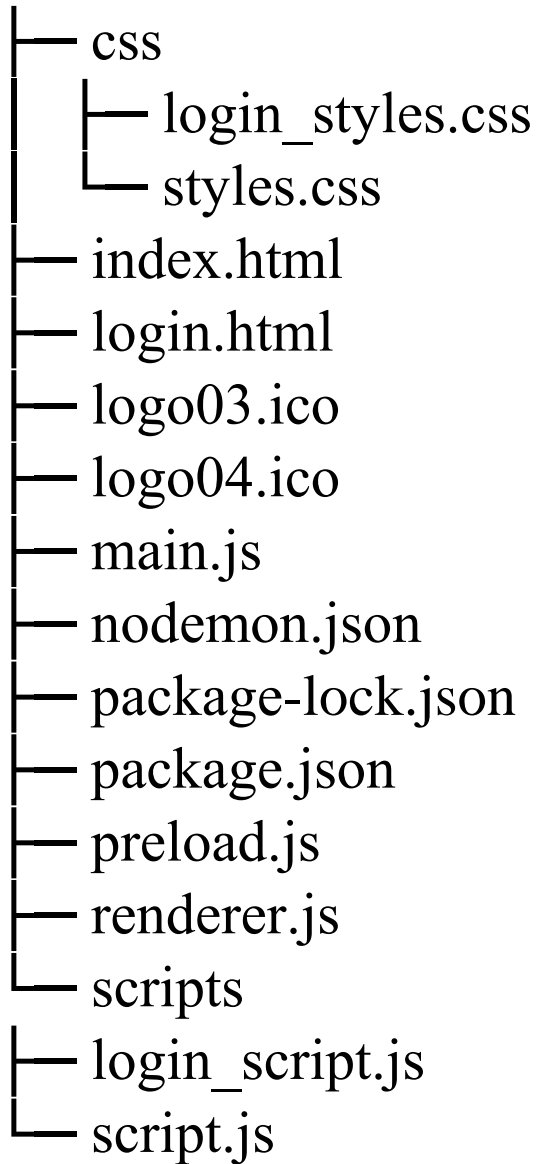
- a. Apply security best practices recommended by Firebase for securing Firebase services, including data validation, authorization rules, and security rules for Firestore.

11. Testing and Quality Assurance:

- a. Conduct unit tests and integration tests for both frontend and backend components to ensure the correctness and reliability of the application.
- b. Perform usability testing to gather feedback from users and make necessary improvements to the user interface.

Folder Structure

Bfarm02



Code Snippet

❖ Backend:

```
▪ Main.js-----
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path');
const { promisify } = require('util');
const { exec } = require('child_process');
const fs = require('fs-extra');
const Store = require('electron-store');
const store = new Store();
const archiver = require('archiver');

const firebase = require('firebase/app');
require('firebase/auth');
require('firebase/firestore');
require('firebase/storage');

const execAsync = promisify(exec);

const firebaseConfig = {

};

firebase.initializeApp(firebaseConfig);

let mainWindow;
const storage = firebase.storage();
const bucket = storage.ref();
const firestore = firebase.firestore();
const auth = firebase.auth();

let userID;
let userData;
let projectMap = { };

function createWindow() {

    const bounds = store.get('bounds');
    if(!store.get('bounds')){
        bounds = [800,600]
    }
    const position = store.get('position');
```

```

if(!store.get('position')){
    position = [100,100];
}

mainWindow = new BrowserWindow({
    width: bounds[0],
    height: bounds[1],
    minWidth: 600,
    minHeight: 575,
    x: position[0],
    y: position[1],
    icon: path.join(__dirname, 'logo03.ico'),
    autoHideMenuBar: true,
    webPreferences: {
        nodeIntegration: true,
        contextIsolation: true,
        preload: path.join(__dirname, 'preload.js')
    }
});

//save window size-----
mainWindow.on('resize', () => {
    store.set('bounds', mainWindow.getSize());
    //console.log(store.get('bounds'));
});
//save window position-----
mainWindow.on('move', () => {
    store.set('position', mainWindow.getPosition());
    //console.log(store.get('position'));
});

if(store.get('userID') !== undefined){
    //auto signin-----
    -----
    mainWindow.loadFile(path.join(__dirname, 'index.html'));
    userID = store.get('userID');
    //console.log(userID);
    userData = store.get('userData');
    //console.log(userData);
    projectMap = store.get('proData');
    //console.log(projectMap);

    updater();
}else{
    mainWindow.loadFile(path.join(__dirname, 'login.html'));
}

```

```

}

app.whenReady().then(createWindow);

// real time updater-----
-----
async function updater(){
  console.log('update
block*****
*****');
  firestore.collection('users').doc(userID).onSnapshot(snapshot => {

    console.log('update check-----');
    fetchProjects(userID);
  });
}

```

▪ **Render.js-----**

```

const fileInput = document.getElementById('fileInput');
const uploadButton = document.getElementById('uploadButton');
const renderButton = document.getElementById('renderButton');
const stopRender = document.getElementById('stopRendering');
const signIn = document.getElementById('signIn');
const signUp = document.getElementById('signUp');
const logout = document.getElementById('logout');

let file_path;

if(signIn){
  signIn.addEventListener('click', (event) => {
    event.preventDefault(); // Prevent default form submission behavior

    const email = document.getElementById('#email').value;
    const password = document.getElementById('#password').value;

    if(email && password){
      ipcRenderer.send('signIn',email,password);
    }
    console.log('signIn : ' + email + ' , ' + password);
  });
}

if(signUp){

```

```

signUp.addEventListener('click', (event) => {
    event.preventDefault();

    const username = document.getElementById('username').value;
    const email = document.getElementById('email').value;
    const password = document.getElementById('password').value;

    if(username && email && password){
        ipcRenderer.send('signUp',username,email,password);
    }
    console.log('signUp :'+ username +', '+email+', '+password);
});
}

if(logout){
    logout.addEventListener('click', (event) => {
        ipcRenderer.send('logout');
        console.log('logout signal send');
    });
}

if(fileInput){
    fileInput.addEventListener('change', (event) => {
        file_path = event.target.files[0].path;
        console.log('Selected file:', file_path);
    });
}

if(uploadButton){
    uploadButton.addEventListener('click', (event) => {
        const startFrame = document.getElementById('sframe').value;
        const endFrame = document.getElementById('eframe').value;
        const gpu = document.getElementById('uStatus');
        const useGpu = gpu.getAttribute('data-value');
        const engine = document.getElementById('uEngine');
        const useEngine = engine.getAttribute('data-value');

        if (startFrame && endFrame && useGpu && file_path && useEngine) {
            ipcRenderer.send('uploadFile', startFrame, endFrame, useGpu, file_path,
useEngine);

            uploadButton.style.backgroundColor = '#62EE9A';
            uploadButton.textContent = 'uploading...';
        }
        console.log('uploadButton');
    });
}

```



```

    }
    ipcRenderer.on('fileUploaded', (event) => {
        uploadButton.style.backgroundColor = '#0070C1';
        uploadButton.textContent = 'Upload File';
    });

    if(renderButton){
        renderButton.addEventListener('click',(event) => {
            const iPath = document.getElementById('insertPath').value;
            const rStatus = document.getElementById('rStatus');
            const useGpu = rStatus.getAttribute('data-value');
            const rEngine = document.getElementById('rEngine');
            const useEngine = rEngine.getAttribute('data-value');

            if(iPath && useGpu && useEngine){
                renderButton.style.display = 'none';
                stopRender.style.display = 'block';
                ipcRenderer.send('renderFile', iPath, useGpu, useEngine);
                console.log('start rendering');
            }
        });
    }
}

```

- **Preload.js**-----


```

//preload.js
const { contextBridge, ipcRenderer } = require('electron');

contextBridge.exposeInMainWorld('electron', {
    ipcRenderer: ipcRenderer
});

contextBridge.exposeInMainWorld('ipcRenderer', {
    send: (channel, ...args) => ipcRenderer.send(channel, ...args),
    on: (channel,...args) => ipcRenderer.on(channel, ...args),
});
            
```

❖ Frontend:

- **HTML**-----


```

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
            
```

```

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.2/css/all.min.css">
    <link rel="stylesheet" href="css/login_styles.css">
    <title>login Bfarm</title>
</head>

<body>

    <div class="container" id="container">
        <div class="form-container sign-up">
            <form>
                <h1>Create Account</h1>
                <div class="social-icons">
                    <a href="#" class="icon"><i class="fa-brands fa-google-plus-
g"></i></a>
                    <a href="#" class="icon"><i class="fa-brands fa-facebook-
f"></i></a>
                    <a href="#" class="icon"><i class="fa-brands fa-github"></i></a>
                    <a href="#" class="icon"><i class="fa-brands fa-linkedin-
in"></i></a>
                </div>
                <span>or use your email for registration</span>
                <input id="username" type="text" placeholder="Name">
                <input id="email" type="email" placeholder="Email">
                <input id="password" type="password" placeholder="Password">
                <button id="signUp">Sign Up</button>
            </form>
        </div>
        <div class="form-container sign-in">
            <form>
                <h1>Sign In</h1>
                <div class="social-icons">
                    <a href="#" class="icon"><i class="fa-brands fa-google-plus-
g"></i></a>
                    <a href="#" class="icon"><i class="fa-brands fa-facebook-
f"></i></a>
                    <a href="#" class="icon"><i class="fa-brands fa-github"></i></a>
                    <a href="#" class="icon"><i class="fa-brands fa-linkedin-
in"></i></a>
                </div>
                <span>or use your email password</span>
                <input id="#email" type="email" placeholder="Email">
                <input id="#password" type="password" placeholder="Password">
                <a href="#">Forget Your Password?</a>
                <button id="signIn">Sign In</button>
            </form>

```

```

    </div>
    <div class="toggle-container">
      <div class="toggle">
        <div class="toggle-panel toggle-left">
          <h1>Welcome Back!</h1>
          <p>Enter your personal details to use all of site features</p>
          <button class="hidden" id="login">Sign In</button>
        </div>
        <div class="toggle-panel toggle-right">
          <h1>Hello, Friend!</h1>
          <p>Register with your personal details to use all of site features</p>
          <button class="hidden" id="register">Sign Up</button>
        </div>
      </div>
    </div>
  </div>

  <script src="renderer.js"></script>
  <script src="scripts/login_script.js"></script>

</body>

</html>

```

- **CSS**-----


```

      @import
      url('https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;500;600;700&display=swap');

      :root {
        --blue: #0070C1;
        --green: #62EE9A;
        --light: #f6f6f9;
        --primary: #1976D2;
        --light-primary: #CFE8FF;
        --grey: #eee;
        --dark-grey: #AAAAAA;
        --dark: #2b3f49;
        --black: #0f171b;
        --danger: #D32F2F;
        --light-danger: #FECDD3;
        --warning: #FBC02D;
        --light-warning: #FFF2C6;
        --success: #388E3C;
        --light-success: #BBF7D0;
      }
      
```

```

*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Montserrat', sans-serif;
}

body{
  background-color: var(--light-primary);
  background: linear-gradient(to right, #e2e2e2, #c9d6ff);
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction: column;
  height: 100vh;
}

.container{
  background-color: var(--light);
  border-radius: 10px;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.35);
  position: relative;
  overflow: hidden;
  width: 768px;
  max-width: 100%;
  min-height: 480px;
}

.container p{
  font-size: 14px;
  line-height: 20px;
  letter-spacing: 0.3px;
  margin: 20px 0;
}

.container span{
  font-size: 12px;
}

.container a{
  color: #333;
  font-size: 13px;
  text-decoration: none;
  margin: 15px 0 10px;
}

```

```

.container button{
  background-color: var(--blue);
  /* background: linear-gradient(to bottom, var(--green), var(--blue)); */
  color: #fff;
  font-size: 12px;
  padding: 10px 45px;
  border: 1px solid transparent;
  border-radius: 8px;
  font-weight: 600;
  letter-spacing: 0.5px;
  text-transform: uppercase;
  margin-top: 10px;
  cursor: pointer;
}

.container button.hidden{
  background-color: transparent;
  border-color: var(--light);
}

.container form{
  background-color: var(--light);
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction: column;
  padding: 0 40px;
  height: 100%;
}

```

■ **JavaScript**-----

```

// menu button logic-----
const uploadArea = document.getElementById('uploadArea');
const renderArea = document.getElementById('renderArea');
const FilesArea = document.getElementById('filesArea');
const profileArea = document.getElementById('profileArea');

let mainArea = document.querySelectorAll('#main #mainBox main');
let menubtn = document.querySelectorAll('#bar ul li');
function activeLink(){
  menubtn.forEach((item) =>
    item.classList.remove('active'));
  this.classList.add('active');

  console.log(this.getAttribute('value'));
}

```

```

    if(this.getAttribute('value') != 'logout'){
      mainArea.forEach((area) =>
        area.style.display = 'none');

      document.getElementById(this.getAttribute('value')).style.display = 'flex';
    }
  }

  menubtn.forEach((item) =>
    item.addEventListener('click',activeLink));

  // const dataBox = document.querySelectorAll('#projetScroll #dataBox');
  // dataBox.forEach((box) => {
  //   box.addEventListener('click', (event) => {
  //     mainArea.forEach((area) =>
  //       area.style.display = 'none');

  //     document.getElementById('fileArea').style.display = 'flex';
  //   });
  // });

  // toggle button logic-----
  //upload page
  const uState = document.getElementById('uStatus');
  const uStateBtn =
  document.getElementById('uStatus').querySelector('#statusBtn');
  uState.addEventListener('click', (event) => {
    const stateValue = uState.getAttribute('data-value');
    if(stateValue == 'no'){
      uState.setAttribute('data-value', 'yes');
      uStateBtn.style.left = '25px';
      uStateBtn.style.backgroundColor = '#62EE9A';
    }else{
      uState.setAttribute('data-value', 'no');
      uStateBtn.style.left = '0px';
      uStateBtn.style.backgroundColor = '#0070C1';
    }
  });

```

DEPLOYMENT

1. Packaging Electron Application:

- a. Utilize Electron Builder or similar tools to package the Electron application into platform-specific binaries (e.g., .exe for Windows, .dmg for macOS, .AppImage for Linux).
- b. Configure Electron Builder settings such as platform, architecture, and target directory for packaging the application.

2. Inno Setup for Windows Deployment:

- a. Utilize Inno Setup, a free script-driven installation system, to create a Windows installer (.exe) for distributing the Electron application.
- b. Write a script (.iss file) specifying installation details such as application name, version, installation directory, shortcuts, and additional files to include.

3. Configuration and Customization:

- a. Configure Inno Setup script to include Electron application binaries, dependencies, and required resources in the installation package.
- b. Customize installer appearance, including welcome screens, license agreements, and installation options, to match the application's branding and requirements.

4. Compilation and Building Installer:

- a. Compile the Inno Setup script using the Inno Setup Compiler to generate the Windows installer executable (.exe) file.
- b. Verify the generated installer to ensure it includes all necessary files, resources, and configurations for successful deployment.

5. Testing and Validation:

- a. Test the generated installer on various Windows environments to ensure compatibility and functionality across different versions and configurations.
- b. Validate the installation process, including application installation, shortcut creation, and post-installation tasks, to identify and resolve any issues or errors.

RESULTS

Login Page (Sign in)

The screenshot shows a web browser window titled "login Bfarm". The page features a light blue gradient background. In the center, there is a white card with a rounded top-left corner. The card is divided into two sections. The left section, titled "Sign In", contains social login buttons for Google+, Facebook, Twitter, and LinkedIn, followed by the text "or use your email password". Below this are input fields for "Email" and "Password", a "Forgot Your Password?" link, and a blue "SIGN IN" button. The right section, titled "Hello, Friend!", contains the text "Register with your personal details to use all of site features" and a white "SIGN UP" button.

Login Page (Sign up)

The screenshot shows a web browser window titled "login Bfarm". The page features a light blue gradient background. In the center, there is a white card with a rounded top-left corner. The card is divided into two sections. The left section, titled "Welcome Back!", contains the text "Enter your personal details to use all of site features" and a white "SIGN IN" button. The right section, titled "Create Account", contains social login buttons for Google+, Facebook, Twitter, and LinkedIn, followed by the text "or use your email for registration". Below this are input fields for "Name", "Email", and "Password", and a blue "SIGN UP" button.

Login Page (vertical)

login Bfarm

Sign In

G+ f

or use your email password

Email

Password

[Forget Your Password?](#)

SIGN IN

Hello, Friend!

Register with your personal details to use all of site features

SIGN UP

Dashboard (light mode)

The screenshot shows the B-FARM dashboard in light mode. The interface is divided into three main sections: a left sidebar, a central main area, and a right sidebar.

Left Sidebar: Contains a blue header with the B-FARM logo and a toggle switch. Below the header are four buttons: "UPLOAD" (green), "RENDER" (blue), "USER" (blue), and "LOG OUT" (blue).

Central Main Area: Titled "UPLOAD FILE", it contains a form for uploading files. The form includes a section for "ENTER FRAME RANGE" with "START FRAME" and "END FRAME" input fields. Below this is a toggle for "ENABLE GPU" (currently on). The "CHOOSE RENDER ENGINE" section has two options: "EEVEE" (selected) and "CYCLE". A "CHOOSE FILE" button is present, followed by the text "NO FILE CHOSEN". At the bottom is a large blue "UPLOAD FILE" button.

Right Sidebar: Titled "PROJECTS", it lists four test projects: "TEST_3", "TEST_2", "TEST_1", and "TEST_5". Each project entry shows the number of frames (10) and the date/time of the last render.

Project Name	Frames	Date/Time
TEST_3	10	04/09/24, 9:39 PM
TEST_2	10	04/09/24, 9:26 PM
TEST_1	10	04/12/24, 5:03 PM
TEST_5	10	04/12/24, 6:18 PM

Dashboard (dark mode)

The screenshot shows the B-FARM dashboard in dark mode. The interface is divided into three main sections: a left sidebar, a central main area, and a right sidebar.

Left Sidebar: Contains a blue header with the B-FARM logo and a toggle switch. Below the header are four buttons: "UPLOAD" (green), "RENDER" (blue), "USER" (blue), and "LOG OUT" (blue).

Central Main Area: Titled "UPLOAD FILE", it contains a form for uploading files. The form includes a section for "ENTER FRAME RANGE" with "START FRAME" and "END FRAME" input fields. Below this is a toggle for "ENABLE GPU" (currently on). The "CHOOSE RENDER ENGINE" section has two options: "EEVEE" (selected) and "CYCLE". A "CHOOSE FILE" button is present, followed by the text "NO FILE CHOSEN". At the bottom is a large blue "UPLOAD FILE" button.

Right Sidebar: Titled "PROJECTS", it lists four test projects: "TEST_3", "TEST_2", "TEST_1", and "TEST_5". Each project entry shows the number of frames (10) and the date/time of the last render.

Project Name	Frames	Date/Time
TEST_3	10	04/09/24, 9:39 PM
TEST_2	10	04/09/24, 9:26 PM
TEST_1	10	04/12/24, 5:03 PM
TEST_5	10	04/12/24, 6:18 PM

Upload section

The screenshot shows the 'B-FARM' application window. The left sidebar contains buttons for 'UPLOAD' (highlighted in green), 'RENDER', 'USER', and 'LOG OUT'. The main area is titled 'UPLOAD FILE' and includes a form for uploading files. The form has a section for 'ENTER FRAME RANGE' with 'START FRAME' and 'END FRAME' input fields. Below this is an 'ENABLE GPU' toggle switch, which is currently turned on. The 'CHOOSE RENDER ENGINE' section has two options: 'EEVEE' (selected) and 'CYCLE'. A 'CHOOSE FILE' button is present, with the text 'NO FILE CHOSEN' next to it. At the bottom of the form is a large blue 'UPLOAD FILE' button. The right sidebar, titled 'PROJECTS', lists four test projects: 'TEST_3', 'TEST_2', 'TEST_1', and 'TEST_5', each with a status indicator and a timestamp.

[B-FARM] APPLE USER

UPLOAD FILE

ENTER FRAME RANGE

START FRAME TO END FRAME

ENABLE GPU : ☒

CHOOSE RENDER ENGINE : ☒ EEVEE ☐ CYCLE

CHOOSE FILE NO FILE CHOSEN

UPLOAD FILE

PROJECTS

TEST_3 ☒ FRAMES : 10 04/09/24, 9:39 PM

TEST_2 ☒ FRAMES : 10 04/09/24, 9:26 PM

TEST_1 ☒ FRAMES : 10 04/12/24, 5:03 PM

TEST_5 ☒ FRAMES : 10 04/12/24, 6:18 PM

Render section

The screenshot shows the 'B-FARM' application window. The left sidebar contains buttons for 'UPLOAD', 'RENDER' (highlighted in green), 'USER', and 'LOG OUT'. The main area is titled 'RENDER FILES' and includes a form for rendering files. The form has a 'PATH / TO / BLENDER.EXE' input field. Below this is an 'ENABLE GPU' toggle switch, which is currently turned on. The 'CHOOSE RENDER ENGINE' section has two options: 'EEVEE' (selected) and 'CYCLE'. At the bottom of the form is a large blue 'START RENDERING' button. The right sidebar, titled 'PROJECTS', lists four test projects: 'TEST_3', 'TEST_2', 'TEST_1', and 'TEST_5', each with a status indicator and a timestamp.

[B-FARM] APPLE USER

RENDER FILES

PATH / TO / BLENDER.EXE

ENABLE GPU : ☒

CHOOSE RENDER ENGINE : ☒ EEVEE ☐ CYCLE

START RENDERING

PROJECTS

TEST_3 ☒ FRAMES : 10 04/09/24, 9:39 PM

TEST_2 ☒ FRAMES : 10 04/09/24, 9:26 PM

TEST_1 ☒ FRAMES : 10 04/12/24, 5:03 PM

TEST_5 ☒ FRAMES : 10 04/12/24, 6:18 PM

Project list

PROJECTS

TEST_3

FRAMES : 10 04/09/24, 9:39 PM

TEST_2

FRAMES : 10 04/09/24, 9:26 PM

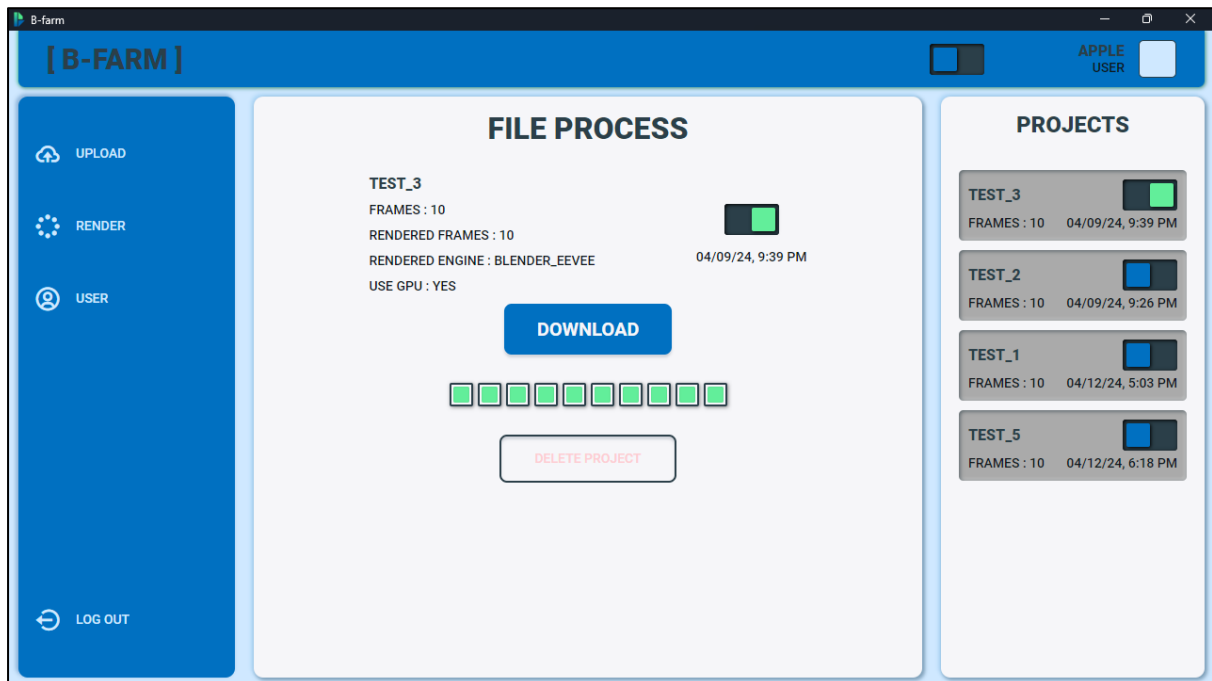
TEST_1

FRAMES : 10 04/12/24, 5:03 PM

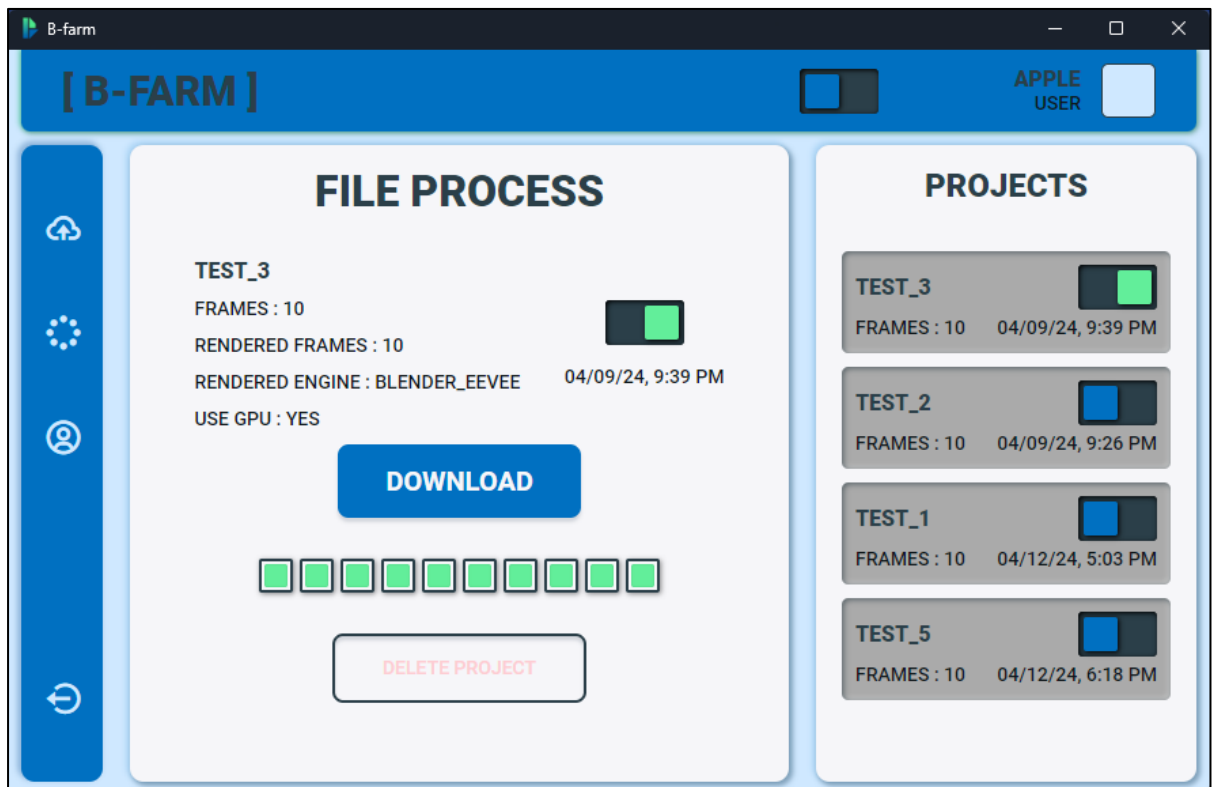
TEST_5

FRAMES : 10 04/12/24, 6:18 PM

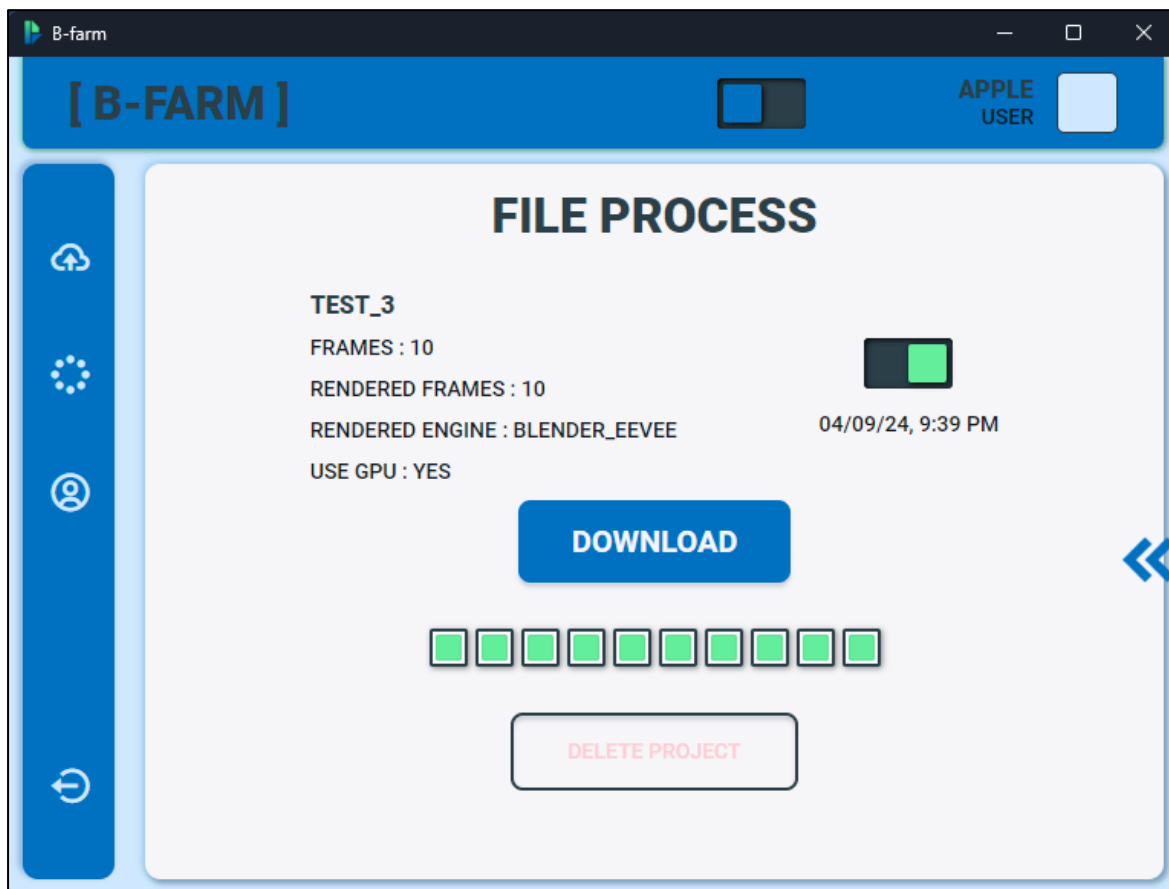
Project processing



Dashboard (mid-size)



Dashboard (low-size)



CONCLUSION AN FUTURE SCOPE

Conclusion:

In conclusion, Bfarm presents a promising solution for managing rendering tasks efficiently in creative industries. Through its development, we've established technical feasibility by leveraging robust technologies like Electron, Node.js, and React. Economic feasibility is supported by a revenue model focused on subscription-based pricing and cost analysis. Operationally, Bfarm aims to address user requirements with intuitive interfaces, scalability, and ease of use.

Legal and regulatory compliance, including data privacy and licensing, has been integrated into the project's development process. Market feasibility has been assessed through demand analysis and differentiation from existing competitors.

With these considerations, Bfarm is poised to offer significant value to its target audience of artists, animators, and filmmakers seeking a streamlined rendering farm management solution.

Future Scope:

Moving forward, Bfarm has several avenues for enhancement and expansion:

- **Point System:** Implementing a point system to incentivize user engagement and reward active participation.
- **Rendering Limitations:** Introducing limitations on rendering and project uploads to optimize resource allocation and enhance platform efficiency.
- **Data Security:** Strengthening data security measures to safeguard user information and project assets.
- **Enhanced Efficiency:** Continuously improving platform efficiency through performance optimizations and user feedback-driven enhancements.
- **Collaborative Rendering:** Introducing collaborative rendering options to enable users to work together on rendering projects and share resources.

- **Friend Rendering:** Facilitating friend rendering through token-based access, fostering collaboration and community engagement.
- **User Profile Enhancements:** Enhancing user profiles with additional features, such as achievements, badges, and performance metrics, to provide a more personalized and rewarding experience.


By incorporating these future enhancements, Bfarm aims to stay ahead of the curve and remain a leading solution in the rendering farm management space, offering users unparalleled efficiency, collaboration opportunities, and security.

REFERENCES

1. **Node.js. (n.d.). Node.js.** Retrieved from <https://nodejs.org/>
2. **Electron. (n.d.).** Build cross-platform desktop apps with JavaScript, HTML, and CSS. Retrieved from <https://www.electronjs.org/>
3. **Firebase. (n.d.). Firebase.** Retrieved from <https://firebase.google.com/>
4. **Archiver. (n.d.). Archiver** - a streaming interface for archive generation. Retrieved from <https://www.archiverjs.com/>
5. **Electron Store. (n.d.). Electron Store.** Retrieved from <https://www.electronjs.org/docs/api/store>
6. **Visual Studio Code. (n.d.). Code Editing.** Redefined. Retrieved from <https://code.visualstudio.com/>
7. **Figma. (n.d.). Figma:** the collaborative interface design tool. Retrieved from <https://www.figma.com/>
8. **Blender. (n.d.). Blender** - a 3D creation suite for modeling, animation, simulation, rendering, compositing, motion tracking, and game creation. Retrieved from <https://www.blender.org/>
9. **Nodemon. (n.d.). nodemon.** Retrieved from <https://nodemon.io/>
10. **Various npm Packages.** Retrieved from respective npm package pages.
11. **Stack Overflow. (n.d.).** Retrieved from <https://stackoverflow.com/>
12. **W3Schools. (n.d.).** Retrieved from <https://www.w3schools.com/>

PLAGIARISM REPORT

- **Report from Plagiarism Detector**

Apr 28, 2024

Plagiarism Scan Report

0%

Plagiarized

100%

Unique

Characters:1302

Words:168

Sentences:8

Speak Time:
2 Min


Excluded URL

None

Content Checked for Plagiarism

The Bfarm project introduces a revolutionary solution to the time-consuming process of rendering Blender files by leveraging distributed cloud computing. This documentation provides a comprehensive overview of the Bfarm platform, detailing its architecture, functionality, and implementation process. The primary objective of Bfarm is to address the challenges faced by Blender users in rendering large projects efficiently. By harnessing the combined power of multiple users' systems connected through a centralized middleware, Bfarm significantly reduces rendering time and enhances productivity for Blender artists and learners. This documentation outlines the project's scope, methodology, and tools and technologies used in its development. It provides insights into the development process, including requirement analysis, system design, implementation, testing, and deployment phases. Through the Bfarm platform, Blender users gain access to high-performance rendering capabilities, irrespective of their system specifications, fostering collaboration and resource-sharing within the Blender community. The documentation aims to serve as a guide for understanding and utilizing the Bfarm platform effectively to streamline the rendering process and enhance productivity in Blender projects.

Sources



[Home](#) [Blog](#) [Testimonials](#) [About Us](#) [Privacy Policy](#)

Copyright © 2024 [Plagiarism Detector](#). All right reserved