Simple React Apps

Assumptions

1. Use Strapi for backend
2. Assume t2.small instance or larger
3. Assume local file store for file storage
4. Assume Material (?) UI Kit
5. Assume Tailwind
6. Assume React
7. We plan on switching from a Lambda based approach to AWS ECS/Fargate

Requirements for Task Management Web Application

1. User Authentication
- 1.1 Clients must be able to log in using secure credentials.
- 1.2 Users can be of two roles: clients or staff

2. Task Creation and Submission
- 2.1 Clients can create a new task using a form.
  - 2.1.1 The form must include fields for task description, file attachment, and a field for providing file locations (Box, Dropbox, Google Drive, etc.).
  - 2.1.2 The form should validate that all required fields are completed before submission.
- 2.2 Upon task submission, an email notification is sent to the designated staff member.

3. Task Management
- 3.1 Task states include: Active, Paused, Blocked, Finished, and Canceled.
- 3.2 Clients can view a summary page listing all their tasks in a tabular format.
  - 3.2.1 The table must display the task name, task state, and other relevant details.
  - 3.2.2 Clients can sort and filter tasks based on status and other criteria.
- 3.3 Clients can click on a task in the list to view detailed information about the task.
  - 3.3.1 Detailed information includes task description, file attachments, comments, and task history.
  - 3.3.2 The task details page must include a tab to view all previous revisions (task versions).
  - 3.3.3 Clients can edit the task details. All edits create a new task version.
  - 3.3.4 Clients can add comments in a blog-style comment section on the task details page.
  - 3.3 The task summary page should have a search box

4. Staff Task Management
- 4.1 Staff members have a dashboard that displays all tasks across all clients.
  - 4.1.1 The dashboard allows sorting and filtering based on client, task state, and other criteria.
- 4.2 Staff members can update the task status.
  - 4.2.1 Setting the task status to "Blocked" allows staff to request additional information or clarification.
  - 4.2.2 An alert is sent to the client when a task status is set to "Blocked."
  - 4.3 The task summary page should have a search box

5. Notifications
- 5.1 Email notifications are sent to the client when the task status changes.
- 5.2 An alert system notifies clients within the app of any actions needed, such as when a task is blocked.

6. Permissions
- 6.1 Clients can only view and manage their own tasks.
- 6.2 Staff members can view and manage tasks for all clients.

7. Security
- 7.1 All file uploads and downloads must be secure.
- 7.2 User sessions must be securely managed with appropriate session timeouts and protections against unauthorized access.
This structure should provide a clear, concise, and unambiguous framework for developing the web application.
- 7.3 The file urls should be access controlled so that only a staff member or the client who submitted the file can be accessed by them

Significant pages

1. Login Page
   - Purpose: Allows users to login using secure credentials.
   - Users: Both clients and staff.
   - Features:
     - Role-based redirection (clients to their dashboard, staff to their dashboard).
     - Forgot password and reset functionality.

2. Task Creation Page
   - Purpose: Allows clients to create a new task.
   - Users: Clients.
   - Features:
     - Form fields for task description, file attachments, and file location URLs.
     - Validation of required fields before submission.
     - Option to attach files from local storage or provide links to cloud storage (Box, Dropbox, Google Drive).
     - Submit button to create the task and trigger an email notification to the staff.

3. Client Dashboard (Task Summary Page)
   - Purpose: Displays a summary of all tasks submitted by the client.
   - Users: Clients.
   - Features:
     - Tabular view of all tasks with columns for task name, task state, creation date, and other relevant details.
     - Sort and filter options based on task state and other criteria.
     - Search box to quickly locate specific tasks.
     - Links to view detailed information about each task.

4. Task Details Page
   - Purpose: Provides detailed information about a specific task.
   - Users: Clients and Staff.
   - Features:
     - Task description, attached files, comments, and task history.
     - Tab for viewing all previous revisions (task versions).
     - Comment section for clients and staff to communicate.
     - Option for clients to edit task details, creating a new version of the task.
     - For staff, options to update task status (e.g., Active, Blocked, Finished).

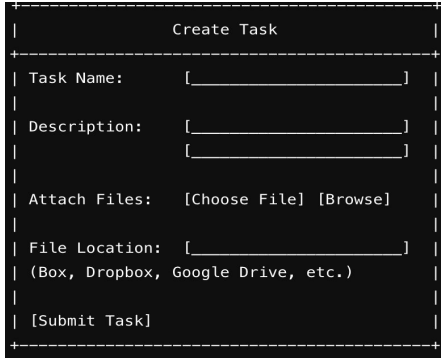
5. Staff Dashboard (Task Management Page)
   - Purpose: Displays all tasks across all clients for staff members to manage.
   - Users: Staff.
   - Features:
     - Tabular view of all tasks with columns for client name, task name, task state, creation date, and other relevant details.
     - Sort and filter options based on client, task state, and other criteria.
     - Search box to quickly locate specific tasks.
     - Links to view detailed information about each task.
     - Ability to update task status, set tasks to "Blocked," and request additional information from clients.

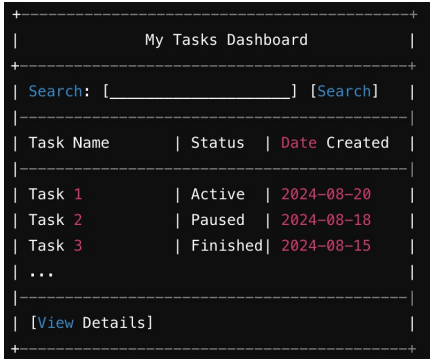
6. Notifications Page (or Section)
   - Purpose: Displays alerts and notifications related to task updates and status changes.
   - Users: Clients and Staff.
   - Features:
     - List of notifications with links to relevant tasks.
     - Mark notifications as read or unread.

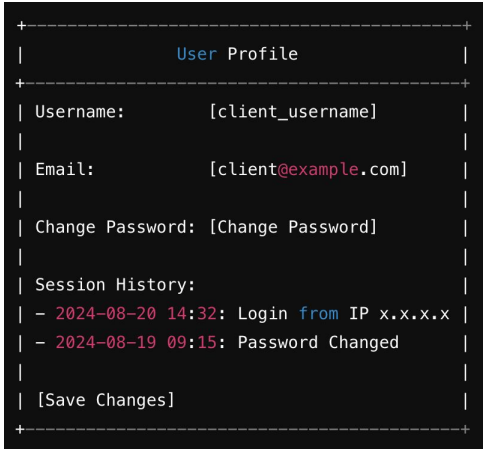
7. File Access Control Page (Admin Page)
   - Purpose: Manages access control for files submitted or linked by clients.
   - Users: Admins or Staff with elevated privileges.
   - Features:
     - View and manage file permissions to ensure only the submitting client or relevant staff can access files.
     - Audit trail for file access and modifications.

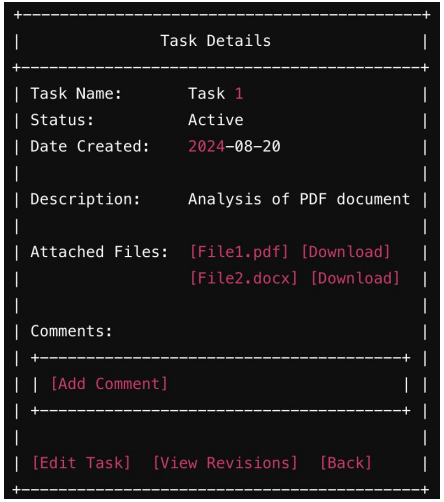
8. User Profile Page
   - Purpose: Allows users to manage their profile information, including passwords.
   - Users: Clients and Staff.
   - Features:
     - Edit personal information, update passwords, and manage security settings.
     - View session history and manage active sessions for security purposes.

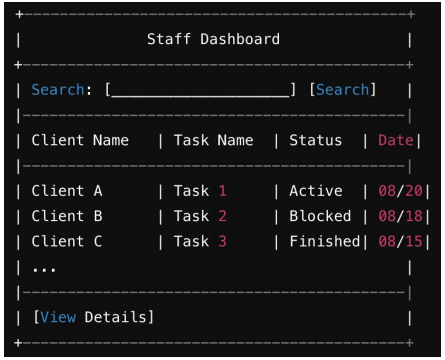

Login
Username: [_____]
Password: [_____]
[ ] Remember Me
[Login]
Forgot your password? [Reset Password]

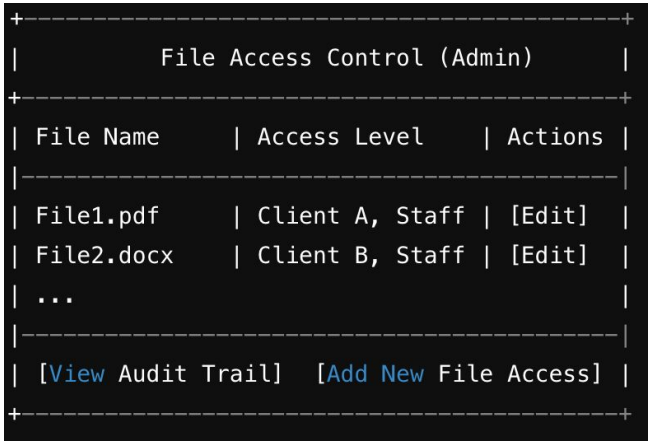

Create Task
Task Name: [_____]
Description: [_____]
[_____]
Attach Files: [Choose File] [Browse]
File Location: [_____]
(Box, Dropbox, Google Drive, etc.)
[Submit Task]

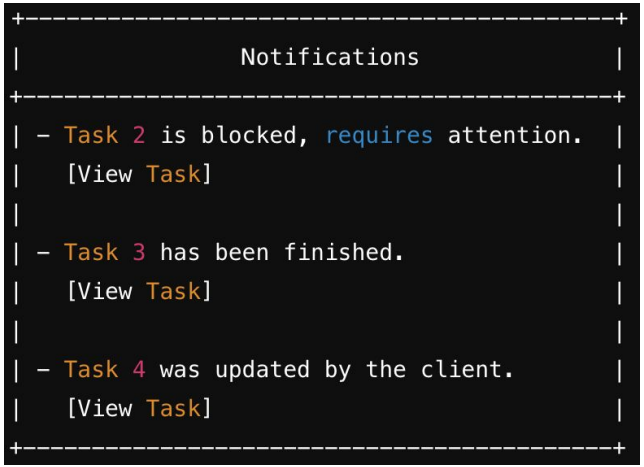

My Tasks Dashboard
Search: [_____] [Search]
Task Name | Status | Date Created
Task 1 | Active | 2024-08-20
Task 2 | Paused | 2024-08-18
Task 3 | Finished | 2024-08-15
...
[View Details]

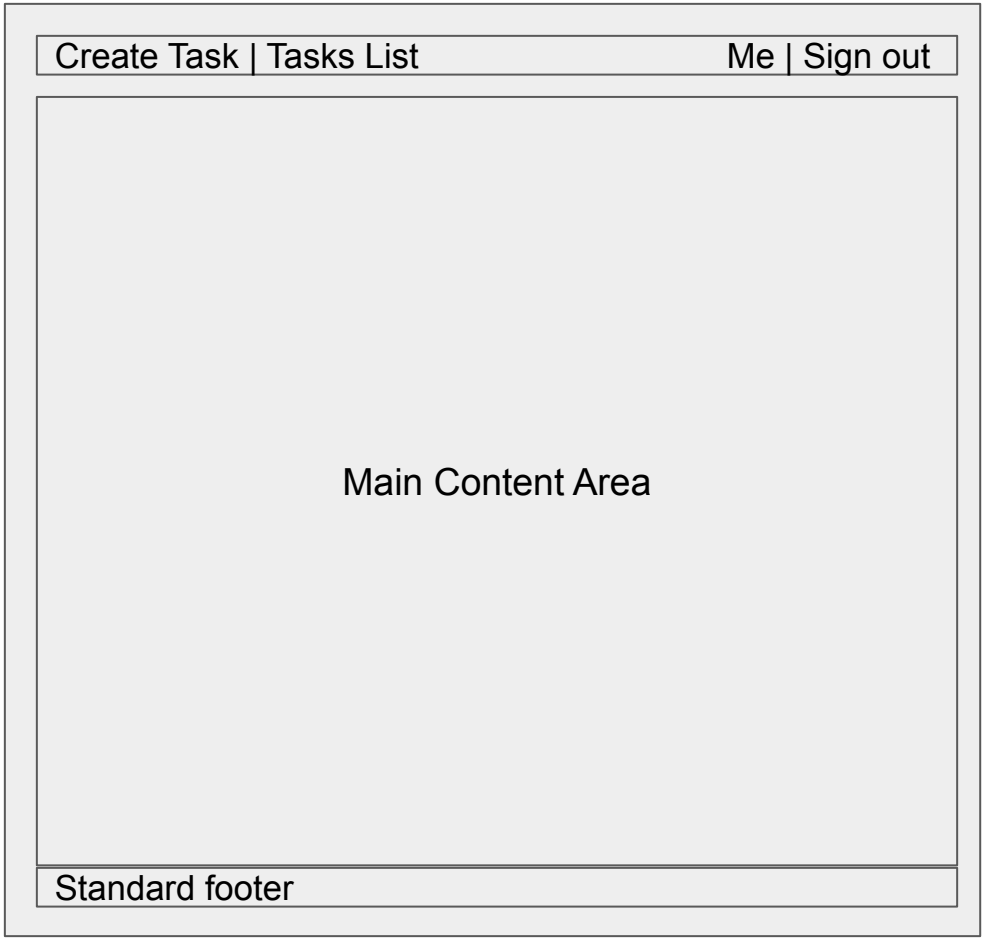

User Profile
Username: [client_username]
Email: [client@example.com]
Change Password: [Change Password]
Session History:
- 2024-08-20 14:32: Login from IP x.x.x.x
- 2024-08-19 09:15: Password Changed
[Save Changes]


Task Details
Task Name: Task 1
Status: Active
Date Created: 2024-08-20
Description: Analysis of PDF document
Attached Files: [File1.pdf] [Download]
[File2.docx] [Download]
Comments:
[Add Comment]
[Edit Task] [View Revisions] [Back]


Staff Dashboard
Search: [_____] [Search]
Client Name | Task Name | Status | Date
Client A | Task 1 | Active | 08/20
Client B | Task 2 | Blocked | 08/18
Client C | Task 3 | Finished | 08/15
...
[View Details]


File Access Control (Admin)
File Name
Access Level
Actions
File1.pdf
Client A, Staff
[Edit]
File2.docx
Client B, Staff
[Edit]
...
[View Audit Trail]
[Add New File Access]


Notifications
- Task 2 is blocked, requires attention.
[View Task]
- Task 3 has been finished.
[View Task]
- Task 4 was updated by the client.
[View Task]


Create Task | Tasks List
Me | Sign out
Main Content Area
Standard footer

User Authentication / Strapi
Story 1: Implement User Roles and Authentication
- Task 1.1: Set up user roles in Strapi (Client, Staff).
- Task 1.2: Configure authentication providers (e.g., email, social logins if needed).
- Task 1.3: Implement secure password storage using bcrypt or similar hashing algorithm.
- Task 1.4: Create API endpoints for user login, logout, and session management.
- Task 1.5: Set up role-based access control (RBAC) to restrict access to resources based on user roles.

Task Creation and Submission / Strapi
Story 2: Develop Task Model
- Task 2.1: Define the task model in Strapi with fields: task name, description, status, attached files, file locations, client reference, and staff reference.
- Task 2.2: Implement validation rules for required fields (e.g., task name, description).
- Task 2.3: Configure relations between the task model and user model (Client and Staff).
- Task 2.4: Create API endpoints for task creation, update, and retrieval.
- Task 2.5: Integrate file upload handling for local storage or external URLs (Box, Dropbox, Google Drive).

Client Dashboard (Task Summary Page) / Strapi
Story 3: Implement Task Summary API
- Task 3.1: Create an API endpoint to fetch a list of tasks for a specific client, filtered by user ID.
- Task 3.2: Implement sorting and filtering mechanisms in the API to handle task states and creation date.
- Task 3.3: Add a search capability in the API to allow clients to search for specific tasks by name or description.
- Task 3.4: Ensure the API returns data in a format suitable for display in a tabular form on the frontend.

Task Details Page / Strapi
Story 4: Develop Task Details API
- Task 4.1: Create an API endpoint to fetch detailed information about a specific task by its ID.
- Task 4.2: Implement functionality to retrieve task revisions and include them in the task details response.
- Task 4.3: Set up a comments model linked to tasks, allowing clients and staff to add and retrieve comments.
- Task 4.4: Enable task updates by clients, ensuring that a new revision is created each time.

Staff Dashboard (Task Management Page) / Strapi
Story 5: Create Staff Task Management API
- Task 5.1: Create an API endpoint to fetch a list of all tasks across all clients for staff members.
- Task 5.2: Implement sorting and filtering capabilities based on client, task state, and other criteria.
- Task 5.3: Enable staff to update task status through the API, including setting tasks to "Blocked" and requesting additional information.
- Task 5.4: Ensure that when the status is updated, the appropriate notifications are triggered for the client.

Notifications / Strapi
Story 6: Set Up Notification System
- Task 6.1: Implement an email notification system in Strapi using email providers or plugins.
- Task 6.2: Configure the system to send emails when a task is created, updated, or blocked.
- Task 6.3: Develop an in-app notification system that stores and displays alerts to clients and staff.
- Task 6.4: Create an API endpoint to fetch and manage notifications for clients and staff.

File Access Control Page (Admin Page) / Strapi
Story 7: Implement File Access Control
- Task 7.1: Set up access control policies in Strapi to ensure that only the submitting client or relevant staff members can access task-related files.
- Task 7.2: Develop an API endpoint for admins to manage file access permissions.
- Task 7.3: Create an audit trail model that logs file access and modifications.
- Task 7.4: Integrate access control checks into file retrieval API endpoints.

User Profile Page / Strapi
Story 8: Implement User Profile Management
- Task 8.1: Create an API endpoint to retrieve and update user profile information, including email and password.
- Task 8.2: Implement functionality to manage and update user passwords securely.
- Task 8.3: Develop session history tracking to log and display user activity.
- Task 8.4: Integrate user profile management with Strapi’s authentication and authorization system.

Security / Strapi
Story 9: Secure File Handling and User Sessions
- Task 9.1: Ensure secure file uploads and downloads using Strapi's file upload plugin or custom middleware.
- Task 9.2: Implement secure session management, including session timeouts and protection against unauthorized access.
- Task 9.3: Configure Strapi's security settings to enforce HTTPS, strong passwords, and other best practices.
- Task 9.4: Audit and test API endpoints for vulnerabilities and ensure compliance with security standards.

User Authentication / Strapi
Story 10: Implement User Authentication UI
- Task 10.1: Create a login page with fields for username and password.
- Task 10.2: Implement form validation for required fields and incorrect login attempts.
- Task 10.3: Add a "Forgot Password" link that redirects to a password reset page.
- Task 10.4: Implement role-based redirection after login (clients to client dashboard, staff to staff dashboard).
- Task 10.5: Create a logout button and functionality to clear the session and redirect to the login page.

Task Creation and Submission / React
Story 11: Develop Task Creation Form
- Task 11.1: Design and implement a task creation form with fields for task name, description, file attachments, and file location URLs.
- Task 11.2: Implement form validation to ensure required fields are completed before submission.
- Task 11.3: Add file upload functionality that integrates with the Strapi backend.
- Task 11.4: Include options for clients to provide links to cloud storage (Box, Dropbox, Google Drive).
- Task 11.5: Implement a submit button that triggers the task creation API and shows confirmation upon success.

Client Dashboard (Task Summary Page) / React
Story 12: Build Client Dashboard UI
- Task 12.1: Design a responsive table layout to display the list of tasks, including task name, state, and creation date.
- Task 12.2: Implement sorting and filtering functionality based on task state and other criteria.
- Task 12.3: Add a search box to quickly locate specific tasks by name or description.
- Task 12.4: Integrate the dashboard with the Strapi API to fetch and display the client's tasks.
- Task 12.5: Implement pagination for the task list if needed.

Task Creation and Submission / React

Story 11: Develop Task Creation Form

- Task 11.1: Design and implement a task creation form with fields for task name, description, file attachments, and file location URLs.
- Task 11.2: Implement form validation to ensure required fields are completed before submission.
- Task 11.3: Add file upload functionality that integrates with the Strapi backend.
- Task 11.4: Include options for clients to provide links to cloud storage (Box, Dropbox, Google Drive).
- Task 11.5: Implement a submit button that triggers the task creation API and shows confirmation upon success.

Client Dashboard (Task Summary Page) / React

Story 12: Build Client Dashboard UI

- Task 12.1: Design a responsive table layout to display the list of tasks, including task name, state, and creation date.
- Task 12.2: Implement sorting and filtering functionality based on task state and other criteria.
- Task 12.3: Add a search box to quickly locate specific tasks by name or description.
- Task 12.4: Integrate the dashboard with the Strapi API to fetch and display the client's tasks.
- Task 12.5: Implement pagination for the task list if needed.

Task Details Page / React

Story 13: Develop Task Details Page UI

- Task 13.1: Design the task details page to display task name, description, status, attached files, and comments.
- Task 13.2: Add a tabbed interface to view task revisions and comments.
- Task 13.3: Implement functionality to display and download attached files.
- Task 13.4: Allow clients to add comments and edit task details, triggering the appropriate API calls.
- Task 13.5: Display a history of task revisions with links to view previous versions.

Staff Dashboard (Task Management Page) / React

Story 14: Create Staff Dashboard UI

- Task 14.1: Design a dashboard that displays all tasks across all clients, including client name, task name, state, and creation date.
- Task 14.2: Implement sorting and filtering based on client, task state, and other criteria.
- Task 14.3: Add a search box for staff to quickly locate specific tasks.
- Task 14.4: Integrate the dashboard with the Strapi API to fetch and display all tasks for staff members.
- Task 14.5: Implement functionality for staff to update task status, set tasks to "Blocked," and request additional information from clients.

Notifications / React

Story 15: Build Notifications UI

- Task 15.1: Design a notifications section or page that lists alerts related to task updates and status changes.
- Task 15.2: Integrate with the Strapi API to fetch notifications for both clients and staff.
- Task 15.3: Implement functionality to mark notifications as read or unread.
- Task 15.4: Ensure that notifications are dynamically updated in real-time or at regular intervals.

File Access Control Page (Admin Page) / React

Story 16: Develop File Access Control UI

- Task 16.1: Design an admin interface to manage file access control, showing a list of files with their associated access levels.
- Task 16.2: Integrate the page with the Strapi API to fetch file access information.
- Task 16.3: Implement functionality to edit access levels and save changes via the API.
- Task 16.4: Add an option to view the audit trail for each file.

User Profile Page / React

Story 17: Create User Profile Management UI

- Task 17.1: Design a user profile page that displays and allows editing of user information (e.g., username, email).
- Task 17.2: Implement a form for users to change their password.
- Task 17.3: Integrate the profile page with the Strapi API to update user details.
- Task 17.4: Display session history, allowing users to view and manage active sessions.
- Task 17.5: Implement save and cancel buttons for profile changes.

Security / React

Story 18: Implement Frontend Security Measures

- Task 18.1: Ensure secure communication between the React frontend and Strapi backend (e.g., using HTTPS).
- Task 18.2: Implement input validation and sanitation to protect against XSS and other attacks.
- Task 18.3: Manage user sessions securely, including handling session timeouts and unauthorized access.
- Task 18.4: Test all frontend API interactions for security vulnerabilities, including access control on the client side.

Developer Guidance for Executing Stories Independently

Use the following as engineering guidelines – not all will be applicable for such a small project – the key is to get people moving independently except when they are integrating at the end

1. Use Mock APIs and Data
- Frontend: Utilize tools like `json-server` or `Mirage.js` to mock API responses. Create dummy endpoints that simulate the Strapi backend behavior.
- Backend: Develop using mock data structures. Implement dummy controllers in Strapi that return static data or simple logic.

2. Isolate Components
- Frontend: Build each page or component in isolation using Storybook or a similar tool. Test components individually with mock props and state.
- Backend: Develop and test each API endpoint independently. Use tools like Postman to simulate frontend requests.

3. Stubs and Mocks for Testing
- Write unit tests with stubs and mocks for dependencies. This ensures that components and APIs can be tested in isolation without waiting for full integration.

4. Feature Flags

- Use feature flags or environment variables to toggle between using the real API and mock data. This allows developers to switch contexts easily.

5. Independent Branching

- Work on separate branches for frontend and backend stories. Merge only when both sides are complete and tested.

6. Continuous Integration with Mocks

- Set up CI pipelines that run tests with mock objects and data. Ensure that both frontend and backend pass tests independently before integration.

7. API Contracts

- Define API contracts (e.g., using Swagger/OpenAPI) early. Use these contracts to guide both frontend and backend development, ensuring alignment without direct dependency.

Break Break Break Break

Objective:

Determine a method and corresponding technology stack that allows us to reliably, consistently, and rapidly deliver secure cloud-based web applications using React, Strapi, and AWS services.

Hypothesis

Using a combination of React, Strapi, and AWS cloud services achieves this goal.

App

- Dev ... simple ... ne hello world app – the test the hypothesis
- Spli ... cts of the hy ... is into sub-hypotheses and test across the team
- Imp ... and test ea ... a inde ... gning ea ... a ...
- Coll ... ta and ana ... ults o ...
- Itera ... findings.
- Inte ... define workflows t ... ement ... inception to ... yment.

Priori ... -Hypotheses Testing A ... Red urge ... lue ge ... hen we can) ... keouts lik ... repre ... w risk w ... derstoo ... s, skip for now

0. Pr ... older structure and Git st ...
1. N ... equirements ... odifi ...
2. New Requirements for Data Model Changes
3. Customization of Graphic Style (CSS Tweaking)
~~4. Backup~~
5. Environment Configuration Management
~~6. Restore from Backup~~
7. Notification of Security Patch
8. Security Patch for React
9. Security Patch for Strapi
10. Security Patch for Custom Logic
11. Integration with Third-party Services
~~12. Updating Security Certificate~~
13. Dependency Management
14. End-of-Life for Components
15. Performance Optimization
16. Compliance and Auditing
17. Monitoring and Logging
~~18. Disaster Recovery Planning~~
19. Deployment Pipeline Maintenance

Note: Previous reliance on ToolJet and AppSmith were infeasible; work to find failure points early.

Partial

Viability Thread

Hello World App
Login + Menu with two items: hello world page, hello world form. On submit form change test on hello world page on reload of hello world page.
GitHub

Front end in React

Backend in Strapi

Deploy Option 1:
Deploy using boto3 + **Lambda** + ELB + RDS + S3 + VPC + SSL Certificate

Deploy Option 2:
Deploy using boto3 + **EKS** + ELB + RDS + S3 + VPC + SSL Certificate

Deploy Option 3:
Deploy using boto3 + **ECS** + ELB + RDS + S3 + VPC + SSL Certificate

New requirement: change hello world page to hello world posts wall

Q: How do we folder structure the project

Q: What is the step by step workflow to go from requirements to Deployed app

Use the simplest hello world app to demonstrate and understand the deployment of the apps to the cloud

REQUIREMENTS ON A LATER SLIDE

Requirements for "Hello World" App

1. User Authentication:
   - Implement a simple login mechanism using a user ID and password.
   - No page should be accessible without successful login.
   - Authentication should maintain user sessions securely.

2. App Layout:
   - Left-Hand Margin: Contains a navigation menu or placeholder content.
   - Header: Fixed at the top, includes the app's name/logo.
   - Footer: Fixed at the bottom, includes static content such as a copyright notice.
   - Main Content Area: Central area for displaying the page content.
   - Horizontal Menu Bar: Positioned between the header and main content area; contains navigation options to different pages.

3. Page 1: Create New "Hello World" Message Post
   - Input Box: Simple text box to enter a new "Hello World" message.
   - Post Button: Submits the message to be saved and displayed on the message view page.
   - Validation: Ensure that the message box is not empty before allowing submission.

4. Page 2: "Hello World" Message View
   - Initial Message: Display a static "Hello World" message when the user logs in for the first time.
   - Message Update: Upon submitting a new message via the Create New "Hello World" page, this page should display the latest message.
   - Persistent Storage: Messages should be stored and retrieved from the backend (Strapi) to ensure they persist between sessions.

5. Security & Access Control:
   - Ensure that no pages are accessible without authentication.
   - Implement session management and expiration for security.

First Enhancement Request – this will help us understand how the development workflow will be structured.

1. Wall of "Hello World" Posts on Login Page:
   - Login Page Update: Replace the simple login page with a wall of "Hello World" posts.
   - Post Display: Display all "Hello World" posts in reverse chronological order (most recent at the top).
   - Login Mechanism: Include the login fields (user ID and password) on this page, allowing users to authenticate directly from the wall of posts.
   - Pagination: If the number of posts is large, implement pagination or infinite scroll to load more posts as the user scrolls down.

Lifecycle Events for the Technology Stack

1. **New Requirements for GUI Modification**:
   - Gathering and analyzing new user requirements.
   - Updating and testing components in React.
   - Deploying changes to the production environment.

2. **New Requirements for Data Model Changes**:
   - Analyzing how new requirements impact the existing data model.
   - Updating the data schema in Strapi and RDS.
   - Migrating and validating existing data to fit the new model.

3. **Customization of Graphic Style (CSS Tweaking)**:
   - Updating CSS or design tokens in React.
   - Ensuring consistent styling across all components.
   - Testing changes for cross-browser compatibility.

4. **Backup**:
   - Regularly scheduling backups for databases (RDS) and storage (S3).
   - Ensuring configuration backups for services like Strapi and Lambda functions.

5. **Environment Configuration Management**:
   - Managing different environments (development, staging, production).
   - Ensuring configuration consistency across environments.

6. **Restore from Backup**:
   - Implementing and testing restore procedures for databases and file storage.
   - Ensuring minimal downtime during the restore process.

7. **Notification of Security Patch**
- How do we get notified that a security patch is available

8. **Security Patch for React**:
- Applying updates to ensure the latest security patches are in place.
- Regularly monitoring for new releases and vulnerabilities.

9. **Security Patch for Strapi**:
   - Applying updates and patches to Strapi.
   - Monitoring for security vulnerabilities and addressing them promptly.

10. **Security Patch for Custom Logic**:
   - Regularly reviewing and updating custom backend logic.
   - Ensuring serverless functions or containers are secure and up-to-date.

11. **Integration with Third-party Services**:
- Managing and updating integrations with other services and APIs.
   - Ensuring compatibility and security in data exchange.

Lifecycle Events for the Technology Stack

12. **Updating Security Certificate**:
   - Monitoring expiration dates for SSL/TLS certificates.
   - Renewing and updating certificates in services like ELB and Route 53.

13. **Dependency Management**:
   - Keeping track of third-party libraries and dependencies.
   - Regularly updating dependencies to avoid security vulnerabilities.

14. **End-of-Life for Components**:
   - Planning for the deprecation and replacement of outdated components or services.
   - Ensuring a smooth transition with minimal disruption to users.

15. **Performance Optimization**:
   - Monitoring and analyzing performance metrics.
   - Scaling resources in EKS or adjusting Lambda configurations as needed.

16. **Compliance and Auditing**:
   - Regularly reviewing and ensuring compliance with relevant regulations.
   - Performing security audits and implementing recommendations.

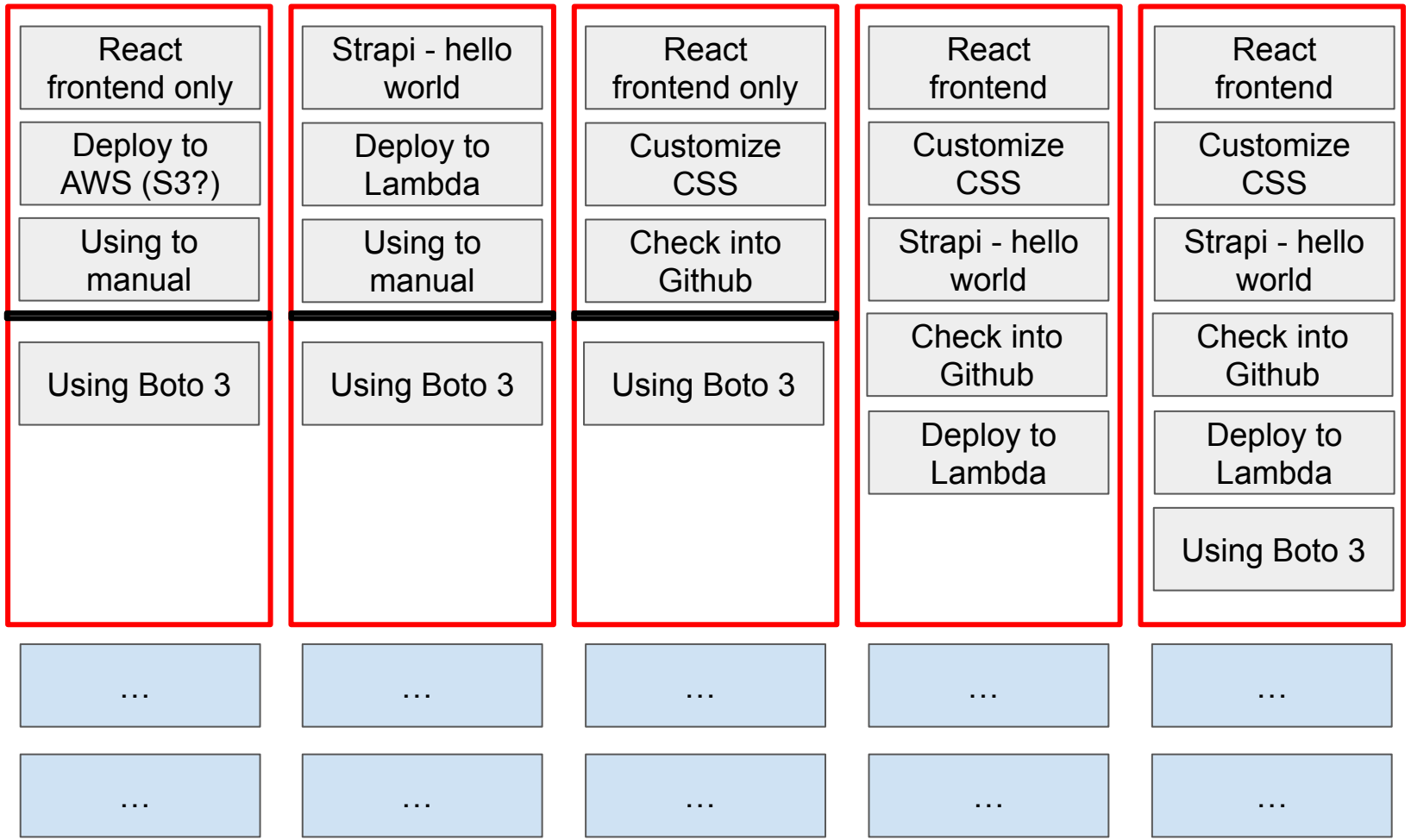
17. **Monitoring and Logging**:
   - Setting up monitoring and logging for all components (e.g., using CloudWatch for AWS services).
   - Regularly reviewing logs for anomalies or issues.

18. **Disaster Recovery Planning**:
   - Developing and testing disaster recovery plans.
   - Ensuring data and services can be quickly restored in case of a major failure.

19. **Deployment Pipeline Maintenance**:
   - Regularly updating CI/CD pipelines.
   - Ensuring smooth and automated deployment processes.

Viability Thread

Parking lot ??



Lifecycle Events for the Technology Stack

App Smith and Tool Jet

1. Drag and Drop App Construction
2. React or Next base
3. Deployment to AWS
4. Ability to
5. Build Export

Common Feature

1. Charting
2. Organic Groups
3. Uploads / Attachments
4. Maps
5. Forms
6. Dynamic Menus (context and user based)
7. Image Cache (see Drupal)
8. Tabular data / spreadsheets
9. Social Login
10. Social features: commenting, liking
11. Using custom tool-chain architecture → drag and drop to cloud
12. LOW PRIORITY: Code Editors
13. LOW PRIORITY: Rich Text Editor
14. LOW PRIORITY: Blog like / web content
15. LOW PRIORITY: Jupyter Notebooks

Mapeditor

JWT authorization

Frontend React

Backend Strapi

- Entities and other configuration
- Roles
- Register
- Self Service

Drupal / Organic Group

https://chatgpt.com/share/148020e6-c624-4956-b5b6-da904eae9837