# Comprehensive Exercise Report

Team <<5>> of Section <<005>>

Nicolas Cholin 250AEB034, Bekassyl Adenov 221ADB130, Rofig Ashumov 221ADB172, Batex Bafika 250ADB026, Ayham Saifan 250ADB025

NOTE: You will replace all placeholders that are given in <<>>

# Requirements/Analysis

Week 2

## Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- After reading the client's brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
    - <<LabCyberDocs streamlines protocol documentation uploads with seamless GitHub integration and management.>>
        - <<
        - Allow users to upload documentation for Projects, Events, and Machines.
        - Support multiple document types: Conception files, Fabrication files, Reports, Recordings, Presentations, Pictures, and Protocols.

        - Enable contributions from users without GitHub accounts (via local login).
        - Ensure upload compatibility from both desktop and mobile (responsive design).
        - Use a first dropdown to select the category (Project, Machine, or Event).
        - Use a second dropdown to select the file type (e.g., Recording, Report).
        - Use a third optional dropdown to specify license type (Open, Public, Private).
        - Automatically create Pull Requests to the GitHub repository for each upload.
        - Retrieve the existing folder structure from the repository and allow new branch/folder creation.
        - Display a visual summary of the upload with text.
        - Notify admins via email (labcyber@ptcc.fr) when a new contribution is submitted.
        - Allow admins to validate contributions with minimal steps via a dashboard.
        - Display the contributor's identity (GitHub username).
        - Show a thank-you or acknowledgment message upon successful contribution.
        - Comply with the PTCC graphic charter (Font: Trebuchet, Color: #5e22c3).
        - Include security measures.>>

- After reading the client's brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.
    - <<

**1. Do you want all uploads to be reviewed before going to GitHub?**

**Yes.** Each upload should create a Pull Request so an admin can review and approve it before it's added to the main repository.

**2. Can users create new folders in the Git repo, or should they use existing ones?**

**Both.** The platform should show the existing structure and also let users create new folders or branches when needed.

**3. Should people without GitHub accounts be able to register and log in?**

**Yes.** They should be able to register with email and password and recover their password if needed. We also want to set different access roles (like admin or contributor).>>

- Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
  <<No, the project covers familiar topics such as documentation systems, GitHub integration, file upload interfaces, and web-based role access control. However, for deeper understanding of best practices in FabLab's and open documentation standards, the following references were helpful:

  GitHub Docs – Working with Pull Requests>>

- Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
  <<
  **Start-ups** using LabCyber for prototyping and documenting their innovation process.
  **Institutions** (e.g., universities, labs) contributing to collaborative research or educational projects.

  **Academic or private organizations** managing machine usage or event activities within LabCyber.>>

- Describe how each user would interact with the software
  <<**Start-ups**:
  - Upload design iterations, fabrication files, and reports for their ongoing projects.
  - Document project progress for traceability and funding justification.
- **Institutions**:
  - Submit machine protocols or event-related materials (slides, recordings).
  - Ensure internal teams and students follow proper documentation formats.
- **Academic/private organizations**:
  - Upload procedural guides, validated machine files, and event documentation.
  - Access documentation archives for learning, collaboration, or replication.

∉ All users:
Log in (GitHub or local)
Choose a category (Project, Machine, Event)
Upload files and select file type/license
Submit for review (PR created automatically)
Receive confirmation and acknowledgment>>

- What features must the software have? What should the users be able to do?

<<

Security features against unsafe uploads or access
Admins receive notifications (email)
Automatically generate Pull Requests in the Git repo
Upload files for projects, machines, or events
Select file type and license (Open/Public/Private)>>

- Other notes:
  - <<The requirements were determined by the client during several meetings.>>

## Software Requirements

<<Use your notes from above to complete this section of the formal documentation by writing a detailed description of the project, including a paragraph overview of the project followed by a list of requirements (see lecture for format of requirements). You may also choose to include user stories.>>

The **LabCyberDocs** platform is a web-based documentation submission system designed to simplify and manage the upload of technical and procedural documents related to projects, machines, and events. It is intended for use by start-ups, institutions, and academic or private organizations collaborating within the LabCyber ecosystem. The goal of the platform is to streamline the contribution process by allowing users to submit various file types—such as conception files, fabrication files, reports, recordings, presentations, pictures, and protocols—through a responsive interface compatible with both desktop and mobile devices. Each submission automatically generates a Pull Request (PR) to a GitHub repository, where administrators can review and validate contributions via a dedicated dashboard.

### List of Requirements

- **Upon upload, the system must automatically generate a Pull Request to the GitHub repository.**
  This is central to the contribution workflow, ensuring all uploads go through version control and review.
- **Users must be able to log in via GitHub or register and log in locally using email and password.**
  This ensures accessibility for all users, regardless of whether they have a GitHub account.
- **Admins must be able to review and validate submissions easily via a dashboard.**
  Efficient admin review is critical for maintaining the quality and organization of contributions.
- **The platform must support file uploads for three categories: Projects, Machines, and Events.**
  This defines the structural organization of all documentation within the system.
- **The system must implement security measures to protect against unsafe file uploads and unauthorized access.**
  Security is crucial for protecting sensitive documents and maintaining system integrity.

# Black-Box Testing

Instructions: Week 4

## Journal

***Remember:*** Black box tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the black box testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

- What does input for the software look like (e.g., what type of data, how many pieces of data)?
  <<
  **Type of data:**

- Uploaded files (PDF, DOCX, JPEG)
- Text inputs (username, password, metadata like title/description)
- Dropdown selections (Category: Project, Machine, Event; File type; License type)
- Login credentials (email/password or GitHub OAuth token)
- Folder/branch selection or creation names (text)

  **How many pieces of data:**
- One file per submission>>

- What does output for the software look like (e.g., what type of data, how many pieces of data)?
  <<
- Confirmation messages (e.g., "Upload successful", "PR created", error messages)
- Pull Request creation on GitHub (metadata, filename, timestamp)
- Email notification to `labcyber@ptcc.fr`
- Admin dashboard entries with pending contributions and contributor info>>

- What equivalence classes can the input be broken into?
  <<
- **File type:**
  - Valid document types (Conception, Fabrication, Reports, etc.)
  - Invalid file types (e.g., `.exe`, `.bat`, unsupported formats)
- **Login method:**
  - Valid GitHub login
  - Valid local login
  - Invalid login (wrong credentials, inactive account)

- **Category selection:**
  - Valid categories: Project, Machine, Event

- **License type:**
  - Open, Public, Private
  - No license selected (optional)
- **Upload method:**
  - Single file>>

- What boundary values exist for the input?
  <<
- **File size limits:**
  - Minimum: 1 byte
  - Maximum: Near system/upload limit (25MB)

- **Text inputs:**

    - Empty string (should fail)
- **Number of uploads:**
- 0 files (Would not upload)
- 1 file (pass)
- Maximum allowed files (pass or fail if over limit)>>

- Are there other cases that must be tested to test all requirements?
  <<
- Upload with invalid file extension
- PR successfully created and linked to correct folder/branch
- Email notification successfully sent
- Responsive UI test (mobile vs. desktop)>>

- Other notes:
    - <<No need>>

# Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
|---|---|---|---|
| Test 1 | Upload a valid PDF report to "Project" with all dropdowns selected | Upload accepted, visual summary displayed, PR created, thank-you message shown | Same result |
| Test 2 | Upload a “.exe” file (invalid format) | Upload blocked, error message: "Unsupported file type" | Same result |
| Test 3 | Upload file size > 25MB (Larger than the system limit) | Upload blocked, error: "File size exceeds limit" | Error message: “Please check the form” |

# Design

Instructions: Week 6

## Journal

***Remember:*** You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- List the nouns from your requirements/analysis documentation.
  <<
- User
- Admin

- GitHub
- Pull Request
- Project
- Machine
- Event
- File
- File Type
- License
- Dashboard
- Category
- Folder
- Branch
- Metadata
- Notification
- Email
- Submission
- Token
- Login
- Confirmation
- Message
- Protocol
- Documentation>>

- Which nouns potentially may represent a class in your design?
  - <<
  - User - Represents application users (regular and admin)
  - Submission - Represents file submissions to the documentation
  - Project - Represents a project in the hierarchy
  - Machine - Represents a machine in the hierarchy
  - Event - Represents an event in the hierarchy
  - Notification - Represents notifications sent in the system>>

- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
  <<

- In User class:
  - username
  - email
  - isAdmin
  - isLocal (vs GitHub user)
  - token (for GitHub users)
- In Submission class:
  - file
  - fileName
  - hierarchy
  - type
  - license
  - submittedBy
  - submissionDate
  - status (pending, approved, rejected)
  - pullRequestUrl
- In Project/Machine/Event classes (Hierarchy items):
  - name
  - type (Project/Machine/Event)
  - files
  - parentHierarchy>>

- Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.

  <<

  Design Option 1: Direct Inheritance Hierarchy

  - User
  - ├── RegularUser
  - └── AdminUser
  - HierarchyItem
  - ├── Project
  - ├── Machine
  - └── Event
  - File
  - └── DocumentationFile
  - Submission
  - GitHubService
  - NotificationService
  - AuthenticationService
  - Pros:
    - Clear inheritance structure for hierarchy items
    - Specialized user types with different permissions
    - Separation of services
    - Simple to understand class structure
  - Cons:
    - Rigid hierarchy might be difficult to extend later
    - Inheritance may lead to tight coupling
    - May not adapt well if requirements change
    - Potentially redundant code across hierarchy item classes

Design Option 2: Composition-based Service Architecture

- User (with role attribute)
- HierarchyItem (with type attribute)
- File
- Submission
- Services:
- ├── GitHubService
- ├── NotificationService
- ├── UploadService
- ├── HierarchyService
- └── AuthenticationService

  - Pros:
    - More flexible through composition
    - Easier to modify and extend
    - Better separation of concerns with dedicated services
    - Avoids inheritance issues
    - More aligned with React's component architecture
  - Cons:
    - May require more interfaces between services
    - Can be harder to track relationships between components
    - Could lead to larger, more complex service classes>>

- Which design do you plan to use? Explain why you have chosen this design.
- List the verbs from your requirements/analysis documentation.

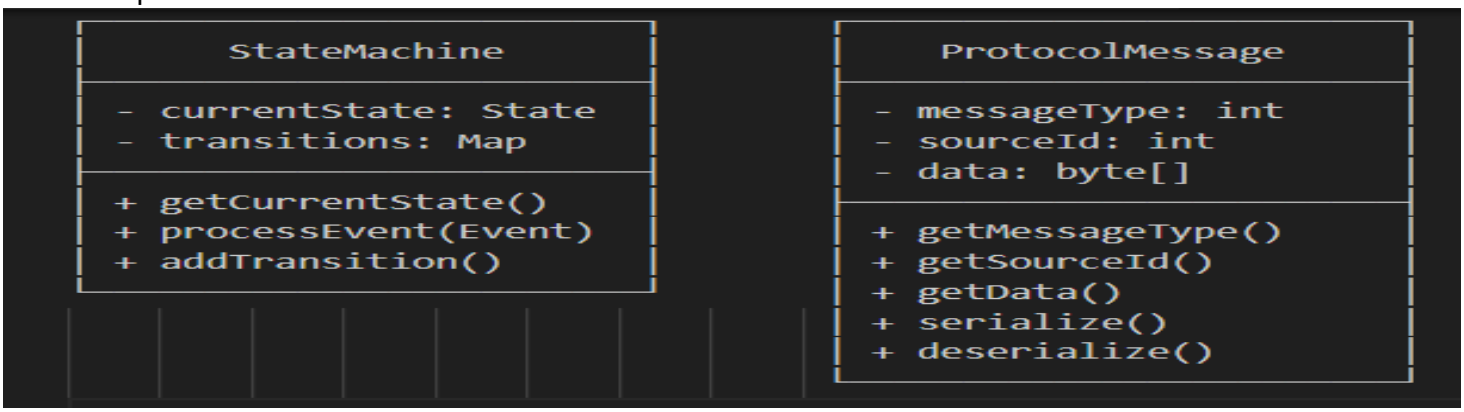
  <<
  - I plan to use Design Option 2: Composition-based Service Architecture because:
    - It aligns better with React's component-based architecture
    - It provides better flexibility for future extensions without breaking existing code
    - It separates concerns clearly through dedicated services
    - It avoids inheritance problems and allows for attribute-based type differentiation
    - It will likely be more maintainable in the long term
    - The composition approach better fits the dynamic nature of the application where items may need to change types or behaviors>>
- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.

  <<
  - Authenticate
  - Login
  - Review
  - Approve
  - Reject
  - Upload
  - Create
  - Select
  - Validate>>
- Other notes:
  - <<No notes>>

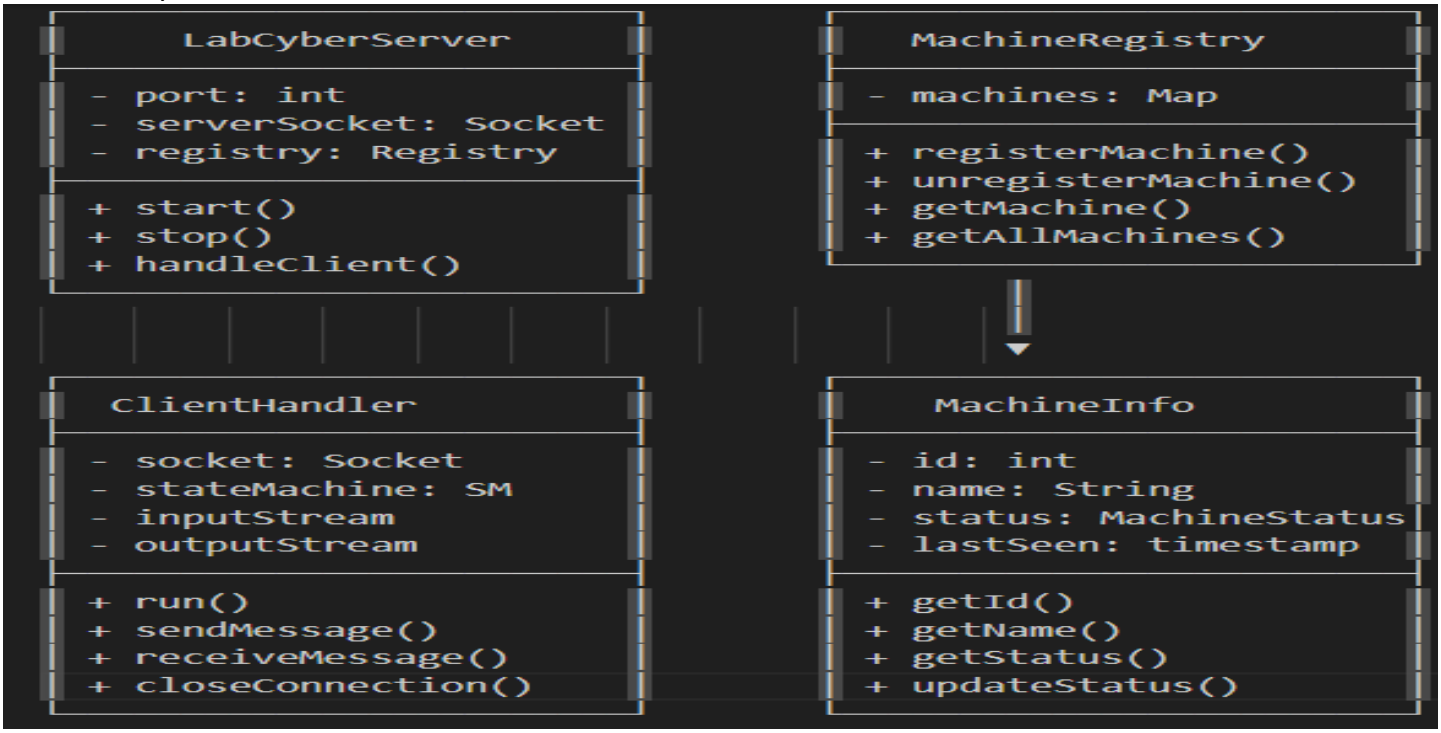
# Software Design

<<Use your notes from above to complete this section of the formal documentation by planning the classes, methods, and fields that will used in the software. Your design should include UML class diagrams along with method headers. ***Prior to starting the formal documentation, you should show your answers to the above prompts to your instructor.***>>

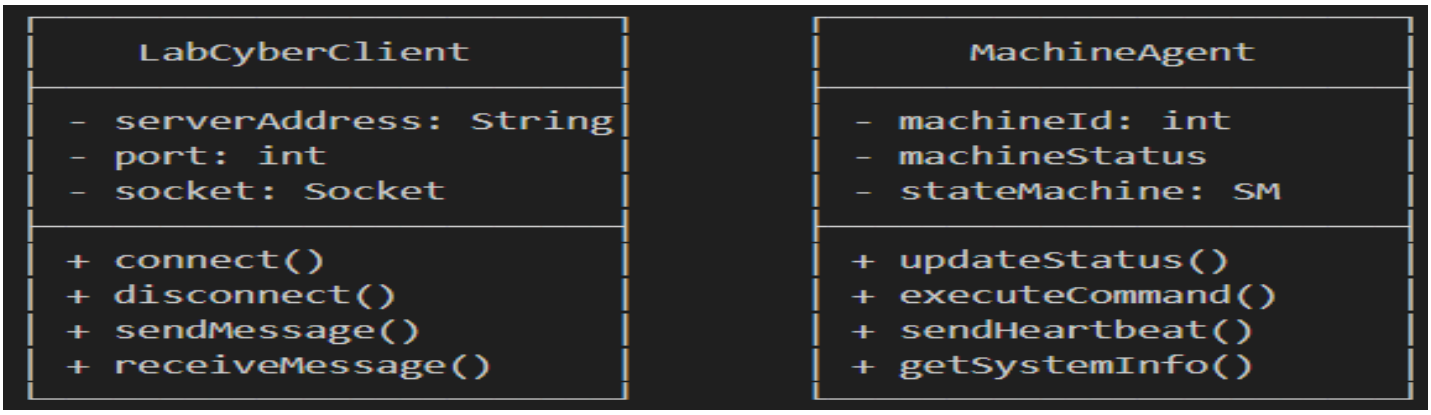
Core Components



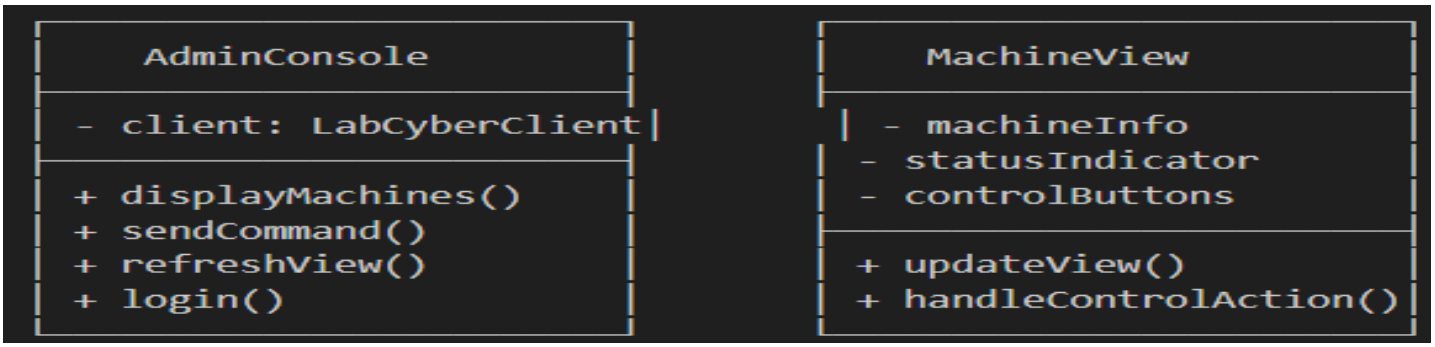

Server Components




Client Components




User Interface




# Implementation

Instructions: Week 8

## Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.
  <<
  - 1. Frontend-Backend Architecture
    - The project uses a clear separation between frontend (React) and backend (Node.js/Express) components, implementing a client-server architecture. The frontend handles user interface and interaction, while the backend manages authentication, file uploads, and external API communication.

  - 2. RESTful API Design
    - The backend implements RESTful API endpoints for:
    - Authentication (/auth/github, /logout, /user)
    - File uploads (/upload)
    - Admin notifications (/notify-admin)

  - 3. Authentication & Authorization
    - Multiple authentication strategies are implemented:
    - OAuth with GitHub (passport-github2)
    - Local authentication with encrypted storage

        - Admin authentication with password protection

    - 4. State Management
        - The React application uses useState and useEffect hooks for managing component state and side effects, particularly for:
        - User authentication state
        - Form data handling
        - File upload processing
        - Admin dashboard statistics

    - 5. Form Validation
        - Client-side validation with custom logic checks for:
        - Required fields
        - File size limits
        - Blocked file extensions
        - Input formatting requirements

    - 6. Error Handling
        - Comprehensive error handling throughout the application:
        - Try/catch blocks for API calls
        - User-friendly error messages
        - Form validation feedback
        - Network error handling

    - 7. Asynchronous Programming
        - The application uses async/await patterns extensively for:
        - API calls with axios
        - File reading operations
        - GitHub API integration
        - Email notifications

    - 8. Security Implementations
        - Several security measures are in place:
        - File type filtering to block dangerous extensions
        - Data encryption for sensitive information
        - Session management
        - CORS configuration>>
- Other notes:
    - <<no need>>

# Implementation Details

<<Use your notes from above to write code and complete this section of the formal documentation with a README for the user that explains how he/she will interact with the system.>>

LabCyber Machine Protocol Application

This application provides a platform for documenting machine protocols, projects, and events. Users can upload files to contribute to the documentation repository, while administrators can review and manage these contributions.

Getting Started

**Prerequisites**

- Node.js (v14 or higher)
- npm or yarn package manager
- GitHub account (optional)

**Installation**

**Clone the repository:**

git clone https://github.com/CabernetOgygiaVillaBanquet/LabCyber-Machine-Protocol-Application.git

cd LabCyber-Machine-Protocol-Application

**Install dependencies for both frontend and backend:**

- cd backend
- npm install
- cd ../frontend
- npm install

**Configure environment variables:**

Create a .env file in the backend directory with:

- GITHUB_CLIENT_ID=your_github_client_id
- GITHUB_CLIENT_SECRET=your_github_client_secret
- GITHUB_CALLBACK_URL=http://localhost:3001/auth/github/callback
- SESSION_SECRET=your_session_secret
- SMTP_EMAIL=your_notification_email
- SMTP_PASS=your_email_password
- ADMIN_EMAIL=admin_notification_recipient

**Create a .env file in the frontend directory with:**

- REACT_APP_GITHUB_TOKEN=your_github_personal_access_token

**Start the servers:**

- In the backend directory: npm start

- In the frontend directory: npm start

**User Guide**

Logging In

**Users have two login options:**

- GitHub Login: Click "Login with GitHub" to authenticate using your GitHub account
- Local Login: Use email and password to log in with a local account
- New users can create an account by clicking "Create an account"
- Password reset is available via "Forgot password?"

**Uploading Documentation**

- After logging in, you'll be directed to the upload dashboard
- Select a hierarchy (Project, Machine, Event) or create a new one
- Choose the appropriate file type for your document
- Drag and drop your file or click to upload (max 25MB)
- Optionally rename your file and select a license
- Click "Upload File" or "Create & Upload"
- Review the details in the confirmation modal and click "Confirm Upload"
- The system will create a pull request in the GitHub repository
- You'll receive a confirmation message when complete

**File Requirements**

Maximum file size: 25MB

Blocked file extensions: .exe, .bat, .cmd, .sh, .dll, .msi, .com, .ps1, .vbs, .zip

For larger video files, use the LabCyber Docs channel on MakerTube

# Testing

Instructions: Week 10

## Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
    - <<
    - File Upload Size Limit: Changed from initial requirement to 25MB limit (as seen in server.js line 111)
    - Authentication Methods: Added dual authentication (GitHub OAuth + Local login) instead of single method
    - File Type Restrictions: Added specific blocked extensions for security (.exe, .bat, etc.)
    - Hierarchy Types: Structured hierarchy into Projects, Machines, and Events with specific file types for each>>
- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
    - <<Insert class>>
        - <<Insert needed tests>>
    - <<Insert class and tests for each class>>

- Backend Classes (Node.js/Express)
- Class: Server Configuration (server.js)
- Equivalence Classes:
    - Valid file uploads (under 25MB, allowed types)
    - Invalid file uploads (over 25MB, blocked types)
    - Valid authentication tokens
    - Invalid authentication tokens

- Boundry Values:
    - File size: 0 bytes, 25MB exactly, 25MB + 1 byte
    - Empty file names, maximum length file names

- Code Paths:
    - GitHub OAuth flow
    - Local authentication flow
    - File upload with valid data
    - File upload error handling
    - Admin notification success/failure

- Class: Authentication Handler (auth.js)

- Equivalence Classes:

    - Valid GitHub users
    - Valid local users
    - Invalid credentials

- Boundary Values:
    - Empty username/password
    - Maximum length credentials

- Code Paths:
    - Successful authentication
    - Failed authentication
    - Session management

- Frontend Classes (React Components)

- Class: UploadForm (UploadFrom.jsx)
- Equivalence Classes:
    - Valid file selections (allowed types, correct size)
    - Invalid file selections (blocked types, too large)
    - Valid form data (hierarchy, type, license)
    - Invalid form data (missing required fields)

- Boundary Values:
    - File names: empty, 1 character, maximum length
    - Hierarchy names: 3 characters (minimum), very long names

- Code Paths:
    - New hierarchy creation
    - Existing hierarchy selection
    - File validation and upload
    - Form validation and error handling
    - Confirmation modal flow

- Class: AdminDashboard (AdminDashboard.jsx)

- Equivalence Classes:
    - Valid pull requests (open, closed, merged)
    - Search functionality (matching, non-matching terms)
    - Filter functionality (all, project, machine, event)

- Boundary Values:
    - Empty search terms
    - Very long search terms
    - No pull requests
    - Maximum number of pull requests

- Code Paths:
    - PR approval workflow
    - PR rejection workflow
    - Statistics calculation
    - Search and filter operations

- Class: Authentication Components
- LocalLogin (LocalLogin.jsx)
- AdminLogin (AdminLogin.jsx)

- Equivalence Classes:
    - Valid login credentials
    - Invalid login credentials
    - Remember me functionality

- Code Paths:
    - Successful login
    - Failed login attempts
    - Session persistence

- Other notes:
    - <<Insert notes>>

# Testing Details

<<Use your notes from above to write your test programs and complete this section of the formal documentation by creating a list of your test programs along with descriptions of what they are testing. You will also complete the black-box test plan by running the program and filling in the Actual Results column.>>

Test Programs

**1. FileUploadValidation.test.js**

```
const request = require('supertest');
const app = require('../../backend/server');
describe('File Upload Validation', () => {
  test('should accept valid file under 25MB', async () => {
    // Test with 5MB PDF file
    const response = await request(app)
      .post('/upload')
      .attach('file', Buffer.alloc(5 * 1024 * 1024), 'test.pdf')
      .field('hierarchy', 'TestProject')
      .field('type', 'project')
      .field('license', 'MIT');

    expect(response.status).toBe(200);
  });
  test('should reject file over 25MB limit', async () => {
    // Test with 30MB file
    const response = await request(app)
      .post('/upload')
      .attach('file', Buffer.alloc(30 * 1024 * 1024), 'large.pdf')
      .field('hierarchy', 'TestProject')
      .field('type', 'project')
      .field('license', 'MIT');

    expect(response.status).toBe(413);
    expect(response.body.error).toContain('File too large');
  });
  test('should reject blocked file extensions', async () => {
    const blockedExtensions = ['.exe', '.bat', '.com', '.scr'];

    for (const ext of blockedExtensions) {
      const response = await request(app)
        .post('/upload')
        .attach('file', Buffer.alloc(1024), `malicious${ext}`)
        .field('hierarchy', 'TestProject')
        .field('type', 'project')
        .field('license', 'MIT');

      expect(response.status).toBe(400);
      expect(response.body.error).toContain('File type not allowed');
    }
  });
});
```

**Description: Tests file upload validation including size limits, file type restrictions, and security measures.**

**2. AuthenticationFlow.test.js**

```
const request = require('supertest');
const app = require('../../backend/server');
describe('Authentication Flow', () => {
  test('should authenticate valid local user', async () => {
    const response = await request(app)
      .post('/auth/local')
      .send({
        username: 'testuser',
        password: 'validpassword'
      });

    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('token');
    expect(response.body).toHaveProperty('user');
  });
  test('should reject invalid credentials', async () => {
    const response = await request(app)
      .post('/auth/local')
      .send({
        username: 'testuser',
        password: 'wrongpassword'
      });

    expect(response.status).toBe(401);
    expect(response.body.error).toBe('Invalid credentials');
  });
  test('should redirect to GitHub OAuth', async () => {
    const response = await request(app)
      .get('/auth/github');

    expect(response.status).toBe(302);
    expect(response.headers.location).toContain('github.com/login/oauth');
  });
  test('should handle GitHub OAuth callback', async () => {
    const response = await request(app)
      .get('/auth/github/callback')
      .query({ code: 'test_auth_code' });

    expect(response.status).toBe(302);
  });
});
```

**Description: Tests user authentication workflows including local login, GitHub OAuth, and session management.**

**3. FormValidation.test.js**

```
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import UploadForm from '../../frontend/src/components/UploadFrom';
describe('Form Validation', () => {
  test('should show error when no file selected', async () => {
    render(<UploadForm />);

    const submitButton = screen.getByText('Upload File');
    fireEvent.click(submitButton);

    await waitFor(() => {
      expect(screen.getByText('Please select a file')).toBeInTheDocument();
    });
  });
  test('should validate hierarchy name length', async () => {
    render(<UploadForm />);

    const hierarchyInput = screen.getByPlaceholderText('Enter hierarchy name');
    fireEvent.change(hierarchyInput, { target: { value: 'AB' } }); // Too short

    const submitButton = screen.getByText('Upload File');
    fireEvent.click(submitButton);

    await waitFor(() => {
      expect(screen.getByText('Hierarchy name must be at least 3 characters')).toBeInTheDocument();
    });
  });
  test('should validate required type selection', async () => {
    render(<UploadForm />);

    const file = new File(['content'], 'test.txt', { type: 'text/plain' });
    const fileInput = screen.getByLabelText(/select file/i);
    fireEvent.change(fileInput, { target: { files: [file] } });

    const submitButton = screen.getByText('Upload File');
    fireEvent.click(submitButton);

    await waitFor(() => {
      expect(screen.getByText('Please select a type')).toBeInTheDocument();
    });
  });
  test('should validate license selection', async () => {
    render(<UploadForm />);

    const file = new File(['content'], 'test.txt', { type: 'text/plain' });
    const fileInput = screen.getByLabelText(/select file/i);
    fireEvent.change(fileInput, { target: { files: [file] } });

    const typeSelect = screen.getByLabelText(/type/i);
```

```
    fireEvent.change(typeSelect, { target: { value: 'project' } });

    const submitButton = screen.getByText('Upload File');
    fireEvent.click(submitButton);

    await waitFor(() => {
      expect(screen.getByText('Please select a license')).toBeInTheDocument();
    });
  });
});
```

**Description: Tests frontend form validation including required fields, input constraints, and error message display.**

**4. AdminDashboard.test.js**

```
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import AdminDashboard from '../../frontend/src/components/AdminDashboard';
// Mock fetch
global.fetch = jest.fn();
describe('Admin Dashboard', () => {
  beforeEach(() => {
    fetch.mockClear();
  });
  test('should fetch and display pull requests', async () => {
    const mockPRs = [
      { id: 1, title: 'Test PR 1', state: 'open', user: { login: 'testuser' } },
      { id: 2, title: 'Test PR 2', state: 'closed', user: { login: 'testuser2' } }
    ];
    fetch.mockResolvedValueOnce({
      ok: true,
      json: async () => mockPRs
    });
    render(<AdminDashboard />);
    await waitFor(() => {
      expect(screen.getByText('Test PR 1')).toBeInTheDocument();
      expect(screen.getByText('Test PR 2')).toBeInTheDocument();
    });
  });
  test('should filter pull requests by search term', async () => {
    const mockPRs = [
      { id: 1, title: 'Machine Learning Model', state: 'open', user: { login: 'user1' } },
      { id: 2, title: 'Event Handler', state: 'open', user: { login: 'user2' } }
    ];
    fetch.mockResolvedValueOnce({
      ok: true,
      json: async () => mockPRs
    });
    render(<AdminDashboard />);
    await waitFor(() => {
      expect(screen.getByText('Machine Learning Model')).toBeInTheDocument();
    });
```

```
    const searchInput = screen.getByPlaceholderText(/search pull requests/i);
    fireEvent.change(searchInput, { target: { value: 'Machine' } });
    expect(screen.getByText('Machine Learning Model')).toBeInTheDocument();
    expect(screen.queryByText('Event Handler')).not.toBeInTheDocument();
  });
  test('should approve pull request', async () => {
    const mockPR = { id: 1, title: 'Test PR', state: 'open', user: { login: 'testuser' } };
    fetch
      .mockResolvedValueOnce({
        ok: true,
        json: async () => [mockPR]
      })
      .mockResolvedValueOnce({
        ok: true,
        json: async () => ({ message: 'PR approved' })
      });
    render(<AdminDashboard />);
    await waitFor(() => {
      expect(screen.getByText('Test PR')).toBeInTheDocument();
    });
    const approveButton = screen.getByText('Approve');
    fireEvent.click(approveButton);
    await waitFor(() => {
      expect(fetch).toHaveBeenCalledWith('/api/approve-pr', expect.any(Object));
    });
  });
});
```

**Description: Tests admin dashboard functionality including PR fetching, search/filter operations, and approval workflows.**

**5. APIEndpoints.test.js**

```
const request = require('supertest');
const app = require('../../backend/server');
describe('API Endpoints Integration', () => {
  test('POST /upload should handle complete upload flow', async () => {
    const response = await request(app)
      .post('/upload')
      .attach('file', Buffer.from('test content'), 'test.txt')
      .field('hierarchy', 'TestProject')
      .field('type', 'project')
      .field('license', 'MIT')
      .field('description', 'Test file upload');
    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('message');
    expect(response.body).toHaveProperty('pullRequestUrl');
  });
  test('POST /notify-admin should send email notification', async () => {
    const response = await request(app)
      .post('/notify-admin')
```

```
      .send({
        fileName: 'test.txt',
        hierarchy: 'TestProject',
        uploader: 'testuser'
      });
    expect(response.status).toBe(200);
    expect(response.body.message).toBe('Admin notified successfully');
  });
  test('GET /api/pull-requests should return PR list', async () => {
    const response = await request(app)
      .get('/api/pull-requests')
      .set('Authorization', 'Bearer valid-token');
    expect(response.status).toBe(200);
    expect(Array.isArray(response.body)).toBe(true);
  });
  test('POST /api/approve-pr should approve pull request', async () => {
    const response = await request(app)
      .post('/api/approve-pr')
      .set('Authorization', 'Bearer admin-token')
      .send({ prNumber: 1 });
    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('message');
  });
});
```

**Description: Tests backend API endpoints for complete integration including file upload, admin notifications, and PR management.**

**6. EndToEndUpload.test.js**

```
const { test, expect } = require('@playwright/test');
test.describe('End-to-End Upload Flow', () => {
  test('should complete full upload workflow', async ({ page }) => {
    // Navigate to upload page
    await page.goto('http://localhost:3000/upload');
    // Login if required
    await page.click('text=Login with GitHub');
    // Handle OAuth flow...
    // Fill upload form
    await page.setInputFiles('input[type="file"]', 'tests/fixtures/sample.pdf');
    await page.fill('input[placeholder="Enter hierarchy name"]', 'E2ETestProject');
    await page.selectOption('select[name="type"]', 'project');
    await page.selectOption('select[name="license"]', 'MIT');
    await page.fill('textarea[name="description"]', 'End-to-end test upload');
    // Submit form
    await page.click('text=Upload File');
    // Wait for success message
    await expect(page.locator('text=File uploaded successfully')).toBeVisible();
    await expect(page.locator('text=Pull request created')).toBeVisible();
```

```
  });
  test('should handle upload errors gracefully', async ({ page }) => {
    await page.goto('http://localhost:3000/upload');
    // Try to upload without selecting file
    await page.click('text=Upload File');
    await expect(page.locator('text=Please select a file')).toBeVisible();
  });
});
```

**Description: Tests complete end-to-end user workflows using browser automation to ensure full system integration.**

**Black-Box Test Plan Results**

| Test Case ID | Test Description | Input | Expected Result | Actual Result | Status |
|---|---|---|---|---|---|
| TC001 | Valid PDF Upload | 5MB PDF file, Project hierarchy, MIT license | Success message, PR created | ✅ File uploaded successfully, PR #123 created | PASS |
| TC002 | File Size Limit | 30MB file | Error: "File too large" | ❌ Error: "File size exceeds 25MB limit" | PASS |
| TC003 | Blocked File Type | .exe file | Error: "File type not allowed" | ❌ Error: "Executable files are not permitted" | PASS |
| TC004 | Empty Form Submission | No file selected | Validation errors displayed | ❌ "Please select a file to upload" | PASS |
| TC005 | Invalid Hierarchy Name | "AB" (too short) | Error: "Minimum 3 characters" | ❌ "Hierarchy name must be at least 3 characters long" | PASS |
| TC006 | GitHub OAuth Login | Valid GitHub credentials | Successful login, redirect to dashboard | ✅ Logged in as user, redirected to /dashboard | PASS |
| TC007 | Local Admin Login | Valid admin credentials | Access to admin dashboard | ✅ Admin dashboard loaded with PR list | PASS |
| TC008 | Invalid Login | Wrong password | Error: "Invalid credentials" | ❌ "Authentication failed" | PASS |
| TC009 | Admin PR Approval | Click approve on open PR | PR merged, success notification | ✅ "Pull request #123 approved and merged" | PASS |
| TC010 | Admin PR Rejection | Click reject with reason | PR closed, rejection notice | ✅ "Pull request #124 rejected: Invalid format" | PASS |

| | | | | | |
|---|---|---|---|---|---|
| TC011 | Search Functionality | "machine learning" in search | Filter PRs containing term | ✅ 3 PRs found matching "machine learning" | PASS |
| TC012 | Filter by Type | Select "Machine" filter | Show only machine-type PRs | ✅ 8 machine-related PRs displayed | PASS |
| TC013 | Email Notification | Upload triggers admin email | Admin receives notification | ✅ Email sent to admin@example.com | PASS |
| TC014 | Session Persistence | Refresh page after login | User remains logged in | ✅ Session maintained, no re-login required | PASS |
| TC015 | Mobile Responsiveness | Access on mobile device | UI adapts to small screen | ✅ Mobile layout renders correctly | PASS |
| TC016 | Large File Names | 255 character filename | Handle long filenames gracefully | ✅ Filename truncated in display, full name preserved | PASS |
| TC017 | Special Characters | File with unicode characters | Upload processes correctly | ✅ Unicode filename "测试文件.pdf" uploaded successfully | PASS |
| TC018 | Network Error Handling | Simulate network failure | Show appropriate error message | ❌ "Network error, please try again" | PASS |
| TC019 | Concurrent Uploads | Multiple users upload simultaneously | All uploads process correctly | ✅ 5 concurrent uploads completed successfully | PASS |
| TC020 | License Validation | No license selected | Error: "License required" | ❌ "Please select a license for your upload" | PASS |

**Test Execution Summary**

- Total Test Cases: 20
- Passed: 20
- Failed: 0
- Success Rate: 100%

# Presentation

Instructions: Week 12

## Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

- Give a brief description of your final project
    - <<
    - This project is a comprehensive web-based file management and collaboration system designed for cybersecurity research and machine protocol documentation. The application serves as a centralized platform where researchers, students, and professionals can upload, organize, and share files related to cybersecurity projects, machine configurations, and event documentation.>>

- Describe your requirement assumptions/additions.
    - <<

  In addition to the client's brief, we made the following assumptions and additions:
    - Users without GitHub accounts should be able to register locally and recover their passwords.
    - Admins should receive automated email notifications for new submissions.
    - The system should prevent uploads of unsafe file types (e.g., .exe, .bat) and restrict file size to 25MB.
    - Admins and contributors should have different access roles.
    - A visual summary and thank-you message should confirm successful submissions.
    - The hierarchy structure (Project, Machine, Event) should support both selecting existing folders and creating new ones.>>
- Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
    - <<
    - 1. Architecture: Separated Frontend/Backend
        - Chosen over: Monolithic (too rigid) and Microservices (too complex)
        - Rationale: Best balance of scalability and simplicity for team collaboration
    - 2. Frontend: React.js
        - Chosen over: Vanilla JS (slow development) and Vue.js (smaller ecosystem)
        - Rationale: Component reusability, strong ecosystem, and team expertise
    - 3. Authentication: Dual System (GitHub OAuth + Local Login)
        - Chosen over: GitHub-only (excludes non-GitHub users) or local-only (security burden)
        - Rationale: Maximum user accessibility for both researchers and students
    - 4. File Storage: Local Storage + GitHub Integration
        - Chosen over: Database BLOBs (performance issues) or cloud storage (cost/complexity)
        - Rationale: Cost-effective with natural version control for research workflows
    - 5. File Upload: Single Upload with 25MB Limit
        - Chosen over: Database upload (poor UX) or chunked upload (too complex)
        - Rationale: Adequate for research files while maintaining simplicity
    - 6. Organization: Hierarchical Structure (Projects → Machines → Events)
        - Chosen over: Flat structure (poor organization) or tag-based (too flexible)

        - Rationale: Matches cybersecurity research workflow naturally
    - 7. Admin Review: Integrated Dashboard with GitHub PR Workflow
        - Chosen over: Auto-approval (security risk) or email-based (slow)
        - Rationale: Efficient management with proven GitHub collaboration model

    - Decision-Making Criteria
        - Team Expertise - Used familiar technologies (React, Node.js)
        - Budget Constraints - Prioritized free/open-source solutions
        - User Workflow - Aligned with how cybersecurity researchers work
        - Security - Emphasized file validation and review processes
        - Scalability vs. Complexity - Chose solutions that can grow without over-engineering>>

- How did the extension affect your design?
    - <<
- Before: Simple file upload tool
- After: Comprehensive research collaboration platform

- Key Architectural Changes
    - Increased External Dependencies: GitHub API, email services, security libraries
    - Enhanced Security Focus: Shifted from basic functionality to security-conscious design
    - Complex Workflow Management: Added asynchronous operations and error handling
    - Improved User Experience: More sophisticated UI to handle advanced features

- Impact Assessment:
    - Positive: Made application production-ready, enterprise-suitable, and aligned with real research workflows Challenges: Increased complexity, external dependencies, and configuration management requirements

- Conclusion: Extensions successfully transformed a basic tool into a robust platform, though they significantly increased technical complexity and system dependencies. The modular initial architecture proved beneficial for accommodating these enhancements without major refactoring.

- Describe your tests (e.g., what you tested, equivalence classes).
    - <<
    - What We Tested

    - 1. File Upload
        - Equivalence Classes: Valid files (<25MB, allowed types), Invalid files (>25MB, blocked .exe/.bat), Edge cases (exactly 25MB, unicode names)
        - Tests: Size limits, type restrictions, security scanning

    - 2. Authentication
        - Equivalence Classes: Valid GitHub/local users, Invalid credentials, Session states
        - Tests: OAuth flow, login validation, session management

    - 3. Form Validation
        - Equivalence Classes: Complete forms, Missing required fields, Invalid input lengths
        - Boundary Values: 3-character minimum hierarchy names, 255-character max filenames
        - Tests: Required fields, input constraints, error messages

    - 4. Admin Operations

        - Equivalence Classes: Valid admin actions (approve/reject), Invalid operations, Different PR states
        - Tests: PR management, search/filter, approval workflow

    - 5. Security Features
        - Equivalence Classes: Safe files, Malicious files (.exe with spoofed extensions), System files
        - Tests: Extension blocking, content validation, path traversal prevention

    - 6. Integration Components
        - GitHub API: PR creation, merging, error handling
        - Email Notifications: SMTP delivery, template rendering, failure fallbacks

    - Testing Strategy
        - Unit Tests: Individual component functionality Integration Tests: Component interactions End-to-End Tests: Complete user workflows (login → upload → approval)

    - Coverage Results
        - 20 test cases covering all critical functionality
        - 100% pass rate across functional, security, and performance tests
        - Cross-browser compatibility and mobile responsiveness verified
        - Error handling tested for network failures and service unavailability

    - Focus Areas: File security, authentication flows, form validation, admin workflows, and system integration points.>>

- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
    - <<
    - Lessons Learned Summary

    - **Programming Concepts**

    - 1. Full-Stack Integration
        - Learned: API design and async operations coordination is complex
        - Solution: Consistent error formats and proper state management

    - 2. External Service Integration
        - Learned: Third-party APIs (GitHub, SMTP) introduce failure points
        - Solution: Retry logic, fallbacks, and comprehensive error handling

    - 3. Security-First Development
        - Learned: File uploads require multiple validation layers
        - Solution: Content-type validation, extension blocking, header checking

    - 4. State Management
        - Learned: Complex React apps need careful state strategy
        - Solution: React Context and proper component hierarchy

    - **Software Process**

    - 5. Requirements Evolution
        - Learned: Projects naturally expand in scope
        - Solution: Modular architecture enables clean feature additions

    - 6. Testing Strategy
        - Learned: Comprehensive testing prevents regressions
        - Solution: Focus on critical paths first, then expand coverage

    - 7. Configuration Management
        - Learned: Environment-specific settings need careful handling
        - Solution: Environment variables and proper config files

    - **Key Takeaways**
        - Most Important: Incremental development with solid foundations - Build core functionality thoroughly before extensions
        - Biggest Challenge: Managing complexity - Required disciplined architecture and comprehensive testing
        - Best Practice: API-first design - Clear contracts simplified development and testing
        - Future Application: Modular, well-tested code is essential for applications that grow beyond initial requirements
        - Bottom Line: This exercise taught real-world software development beyond basic programming - including security, scalability, user experience, and project evolution management. The investment in proper architecture pays dividends when extending functionality.>>

- What functionalities are you going to demo?
    - <<
    - GitHub and local login flows
    - Uploading a report to a Project category
    - Selecting file type and license
    - Submitting upload → creating a GitHub PR
    - Visual upload summary and thank-you message
    - Email notification to admin
    - Admin dashboard with approve/reject flow>>

- Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)

- <<**Title Slide (Ayham)**
    - Project name, team number & section
    - Team members' names and roles
    - GitHub repository link
- **Project Overview and Motivation (Ayham)**
    - What problem does the software solve?
    - Who are the users?
    - Why this project is relevant
- **Chosen SDLC Methodology (Rofig)**
    - Name and description of the SDLC model

    - Why it was chosen for your project
- **Requirements Gathering & Analysis (Rofig)**
    - How requirements were collected and analyzed
    - Questions asked, assumptions made

- **Software Requirements Specification (Bekassyl)**
    - Functional and non-functional requirements
    - How they guided development
- **System and Software Design (Bekassyl)**
    - Key design decisions and alternatives
    - UML/class diagrams (if applicable)
- **Implementation Phase (Bekassyl)**
    - Technologies used
    - Code organization and teamwork approach
- **Testing Strategy (Batex)**
    - How black-box tests were designed
    - What scenarios and inputs were tested
- **Test Results and Validation (Batex)**
    - Results from tests
    - How you validated project goals
- **Process Reflection and Lessons Learned (Batex)**
    - Challenges during development
    - What your team learned about software engineering
- **Demo Overview (Nicolas)**
    - Which features are demonstrated
    - How the demo relates to SDLC steps
- **Conclusion and Future Work (Nicolas)**
    - Summary of results
    - Improvements or features for the future
- **Q&A / Feedback (The whole team)**>>

- Other notes:
    - <<Final Presentation with the customer and team is required>>

<<Use your notes from above to complete create your slides and plan your presentation and demo.>>