Renova

Design Document

sdmay26-16
Advisor MD Maruf Ahamed

Team Members
Prakeerth Regunath: Frontend & Scheduler/Organizer
Onur Onal: Project Lead/Project Manager & Full-stack
Kemal Yavuz: Full-stack
Erroll Barker: Lead Frontend UX/UI
Bae Meh: Backend
Om Sanghvi: Backend

Team Email
sdmay26-16@iastate.edu
Team Website
https://sdmay26-16.sd.ece.iastate.edu/

Revised: 5/4/2026

Executive Summary

Renova is a web-based platform that combines user creativity with the power of artificial intelligence. The core problem we address is the high cost, time commitment, and expertise required for professional interior design services. Many homeowners and renters struggle to visualize renovation ideas before committing financially, which can lead to dissatisfaction and wasted resources. Renova makes this process accessible, intuitive, and risk-free.

Our key design requirements are to create a user-friendly moodboard editor for arranging room items, a robust backend to manage user data and AI job orchestration, and seamless integration with multiple AI models to generate photorealistic interior previews based on user designs. The platform must generate high-quality, realistic previews of renovated spaces based on the user's unique design choices.

The final design is decomposed into a Next.js frontend for the interactive editor and account workflows, an Express backend API for authentication, persistence, billing, media, Pinterest, and AI orchestration, a MySQL database accessed through Prisma, S3-backed object storage for media, and Fal-hosted image models for generation and background removal. The current system no longer depends on a separately deployed custom preprocessing service; instead, the backend coordinates external AI calls directly and records the results in the application database.

Progress has moved beyond planning into a functioning full-stack implementation. Users can register or sign in, create and edit moodboards, upload and transform images, import inspiration from Pinterest, generate AI previews from selected board items, review generated images, and manage account, plan, and billing-related workflows.

The remaining work is focused less on defining the architecture and more on final polish: strengthening automated test reliability, tightening authentication and media-access hardening, improving generation feedback and error handling, and refining the user experience around credits, plans, and long-running AI requests.

Learning Summary

Development Standards & Practices Used

Software Development: Agile methodology, Version Control (Git), RESTful API design, Modular code structure.

Frontend: Component-based architecture (Next.js), Responsive web design principles.

Backend: Secure Authentication (JWT), Database Normalization (MySQL), API Endpoint security.

Machine Learning / AI Integration: Use of Fal-hosted Seedream 4.0, Seedream 4.5, Seedream 5.0 Lite, Google Nano Banana, and Google Nano Banana 2/Pro edit models, plus background removal through Fal/BiRefNet. The implementation encapsulates model-specific request formats inside the backend rather than through a separate custom preprocessing service.

Summary of Requirements

Functional: User registration/login, Image upload to a personal library, Drag-and-drop moodboard editor (resize, rotate, layer items), AI-powered renovation preview generation with selection of multiple AI Models.

Applicable Courses from Iowa State University Curriculum

- COM S 309: Software Development Practices
- SE 319: Construction of User Interfaces
- COM S 228: Introduction to Data Structures
- COM S 339: Software Architecture
- COM S 363: Introduction to Database Management Systems
- COM S 417: Software Testing
- Various Machine Learning / Data Science courses

New Skills/Knowledge acquired that was not taught in courses

- Integration of multiple AI models into a single pipeline.
- Specifics of external AI APIs, multipart image preparation, presigned media URLs, model-specific prompt/input formats, and background-removal workflows.
- Using React-Konva and Zustand for advanced canvas interactions, including object transforms, crop modes, undo/redo history, text editing, drawing, and layer ordering.
- Orchestrating a multi-service full-stack architecture across Next.js, Express, Prisma/MySQL, S3, Stripe, Pinterest, and Fal-hosted AI services.

Table of Contents

1. Introduction 6
    1.1. Problem Statement 6
    1.2. Intended Users 6
2. Requirements, Constraints, and Standards 6
    2.1. Requirements & Constraints 6
    2.2. Engineering Standards 7
3. Project Plan 8
    3.1. Project Management/Tracking Procedures 8
    3.2. Task Decomposition 8
    3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria 8
    3.4. Project Timeline/Schedule 9
    3.5. Risks and Risk Management/Mitigation 11
    3.6. Personnel Effort Requirements 12
    3.7. Other Resource Requirements 12
4. Design 12
    4.1. Design Context 12
        4.1.1 Broader Context 13
        4.1.2 Prior Work/Solutions 14
        4.1.3 Technical Complexity 14
    4.2. Design Exploration 15
        4.2.1 Design Decisions 15
        4.2.2 Ideation 16
        4.2.3 Decision-Making and Trade-Offs 17
    4.3. Final Design 18
        4.3.1 Overview 18
        4.3.2 Detailed Design and Visual(s) 18
        4.3.2.1 Billing & Entitlements (Stripe) 21

4.3.3 Functionality 21
4.3.3.1 Billing User Flows 22
4.3.4 Areas of Challenge 22
4.4. Technology Considerations 23
4.4.1 Stripe Integration 23
5. Testing 23
5.1. Unit Testing 24
5.2. Interface Testing 24
5.3. Integration Testing 24
5.4. System Testing 25
5.5. Regression Testing 25
5.6. Acceptance Testing 26
5.7. Security Testing 26
5.8. User Testing 27
5.9. Testing Results 27
6. Implementation 28
6.1. Design Analysis 28
7. Ethics and Professional Responsibility 29
7.1. Areas of Professional Responsibility/Codes of Ethics 29
7.2. Four Principles 31
7.3. Virtues 32
8. Conclusions 34
8.1. Summary of Progress 34
8.2. Value Provided 35
8.3. Next Steps 35
9. References 36
10. Appendices 37
10.1. Appendix 1 – Operation Manual 37
10.2. Appendix 2 – Alternative/initial version of design 37
10.3. Appendix 3 – Team Contract 37

List of figures/tables/symbols/definitions

Figures

- **Figure 4.3.2.01** Full Renova moodboard editor page showing the sidebar, canvas workspace, board actions, and editing toolbar.

Tables

- **Table 3.4.1** - Project Schedule (Semester 1 & 2 Sprint Breakdown). Provides a sprint-by-sprint breakdown of frontend, backend, and ML service tasks, along with integrated deliverables.

- **Table 3.6.1** - Personnel Effort Requirements. Estimates the number of hours required for major tasks, including frontend, backend, and ML service development.

- **Table 4.1.1.1** - Broader Context Considerations. Summarizes the project's impacts in public health, global/cultural/social, environmental, and economic areas, with specific examples.

- **Table 4.2.3.1** - Decision-Making and Trade-Off Matrix. Compares multiple frontend canvas implementation options and highlights the rationale for selecting React-Konva.

Symbols

- **POST /api/generation - Core backend API endpoint that accepts selected board images, an optional base room image, a prompt, and a model key, then orchestrates AI generation and stores the resulting Generation record.**
- **Presigned URL - A temporary S3 access URL used by the backend to let external services retrieve private input images without exposing permanent public storage access.**
- **MVP (Minimum Viable Product) - The baseline version of the system; in the current final design, this has expanded into an implemented full-stack editor, generation, account, Pinterest, and billing workflow.**

Definitions

1. **Moodboard Canvas** - An Interactive editor where users arrange furniture and décor assets over a room photo.

2. **Design Prompt:** A text description derived from the user's moodboard layout, used to guide AI image generation.

3. **Generation Job:** A request sent to Image Generation AI containing the user's room image, placed assets, and design prompt to produce a renovated preview.

4. **Signed URL** - Secure, time-limited link granting users access to files stored in object storage.

5. Introduction

1.1. Problem Statement

Imagine you want to redecorate your living room. You have ideas: a new sofa, different colored walls, and a modern lamp. However, you can't picture how it will all look together. Hiring an interior designer is expensive, and buying furniture only to discover it doesn't fit the space is a costly mistake. This kind of scenario is a common frustration for homeowners and renters. The problem is the gap between imagination and reality in home renovation.

Renova addresses this by providing a digital sandbox where users can visually plan their space and see an AI-generated preview of their ideas applied to their actual room. The system is implemented as a full-stack web application with a web frontend, backend API, database storage, and external AI generation services. As a result, it can help reduce financial risk, boost confidence in design decisions, and make interior design accessible to everyone, not just professionals.

1.2. Intended Users

The DIY Homeowner: Tech-savvy individuals who enjoy home improvement projects but lack formal design training. They need a tool to experiment with styles and visualize changes without commitment. Renova provides them with a risk-free environment to plan and perfect their renovations.

The First-Time Renter: Young adults living in rental properties who want to personalize their space within landlord constraints. Their need is for non-permanent, reversible design ideas. Renova helps them visualize temporary changes, such as new furniture and decor, ensuring they love the result before making a purchase.

The Interior Design Student: Students learning design principles who need a rapid prototyping tool to test concepts and build portfolios. They benefit from Renova's rapid iteration cycle, which enables them to efficiently explore multiple design variations for a single room.

2. Requirements, Constraints, and Standards

2.1. Requirements & Constraints

User Account Management: The system shall allow users to create an account and log in. (Functional)

Moodboard Editor: The system shall provide a canvas where users can drag, drop, resize, rotate, crop, mirror, layer, duplicate, shadow, and delete images. Users can add text boxes and freehand drawings, change the board background, upload local images, remove image backgrounds, and use images imported from Pinterest. (Functional, User Experience)

AI Renovation Trigger: The system generates a renovation preview when a user selects at least one board item, optionally uploads a base room image, chooses an AI model, enters a prompt, and clicks the Generate button. (Functional)

Preview Display: The system shall display generated previews in a gallery with the ability to view their prompt, as well downloading their AI-Generated Image. (Functional)

Performance: AI preview generation process should complete within a reasonable time (target under 120 seconds), depending on the external AI service's response time. (Constraint)

Data Security: User passwords shall be hashed before storage in the database. Authentication uses JWT access and refresh tokens, with protected backend routes enforcing user identity and role checks. Media is stored in S3 and served through controlled backend or Next.js proxy routes so the frontend does not need permanent direct access to private object storage. A future hardening improvement is to move refresh-token handling to hashed server-side session records and HttpOnly cookies.

2.2. Engineering Standards

Importance of Standards: Engineering standards ensure consistency, interoperability, and safety across systems. They provide a common framework that enables the smooth integration of software and hardware components, while also protecting user data and ensuring quality. For a project like Renova, standards help the team adopt industry best practices in security, software engineering, and data handling.

Relevant Standards:

- **ISO/IEC 27001 (Information Security Management):** Ensures secure handling of user data, aligning with our requirement to hash and protect user passwords.
- **IEEE 830 (Software Requirements Specification):** Guides the documentation of functional and non-functional requirements, ensuring clarity and completeness in system design.
- **ISO/IEC 25010 (System and Software Quality Models):** Provides a framework for evaluating software quality (usability, performance, reliability), which directly applies to our functional and non-functional requirements.

Relevance to Project: These standards are directly applicable because they address secure data management, clear documentation of requirements, and measurable quality benchmarks, all essential for Renova's success.

Modifications to Design for Standards:

- Adopt secure password handling, token-based authentication, and encrypted communication in line with ISO/IEC 27001.
- Formalize requirements documentation following IEEE 830, ensuring traceability to user needs.
- Use ISO/IEC 25010 as a reference when evaluating usability and performance during testing.

3. Project Plan

3.1. Project Management/Tracking Procedures

We have adopted an Agile methodology with elements of Scrum to manage the development of Renova. This project management style is justified by the complex and interdependent nature of the current system: an interactive moodboard editor, an Express/Prisma backend, S3 media storage, Stripe billing, Pinterest import, and Fal-hosted AI generation models. Agile allowed the team to adapt as the design shifted away from a custom ML preprocessing service and toward a backend-orchestrated external AI pipeline.

Tracking tools: GitLab (for code repositories and issue assignment and tracking), Discord (for weekly in-person team meetings and communication), Microsoft Teams (for advisor team meetings).

3.2. Task Decomposition

The project is decomposed into three main task streams, mirroring the implemented architecture:

1. **Frontend (Next.js) Stream**:
    a. User Authentication
    b. Asset Library
    c. Canvas View
    d. Dashboard View (Explore, Generations, Moodboards)
    e. Consistent Theme (Light/Dark)
2. **Backend (Node.js) Stream**:
    a. User Authentication API
    b. CRUD APIs for Projects/Boards/Assets
    c. AI Generation Orchestration
    d. Database Schema & Integration
3. **AI, Integrations, and Infrastructure Stream:**
    a. Fal model integration and request encapsulation
    b. Seedream 4.0 / 4.5 / 5.0 Lite integration
    c. Google Nano Banana 2 / Pro integration
    d. Fal/BiRefNet background-removal integration
    e. Stripe subscription, add-on, and invoice integration
    f. Pinterest OAuth, board browsing, pin import, and rate limiting
    g. Docker, GitLab CI/CD, health checks, and deployment automation
    h. Media proxying through S3 presigned URLs and Next.js proxy routes

3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria

- Milestone 1 (MVP Foundation): A user can log in, create and use an interactive moodboard, upload a room image or design asset, and receive a basic AI-generated renovation preview.
    - Metric: Working moodboard persistence and successful end-to-end generation through the backend /api/generation route and a supported Fal edit model.

- Milestone 2 (Feature Completion): Advanced moodboard tools, including background color editing, text, drawing, crop, layer ordering, remove background, autosave, and billing/plan usage enforcement are functional.
    - Metric: Advanced canvas tools work inside saved boards, and sandbox subscription/add-on flows update plan and usage state through the backend.
- Milestone 3 (Final Delivery): Full integration of frontend, backend, database, media storage, AI generation, Pinterest, billing, CI/CD, and deployment workflows.
    - Metric: Users can complete the main flows end-to-end: authenticate, create/edit/save a board, import or upload assets, generate with a selected AI model, view results, and manage plan or billing state.

3.4. Project Timeline/Schedule

Detailed Sprint Schedule & Task Breakdown (Semesters 1 & 2)

Semester 1: MVP (Minimum Viable Product) Foundation

Sprint	Timeframe	Frontend Tasks	Backend Tasks	ML Tasks	Integrated Deliverables
1-2	Early Sept - End Oct	- Set up Next.js app. - Create Login/Signup Components. - Build basic dashboard layout.	- Set up Express.js server. - Design MySQL schema (Users, Projects). - Implement JWT authentication API.	- Research Seedream 4.0. - Document API call parameters for inpainting.	Basic authenticated application skeleton.
3-4	Early Nov- Mid Nov	- Develop "My Uploads" page. - Implement image uploader component & gallery view.	- Build Asset CRUD APIs (upload, list, delete). - Set up cloud storage for images.	- Create a proof-of-concept script: call the Seedream 4.0 with a test image and prompt.	Users can upload and manage their image library.
5-6	Mid Nov- End Nov	- Implement core Moodboard Canvas with React-Konva. - Enable drag, drop, and resize of images from the library onto the canvas.	- Create Project/Board CRUD APIs. - Develop an endpoint to save/load canvas JSON state.	- Formalize the "job" data structure needed from the frontend (image URLs, mask data, prompt). - Build a helper module to construct Seedream 4.0 API calls.	Functional moodboard editor that saves/loads state.

7-8	Early Dec - Mid Dec	- Build Preview Gallery UI for side-by-side comparison. - Connect "Generate" button to backend endpoint. - Polish full user flow.	- Build the /render job orchestration endpoint. - Integrate Seedream 4.0 API module; handle async job polling and status updates. - Store the final generated images.	- Develop basic error handling for AI generation failures.	**Milestone 1: MVP Demo** End-to-End Flow: Moodboard -> AI Preview -> Comparison.

Semester 2: Enhancement & Integration

Sprint	Timeframe	Frontend Tasks	Backend Tasks	ML Tasks	Integrated Deliverables
9-10	Mid Jan- End Jan	- Add moodboard features: rotate, layer ordering (z-index). - Implement undo/redo functionality.	- Refactor backend to call the microservices. - Create new endpoints to handle the microservices and backend - Update data model for new pipeline.	- Set up FastAPI server. - Create a depth map generation endpoint. - Dockerize the service.	Enhanced moodboard. Functional depth map API.
11-12	Early Feb- Mid Feb	- Develop a UI for manual mask refinement (contingency for Risk 2). - Add visual feedback for the generation pipeline status.	- Update the /render pipeline: Backend -> Microservice -> Seedream 4.0 API.	- Combine depth & mask generation into a unified preprocessing pipeline.	**Milestone 2: Automated Masking** Full AI pipeline using the custom microservice.

13-14	Mid Feb- End Mar	- Performance optimization (e.g., virtual scrolling in asset library). - Conduct internal usability tests and gather feedback.	- Orchestrate and optimize the full, multi-step pipeline. - Implement caching for results to reduce costs and latency. - Conduct backend load testing.	- Optimize model inference speed (e.g., model quantization). - Improve mask accuracy and handle edge cases from user tests.	Fully integrated, optimized, and stable AI renovation pipeline.
15-16	Early Apr- End May	- Bug fixing, UI/UX polish, and final responsiveness checks. - Conduct internal usability tests and gather feedback. - Conduct formal user testing with target users.	- Final security review and API hardening. - Prepare for deployment (environment variables, configs).	- Final model tuning based on user test data. - Prepare microservice for production deployment.	**Milestone 3: Final Delivery** A polished, tested, and deployable product.

3.5. Risks and Risk Management/Mitigation

- Risk 1: The AI Model's cost exceeds the ideal budget due to high usage.
  - Probability: Medium
  - Mitigation: Implement backend usage limits per account, enforce monthly and add-on credit balances server-side, prevent duplicate add-on grants through processed session tracking, and monitor high-cost model usage.
- Risk 2: Most AI models may not accurately interpret user layouts.
  - Probability: High
  - Mitigation: Let users provide stronger inputs through selected board items, an optional base room image, custom prompts, and model choice. The editor also supports remove-background and crop tools so references can be cleaned up before generation. Further work can add better progress indicators, cached retries, and clearer guidance when model output quality varies.
- Risk 3: Performance lag in the React-Konva canvas with many items.
  - Probability: Medium
  - Mitigation: Implement virtual scrolling for the asset library sidebar and optimize the React-Konva redraw logic to maintain smoothness during interaction.

3.6. Personnel Effort Requirements

Task	Estimated Effort per Semester (Hrs)	Actual Effort per Semester (Hrs)
Moodboard Frontend	80	110
Moodboard Backend	100	140
Total	**~180**	**~250**

The difference in effort per semester was driven by substantial overruns in both frontend and backend development, which required 30 and 40 additional hours, respectively, beyond their original projections. These increases likely stemmed from the technical complexity of integrating APIs and AI generation models, as well as the formation of a multi-component application architecture.

3.7. Other Resource Requirements

Other required resources include GitLab for source control and CI/CD, Docker and Docker Compose for local and production containers, a MySQL database, AWS S3-compatible storage for media, Stripe test/production configuration, Pinterest developer credentials, Fal API credentials, Google OAuth credentials, and a production VM capable of running the frontend, backend, and database services.

4. Design

4.1. Design Context

Renova is a web-based interior design planning platform centered on a moodboard editor and an AI-assisted visualization workflow. The system allows users to create an account, manage personal boards, upload images, arrange items on an interactive canvas, and generate AI-based room redesign outputs based on selected board assets and optional room reference images. The current implementation reflects a practical product-oriented architecture rather than the more speculative pipeline described in earlier drafts.

The system now consists of a Next.js frontend, an Express backend, a MySQL database accessed via Prisma, S3-backed media storage, and external AI services integrated through Fal for Seedream, Nano Banana, and background-removal workflows. This architecture supports the actual user workflow implemented: board creation, board editing, autosave, image upload, Pinterest import, background removal, AI generation, account management, billing, and image retrieval.

The design context is shaped by three primary concerns. First, the editor must be responsive enough to support direct manipulation tasks such as adding, selecting, transforming, layering, cropping, and styling visual assets. Second, the backend must persist both structured board state and unstructured image media in a way that remains manageable as the application grows. Third, the AI generation pipeline must be integrated in a way that is realistic for a semester project, meaning the team can deliver useful functionality without taking on the overhead of hosting and operating custom vision or diffusion infrastructure.

4.1.1 Broader Context

Renova exists within a broader social, technical, and economic context than a simple class project. It is intended to reduce uncertainty in home design decisions, lower user experimentation costs, and make visual design exploration more accessible. At the same time, it operates on user-uploaded images and depends on external AI generation, so privacy, transparency, and system reliability remain important design concerns.

Area	Description	Examples
Public health, safety, and welfare	Users may make purchasing or renovation decisions based on generated previews. Outputs should support exploration but not be interpreted as construction-ready plans or guarantees of real-world fit.	Renova is positioned as a visualization and planning aid rather than a certified design or construction tool. The system emphasizes previewing concepts rather than validating measurements, engineering feasibility, or code compliance.
Global, cultural, and social	Interior design preferences are highly personal and culturally influenced. A design platform should not assume a single “default” aesthetic.	The current system supports open-ended image-based workflows through custom uploads, Pinterest import, and prompt-driven AI generation rather than forcing users into a fixed template library. This gives users more control over style direction.
Environmental	Poor design decisions can lead to wasteful purchases and unnecessary replacement cycles. A tool that supports preview-before-purchase can reduce this risk.	Renova helps users experiment digitally before spending money on furniture or décor. By testing layouts and styles virtually, users can make more informed decisions before buying physical products.
Economic	Traditional interior design support can be expensive, while furniture mistakes are costly for users.	Renova is designed to provide a lower-cost, self-serve alternative for early-stage design exploration. The platform’s board workflow and AI preview system help users evaluate ideas before making purchases.

4.1.2 Prior Work/Solutions

Renova is related to several existing tool categories, but it does not align perfectly with any single one. Traditional interior design software often offers advanced layout planning, but these tools are often too complex for casual users and are usually aimed at professionals. At the other end, lightweight inspiration tools make it easy to collect ideas but do not provide a robust workflow for turning those ideas into an interactive design surface connected to AI-generated outputs.

The closest practical comparisons are moodboard tools, general-purpose design platforms, and AI room redesign products. Moodboards and design tools are useful for collecting and arranging visual inspiration, but they typically stop at composition and do not generate a transformed room output. AI room redesign tools can generate interesting results, but they often provide limited user control over what specific items or references influence the output. Renova attempts to bridge those two spaces by combining a user-controlled visual board with an AI generation step.

The major distinction in Renova's current design is that the generation flow is grounded in the actual board state. Users are not only typing a prompt; they are selecting existing board items, optionally uploading a base room image, and using those references as inputs to the generation request. This makes the system more interactive and design-oriented than a prompt-only generator, while remaining lighter-weight than a full professional design suite.

4.1.3 Technical Complexity

Renova meets the project's technical complexity expectations through the interaction of multiple subsystems, each with distinct implementation concerns.

On the frontend, the project includes a stateful interactive canvas editor implemented with React-Konva and a Zustand board store. This editor supports image placement, object selection, drag-and-drop, resizing, rotation, duplication, z-order changes, mirroring, shadow toggling, text insertion and live editing, freehand drawing, image/text/drawing crop modes, background color editing, keyboard shortcuts, undo/redo, thumbnail export, and full-board image export. Building these interactions in a stable way requires nontrivial client-side state management and careful coordination between the canvas, toolbar, sidebar, and persistence logic.

On the backend, the system manages authenticated CRUD operations for users, moodboards, generations, billing-related flows, and Pinterest integration. It also handles multipart uploads, persistence of board scene JSON, metadata storage for generated content, protected media retrieval, S3 presigned URL generation, role-based admin routes, plan entitlement checks, and integration with external services such as Stripe, Pinterest, and Fal.

The AI workflow adds another layer of complexity. User-selected board images and optional room inputs must be converted into multipart files, uploaded into S3, transformed into presigned URLs, sent to the selected Fal model, retrieved again once generation is complete, uploaded into managed storage, and linked back into the database as a Generation record. A related background-removal workflow sends a selected image through Fal/BiRefNet, stores the processed PNG, and replaces the item on the board.

The project is therefore technically complex, not because of any single algorithm, but because it combines interactive frontend engineering, backend orchestration, cloud media handling, database persistence, and third-party AI integration into one coherent user-facing system.

4.2. Design Exploration

The design of Renova changed substantially as the team moved from early conceptual planning into implementation. Earlier versions of the design assumed a more elaborate ML-heavy pipeline. As implementation progressed, the team shifted toward a more pragmatic architecture that preserved the core user experience while reducing unnecessary infrastructure complexity.

The design exploration phase focused primarily on four questions:

1. How should the interactive board editor be implemented?
2. How should board state and media assets be stored?
3. How should AI generation be integrated without creating an unmanageable infrastructure burden?
4. How should external media and third-party content be introduced into the editor safely and consistently?

These questions shaped the major design decisions described below.

4.2.1 Design Decisions

Interactive canvas editor using React-Konva

The team chose a dedicated canvas abstraction instead of building all object interaction logic on top of the raw browser canvas API. The editor required selection boxes, transforms, drag-and-drop interactions, cropping overlays, and layering behavior, all of which are easier to manage with a React-friendly abstraction.

The implemented editor also separates reusable interaction concerns. BoardCanvas renders images, text, and drawings as different Konva item types; Toolbar exposes selection-specific actions; BoardActions exposes board-level actions; and the Zustand store keeps the current scene, selection, clipboard, mode, and undo/redo history in one shared client-side state model.

Structured board state stored as scene JSON

Rather than flattening every editor change into separate relational tables, Renova stores the logical moodboard contents as structured JSON under the scene field of a moodboard. This simplifies the editor's save/load behavior and allows the frontend to reconstruct the board from a single persisted object.

Separation of metadata and binary media

The team chose to store structured entities such as users, moodboards, and generations in MySQL through Prisma, while placing uploaded thumbnails, board item images, profile images, and generated images in S3. This keeps the database focused on relational data and avoids storing large binaries directly in SQL.

Backend-orchestrated AI generation

Instead of maintaining a separate custom ML preprocessing service, the current implementation uses the Express backend as the orchestration point for generation. The backend uploads source files, creates presigned access paths for generation inputs, calls Seedream through Fal, stores the resulting image in S3, and records the result in the database.

Background removal is handled as a smaller AI-assisted editing feature rather than a separate preprocessing pipeline. When the user selects an image and clicks Remove BG, the backend verifies board ownership, creates a presigned URL for the stored image, sends it to Fal/BiRefNet, uploads the returned PNG to S3, and returns a new image source for the selected board item.

Media delivery through controlled routes

Because user images and generated assets are stored externally, the project needed a safe way to display them in the UI. Renova uses backend-controlled media routes and proxy patterns so clients do not have to directly manage private storage access.

Pinterest as an input source rather than a first-class storage system

Pinterest integration was designed to support the import of inspiration into the editor without making Pinterest the system of record. Pins are browsed through the integration flow, then copied into the board as regular image items.

4.2.2 Ideation

During design exploration, the team considered several possible directions for the editor and generation architecture.

For the **editor**, the main alternatives were:

- Native HTML5 Canvas with custom interaction logic
- SVG-based rendering with a library such as D3
- Fabric.js
- React-Konva
- p5.js

For the **generation backend**, the main alternatives were:

- A separate preprocessing service pipeline
- A backend-managed orchestration flow that directly integrates with an external AI service
- A self-hosted image generation stack

For the **data model**, the team also considered whether moodboards should be modeled as many normalized relational records or as a single persisted scene structure with external media references.

As implementation began, the team realized that the most important design goal was not theoretical flexibility, but a realistic path to shipping a working end-to-end product. That pushed

the design toward simpler orchestration, fewer independently deployed services, and a frontend architecture that accelerated editor feature development.

4.2.3 Decision-Making and Trade-Offs

The most consequential design trade-off on the frontend was the choice of canvas technology. Because the editor is central to the product, the team evaluated several options based on React compatibility, implementation effort, interaction support, maintainability, and suitability for the required moodboard workflow.

Option	**React Integration**	**Implementation Effort**	**Interaction Support (drag/resize/ rotate/crop)**	**Maintainability**	**Overall**
Native HTML5 Canvas	Low	Very High	Low without substantial custom code	Medium	Medium-Low
SVG / D3.js	Medium	High	Medium	Medium	Medium
Fabric.js	Medium	Medium	High	Medium	High
React-Konva	High	Low-Medium	High	High	Very High (chosen)
p5.js	Medium	Medium	Medium	Medium-Low	Medium-Low

Trade-Off Discussion

Native HTML5 Canvas offered the greatest theoretical control, but it would have required the team to implement nearly all object manipulation behavior manually. That would have significantly increased development time and risk for a feature that already sits on the product's critical path.

SVG-based approaches were considered because they are expressive and easy to inspect, but they are less natural for a canvas-style design editor with layered image manipulation, transform controls, and cropping overlays. D3 in particular is stronger for visualization than for building a general-purpose editor.

Fabric.js was a credible alternative because it provides many of the required editing features out of the box. However, React-Konva aligned better with the project's React-based frontend structure and offered a cleaner integration model for component-driven state and rendering.

p5.js was not selected because it is more oriented toward sketching, creative coding, and experimental graphics than production-style editor workflows. It would not have reduced the amount of application-specific interaction logic the team still needed to write.

React-Konva ultimately provided the best balance. It reduced implementation overhead while still supporting the interactions required for the board editor. The trade-off is dependence on an additional library and the need to be mindful of rendering performance as the number of board items grows. The team judged that this trade-off was acceptable because it allowed the editor to become functional much earlier and with less framework friction.

4.3. Final Design

4.3.1 Overview

The proposed design is now best described as a full-stack web application centered on user-managed moodboards and AI-assisted generation, rather than a speculative rendering pipeline. The user journey is as follows:

1. A user creates an account or logs in.
2. The user creates a new moodboard or opens an existing one.
3. Inside the editor, the user uploads images, imports inspiration from Pinterest, adds text or drawings, and arranges selected items on the board.
4. The board state is persisted via save and autosave.
5. The user optionally selects board items and a base room image, adds a prompt, and requests an AI generation.
6. The backend orchestrates the generation request, stores the output, and links it back to the user and optionally to the moodboard.
7. The user can revisit generated images, continue editing the board, and manage account and plan-related settings through the dashboard.

This design keeps the moodboard editor at the center of the product. Other features, such as authentication, account management, media storage, Pinterest integration, and generation history, exist primarily to support that central workflow.

4.3.2 Detailed Design and Visual(s)

The Renova moodboard page serves as the system's primary workspace and brings together the project's main user-facing features in a single interface. The page is designed around a central editable canvas, supported by surrounding controls for board management, editing actions, inspiration browsing, and AI generation. This layout reflects the platform's core workflow by keeping the moodboard at the center of the user experience while placing supporting tools around it.

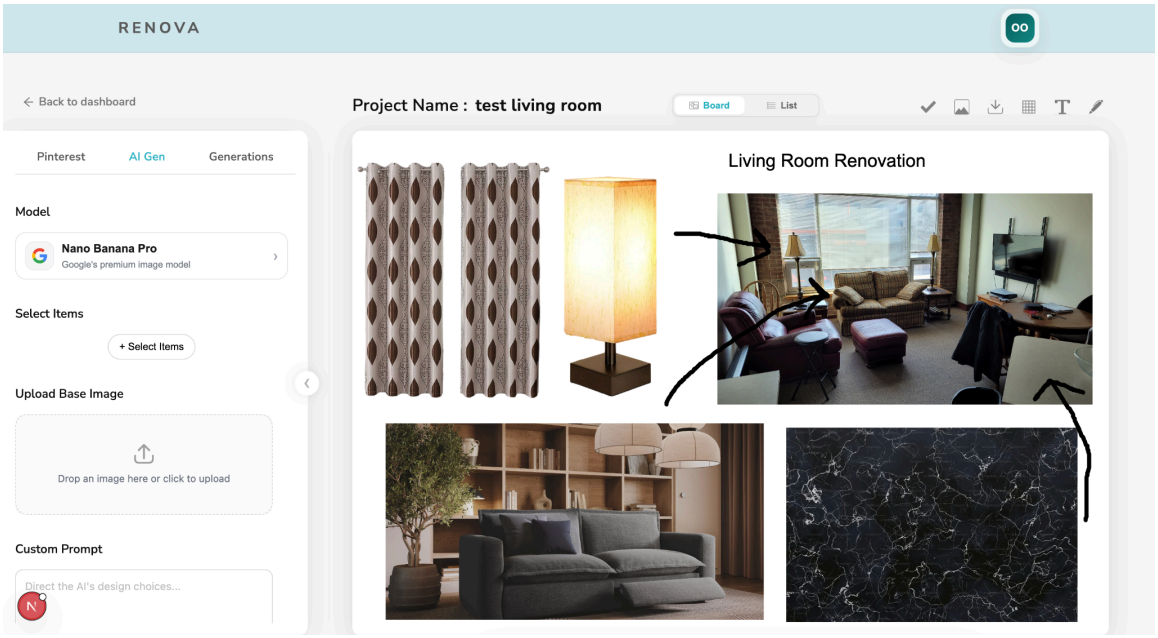

Fig. 4.3.2.01

The page is organized into several functional areas. The left sidebar provides access to Pinterest content, AI generation inputs, and previously generated images. The central canvas area displays the active moodboard and supports direct interaction with items placed on the board. Above and around the canvas, the interface includes board-level actions such as save, image upload, export, background editing, text insertion, and drawing. When an item is selected, the toolbar provides object-level editing operations such as crop, mirror, duplicate, reorder, shadow toggle, and delete.

The supporting system architecture behind this interface can be understood as four major layers:

1. Frontend Layer (Next.js / React)
This layer includes:

- public landing pages
- authentication pages
- dashboard pages
- moodboard collection view
- individual moodboard editor pages
- account management pages
- generation browsing views

Within the editor, the frontend includes:

- BoardCanvas for rendering images, text, drawings, selection, transforms, crop overlays, keyboard shortcuts, and exports
- BoardActions for save/upload/export/background/text/drawing actions
- Toolbar for object-level editing operations such as remove background, crop, mirror, duplicate, shadow, layer order, and delete

- sidebar views for Pinterest, AI generation, and generation history
- Zustand board state for item storage, mode tracking, selection, clipboard, and undo/redo history

2. Backend API Layer (Express)
This layer includes:

- authentication routes
- moodboard CRUD routes
- generation routes
- media routes
- Pinterest integration routes
- user/account routes
- billing/plan routes

The backend handles authorization, validation, multipart upload processing, orchestration of generation, and database access.

3. Persistence and Media Layer
This layer includes:

- MySQL accessed through Prisma for relational and structured application data
- S3 for uploaded images, thumbnails, and generated outputs

Moodboards store metadata and scene JSON in the database, while large binary media files are stored in object storage.

4. External Services Layer
This layer includes:

- Fal for Seedream/Nano Banana AI generation and BiRefNet background removal
- Pinterest APIs for browsing boards and pins
- Stripe-related plan or billing functionality, where applicable to account management

The overall system flow begins in the frontend, where authenticated users create or open a moodboard and interact with the editor. Board data, including title, visibility, dimensions, background color, and scene contents, is sent to the backend through the moodboard API routes. The backend validates these requests and persists structured data in MySQL through Prisma. Uploaded media such as board item images, thumbnails, profile images, and generated outputs are stored separately in S3, with the application maintaining references to those assets in the database.

The AI generation workflow is coordinated through the backend /api/generation route. Users select up to five images already placed on the board, optionally provide a base room image, choose a supported model, and submit a prompt. The backend uploads the required input files, prepares presigned access paths, builds the model-specific Fal input, retrieves the generated result, stores the output in S3, and records the generation in the database. The generated image is then made available to the frontend through the generation views associated with the current user and moodboard.

Pinterest integration supports the inspiration side of the workflow. Users can connect their Pinterest account, browse boards and pins through the sidebar, and add selected images to the moodboard as regular board items. Pinterest image URLs are routed through a Next.js proxy to avoid CORS and hotlinking issues, while the backend uses a rate-limited Pinterest fetch helper to reduce the risk of hitting external API limits.

The detailed design shown in this section is more accurate than earlier drafts because it reflects the implemented Renova system rather than an assumed one. The design centers on a real interactive editor, a functioning backend orchestration layer, structured database persistence, external object storage, protected media proxying, billing entitlements, Pinterest import, background removal, and a practical AI image-generation pipeline.

4.3.2.1 Billing & Entitlements (Stripe)

Renova's billing system is implemented as a backend "Billing & Entitlements" subsystem integrated with Stripe. Stripe is used for customer management, subscriptions, payment methods, checkout, invoices, and scheduling subscription changes. Internally, Renova maps Stripe Price IDs to our database Plan table (seeded via Prisma) to determine plan entitlements and display the plan catalog consistently across the frontend and backend. This approach keeps pricing and limits consistent and allows the UI to render subscription plans and one-time add-ons from the same source of truth.

Subscription purchases and upgrades are handled through Stripe Embedded Checkout. After checkout, the backend confirms the session and enforces the "one active subscription" rule by canceling any prior active subscriptions to prevent the user from being charged for multiple plans simultaneously. Renova also supports "Switch now" versus "Switch at period end." "Switch now" starts a new billing cycle immediately, while "Switch at period end" uses a Stripe Subscription Schedule to delay the plan change until the current billing period ends.

One-time purchases (add-ons) are implemented as additional Stripe one-time Price IDs that also exist in the Plan table and are flagged as add-ons (e.g., add-on type and amount). Upon successful confirmation of an add-on purchase, Renova stores persistent credits in the database (e.g., banked generation credits or extra moodboard slots). Entitlement enforcement occurs on the backend for both moodboards and AI generations by checking the user's plan limits and then consuming banked add-on credits as needed. Invoice viewing is integrated into Renova by proxying invoice PDFs through the backend, allowing users to view invoices in-app rather than being redirected to Stripe-hosted pages.

4.3.3 Functionality

Renova currently functions as an authenticated design workspace built around editable moodboards.

A user can create a board, open it in the editor, and add content through several paths. They can upload images directly into the board, browse Pinterest boards and pins through the integration panel, insert text boxes, add freehand drawings, manipulate the board background color, and remove the background from selected image items. Once items are placed on the board, the user

can select them and perform object-level operations such as resize, rotate, mirror, crop, duplicate, reorder, add a shadow, or delete.

Board persistence is handled through explicit save and autosave. Explicit save captures the board name, visibility state, scene JSON, dimensions, background color, and thumbnail. Autosave sends debounced scene updates to the backend during ongoing editing sessions, while a separate thumbnail autosave periodically exports the Konva stage and updates the board preview image.

The AI generation flow is also board-centered. Users select one or more board items, optionally upload a base image of their room, choose a model such as Seedream or Nano Banana, provide a prompt, and trigger generation. The backend uploads the selected references, requests a new image from Fal, stores the result, and creates a Generation record. Generated outputs are then displayed in the Generations panel associated with the board and in the broader user generation gallery.

Outside the editor, Renova also includes supporting account-level functionality such as profile, security, billing, and plan-related interfaces. These features do not replace the core board workflow; they support the platform structure around it.

4.3.3.1 Billing User Flows

Users manage subscriptions and purchases from the Account to Plan tab. When a user selects a new subscription plan, Renova presents two options: switching immediately or switching at the end of the current billing cycle. If the user chooses “Switch now,” Embedded Checkout opens, payment is collected (if required), and the backend cancels the previous subscription immediately to ensure only one plan remains active. If the user chooses “Switch at period end,” Renova schedules the change to occur automatically at renewal, shows a persistent “scheduled change” banner, and allows the user to cancel the scheduled change at any time.

Renova also supports one-time add-ons for additional credits. Users can purchase add-ons from the same Plan page, complete checkout via Embedded Checkout, and the purchased credits are added to a secondary balance that does not reset monthly. When the user reaches their subscription usage limit, the system automatically consumes add-on credits. Users can also manage their billing profile by adding a payment method, choosing a default card, saving a billing address, and viewing invoices. Invoices are visible in Renova under the Billing tab, with an in-app PDF viewer and a download option.

4.3.4 Areas of Challenge

One major challenge was making the canvas feel like a real editor rather than a static image board. The team had to coordinate Konva rendering, Zustand state, selection, resizing, rotation, crop behavior, text editing, drawing, layer ordering, keyboard shortcuts, and autosave without allowing the UI state and persisted scene JSON to drift apart.

Another challenge was integrating several external systems without making the architecture too fragile. Renova depends on S3 for media, Fal for AI generation and background removal, Pinterest for inspiration import, and Stripe for subscriptions and add-ons. The team addressed this by keeping the Express backend as the orchestration point, enforcing ownership and entitlement rules server-side, and isolating service-specific details inside controllers and helper libraries.

A third challenge was scope control. Earlier designs assumed a custom depth-map preprocessing pipeline, but the final implementation focused on a more realistic and shippable workflow: user-controlled moodboards, selected reference images, model choice, backend-managed AI calls, and persistent generation history.

4.4. Technology Considerations

The final implementation uses Next.js and React for the web interface, React-Konva for the canvas editor, Zustand for board state, Ant Design for major UI controls, Express for the API layer, Prisma/MySQL for relational persistence, S3 for media storage, Fal for AI generation and background removal, Pinterest APIs for inspiration import, Stripe for billing, Docker for containerization, and GitLab CI/CD for validation, image builds, and deployment. The main trade-off is that external services reduce infrastructure burden but introduce configuration, latency, cost, and availability dependencies.

React-Konva was selected because it provides a React-friendly canvas abstraction with transform support, while still giving the team enough control to implement custom crop overlays, text editing behavior, drawing normalization, and export. Zustand was selected because the editor needs shared state across the canvas, toolbar, sidebar, and board page without forcing every interaction through deeply nested React props.

Prisma and MySQL provide a clear relational model for users, moodboards, generations, Pinterest integrations, and plans, while S3 keeps large binary images out of the database. This separation improves scalability and keeps the database focused on structured records and relationships.

4.4.1 Stripe Integration

Stripe was selected because it provides mature support for subscriptions, one-time payments, embedded checkout, saved payment methods, invoices, and subscription scheduling while offloading PCI compliance requirements. Embedded Checkout improves user experience by keeping payment flows inside Renova's UI. Stripe Subscription Schedules provide a reliable way to defer subscription plan changes to the end of a billing period without requiring custom background jobs.

Key trade-offs include dependency on correct Stripe configuration (Price IDs, API keys, environment variables) and the need to maintain a mapping between Stripe Price IDs and internal plan records. Additionally, supporting both subscriptions and one-time add-ons requires careful entitlement modeling so usage is enforced correctly and users can transparently see monthly credits versus persistent add-on balances.

5. Testing

Our testing approach directly connects to the project requirements: functional requirements are covered by unit, integration, component, or end-to-end tests, while non-functional requirements such as usability and performance are validated through manual and user testing. Because the current system involves a Next.js frontend, Express backend, database, object storage, Stripe, Pinterest, and Fal, special attention is given to the /api/generation workflow, moodboard persistence, media proxying, and billing entitlement enforcement. Automated testing exists across

the repo, but the final test suite still needs cleanup so all runners can be treated as strict blocking quality gates.

5.1. Unit Testing

Units: React components, Zustand store actions, utility functions, Express route handlers, controllers, middleware, and service helper libraries.

Tools: Jest and React Testing Library for frontend unit/component tests, Cypress and Playwright for user journeys, and Jest + Supertest for backend unit and integration tests.

The testing architecture was designed to validate React components, canvas state behavior, Express route handlers, controller logic, and helper functions. On the frontend, tests cover authentication forms, auth state, moodboard actions, toolbar behavior, sidebar views, AI generation UI states, and the moodboard editor page. On the backend, Jest and Supertest are used to test route behavior and controller logic with mocked Prisma, S3, Stripe, JWT, and related service dependencies.

5.2. Interface Testing

The primary interfaces in our system include: (1) the frontend React application communicating with the backend via REST APIs, (2) the backend interacting with Prisma/MySQL and S3 for persistence and media retrieval, (3) the backend communicating with Fal for image generation and background removal, (4) the backend communicating with Stripe for billing operations, and (5) the frontend/backend communicating with Pinterest APIs and media proxy routes.

To test these interfaces, we focused on validating the correctness of data flow between components. For example, frontend-to-backend interfaces were tested by simulating real user actions such as updating a moodboard, uploading media, or submitting a generation request and verifying that correct API responses were reflected in the UI. Backend service interfaces were tested by mocking external calls and verifying that generation responses, billing states, and media paths were handled and stored correctly.

5.3. Integration Testing

Critical integration paths were identified based on core functional requirements, particularly those involving multiple subsystems. The most critical path in the current system is the /api/generation pipeline, where the frontend sends selected board images and an optional base image, the backend uploads/prepares the inputs, calls the selected Fal model, stores the output in S3, and returns a persisted Generation record to the user.

This pipeline was considered critical because it directly supports the main feature of the platform: AI-powered interior design generation. Failures in this flow would render the core product unusable.

Integration testing was performed using a combination of Playwright/Cypress for end-to-end frontend flows and Jest/Supertest for backend route integration. Tests and manual checks cover uploading assets, saving and autosaving boards, triggering generation, verifying stored results,

enforcing plan limits, handling invalid inputs, and confirming that external-service failures return meaningful errors instead of breaking the user flow.

5.3.1 Billing/Stripe Integration Paths

Critical integration paths were tested across frontend and backend using end-to-end flows. This included (1) subscription checkout via Embedded Checkout and verifying the updated plan state, (2) enforcing "one active subscription" by upgrading/downgrading and confirming prior subscriptions are canceled as expected, (3) scheduling a plan change at period end via a subscription schedule and verifying that the scheduled change persists in the UI, (4) canceling the scheduled change and verifying the schedule is removed, (5) purchasing one-time add-ons and confirming that credits are added to the correct persistent balance, (6) consuming credits during AI generation and moodboard creation, and (7) loading invoices and invoice PDFs inside Renova via backend proxy endpoints.

5.4. System Testing

System-level testing validated the complete application against both functional and non-functional requirements. This involved running a combination of unit tests, interface tests, and integration tests to ensure the system behaved correctly as a whole.

Functional system testing ensured that major features, including authentication, moodboard creation, canvas editing, autosave, media upload/proxying, background removal, AI generation, subscription management, and add-on usage, worked together. Non-functional testing focused on responsiveness, reliability during extended editing sessions, and usability of navigation, credit messaging, and generation feedback.

Tools used included Playwright for full system end-to-end testing, along with manual exploratory testing to simulate real-world usage scenarios. These combined approaches ensured that the system met both technical and user-facing requirements.

5.4.1 Subscription + Add-on Entitlements

System-level testing focused on correctness of entitlements under different user states: free users, subscribed users, users with add-on credits, and users who have reached monthly limits. We verified that the backend is the source of truth for enforcement and that the frontend reflects the same values from /api/plans/current and /api/plans/usage. We also tested that add-on credits persist across billing cycles and that monthly subscription limits reset appropriately at the next billing period. The system was verified to correctly block actions when both subscription credits and add-on credits are exhausted.

5.5. Regression Testing

Regression testing was driven directly by project requirements to ensure that newly introduced features, particularly billing add-ons and scheduling ,did not break existing functionality.

We implemented automated test suites using Jest, Cypress, Playwright, and Supertest. Critical features protected by regression tests include authentication flows, board persistence, editor actions, generation workflows, subscription checkout, invoice retrieval, and entitlement

enforcement. Some test-runner configuration and stale expectations still require cleanup before the automated suites should be described as completely passing quality gates.

Any time a new feature was added, existing test cases were re-run to verify that prior functionality remained intact. In addition, new regression test cases were added for previously fixed bugs to ensure they did not reoccur. This continuous testing approach helped maintain system stability throughout development.

5.5.1 Protecting Existing Functionality

Regression testing ensured that introducing add-ons and scheduling did not break existing subscription checkout, saved card management, invoice listing, or existing moodboard/generation flows. We verified that non-admin users remain fully constrained by entitlements, while admin users can be exempted without affecting normal users. We also re-tested invoice viewing and billing address updates after introducing new endpoints and UI changes to ensure existing billing functionality continues to operate.

5.6. Acceptance Testing

Acceptance testing was conducted through structured demonstrations aligned with project requirements. Each demo scenario mapped directly to a functional requirement, ensuring full coverage of expected system behavior.

The client participated by observing with the system during guided walkthroughs. Feedback was collected on usability, clarity of workflows, and overall system responsiveness.

In addition to validating billing functionality (as described in your section), acceptance testing also covered core features such as moodboard creation and AI rendering. Users were able to complete tasks end-to-end, confirming that the system met both functional correctness and usability expectations.

5.6.1 Demonstrating Billing Requirements

Acceptance testing was performed by running a guided demo of the Account to Plan and Billing pages. The demo validated that users can (1) choose a plan and complete checkout inside Renova, (2) switch immediately with clear warnings and observe the old subscription being canceled, (3) schedule a plan change at period end and see the persistent banner, (4) cancel the scheduled change, (5) purchase add-ons and observe the secondary usage bar update, (6) consume add-on credits after reaching monthly limits, and (7) view and download invoices inside Renova. These acceptance steps confirmed the billing system supports both usability and correctness requirements.

5.7. Security Testing

5.7.1 Payment and Access Control

Security testing verified that Renova does not directly handle raw payment card data. Payment collection is performed through Stripe Elements/Embedded Checkout, and the backend uses Stripe API keys securely on the server side. All billing endpoints require authentication, and sensitive actions (checkout session creation, confirm endpoints, invoice PDF retrieval) require a valid user

token to prevent unauthorized access. We also validated that entitlement enforcement is performed server-side (not client-side), preventing users from bypassing limits by altering frontend state.

5.8. User Testing

User testing was conducted with peers who represented typical end users of the platform. Participants were asked to complete common tasks such as creating a moodboard, modifying elements on the canvas, generating AI designs, and managing their subscription or credits.

Observations showed that users were generally able to navigate the system without guidance, indicating strong usability. However, some users initially struggled with understanding credit usage and the distinction between subscription limits and add-on credits. Based on this feedback, we improved UI elements such as usage bars, labels, and warning messages to provide clearer guidance.

Overall, users responded positively to the interactive canvas and AI generation features, noting that the system felt responsive and intuitive. This testing confirmed that the design effectively addresses user needs while also highlighting areas for incremental improvement.

5.9. Testing Results

Testing results show that the project has meaningful automated and manual coverage across its major workflows, but they should be described realistically rather than as a perfect final gate.

Unit and Integration Tests: The repository includes frontend Jest tests, Cypress and Playwright journeys, and backend Jest/Supertest tests for controllers, middleware, helpers, and routes. Current cleanup work includes keeping Playwright specs out of Jest runs, ensuring backend test dependencies are installed consistently, and updating stale unit expectations after editor behavior changes.

Performance: AI generation latency depends on the selected external model and service queue, so the system presents generation as a long-running operation and relies on backend error handling rather than promising a fixed render time in all cases.

Reliability: Manual and automated testing exercised common failure paths such as invalid inputs, exhausted plan credits, failed uploads, and external-service errors. Remaining reliability work focuses on cleaner progress feedback, retry/caching behavior, and stricter CI enforcement.

Usability: User testing indicated that most tasks could be completed without assistance, with only minor improvements needed for clarity around billing and credits.

Overall, testing confirms that the implemented architecture supports the required workflows, while also identifying practical next steps: test-suite cleanup, CI gate tightening, and continued hardening around authentication, media access, and third-party service failures.

6. Implementation

Implementation for Renova is where the main user flow works end-to-end. The application now includes authentication, dashboard navigation, account management, editable moodboards, image upload and media proxying, Pinterest import, background removal, AI generation, generated-image galleries, plan usage enforcement, Stripe billing flows, and containerized deployment support. The implementation reflects the final architecture more than the early speculative design: the Express backend coordinates storage, authorization, billing, and AI service calls while the Next.js frontend focuses on a responsive editing experience.

6.1. Design Analysis

Renova is built as a full-stack web app with several cooperating parts:

- **Frontend (Next.js / React App Router): public pages, authentication, dashboard pages, account/plan/billing interfaces, the moodboard hub, the moodboard editor, public exploration, and generation galleries.**
- **Backend (Express API + Prisma/MySQL): authentication, user/account CRUD, moodboard CRUD, media upload/proxying, Pinterest OAuth and browsing, AI generation orchestration, background removal, Stripe billing, plan usage enforcement, admin routes, and health checks.**

S3-style object storage is used for thumbnails, uploaded board assets, avatars, background images, generation inputs, and generated outputs. The SQL database stores structured records and URLs/keys, while backend and Next.js proxy routes provide consistent and safer access to those media objects.

A big theme of the implementation is: the frontend handles the user experience, but the backend enforces the rules. Plan limits, add-on credit usage, ownership checks, admin access, invoice access, generation creation, and moodboard creation are enforced server-side so users cannot bypass them by changing client-side state.

Frontend implementation: The editor page composes BoardCanvas, Toolbar, BoardActions, Sidebar, and BoardList. BoardCanvas uses a fixed logical board size with responsive scaling, renders image/text/drawing items through React-Konva, exposes thumbnail and PNG export through a forwarded ref, and delegates shared scene state to the Zustand board store. Toolbar actions call store operations for duplicate, crop, mirror, shadow, layer order, and delete, while BoardActions controls save, image upload, export, background editing, text insertion, and drawing mode.

Canvas implementation: The board store keeps the current item list, selected item, clipboard, mode, and undo/redo stacks. Image items store position, size, rotation, z-order, crop, mirror, and shadow properties. Text items support live textarea editing over the Konva stage, automatic height measurement, wrapping, resizing, and rotation. Drawing items are stored as normalized point arrays with bounding boxes so they can be selected, transformed, cropped, and persisted like other scene items.

Persistence implementation: Moodboards are stored as Prisma records with title, dimensions, visibility, background color, thumbnail URL, and scene JSON. The editor performs a debounced

autosave for scene updates and a separate thumbnail autosave that exports the Konva stage to a JPEG. Manual save creates or updates the board and can attach pending generations to the saved moodboard.

AI implementation: AIGenView lets users select up to five board images, optionally upload a base room image, choose a model, and provide a prompt. The frontend sends multipart data to POST /api/generation. The backend validates ownership and plan limits, uploads inputs to S3, creates presigned URLs, builds the selected Fal model input, downloads the generated result, stores it in S3, and creates a Generation row linked to the user and optionally to the moodboard.

Background-removal implementation: Remove BG is implemented as an editor action for selected image items. The frontend calls POST /api/moodboard/:id/remove-bg with the selected item source. The backend checks ownership, signs the stored S3 image URL, sends it to Fal/BiRefNet, stores the returned PNG in S3, and returns a replacement image source that the frontend applies to the selected item.

Billing and entitlement implementation: Stripe customers are created or reused per user. Plan records map Stripe Price IDs to internal limits and perks. The backend reports current plan and usage, creates embedded checkout sessions for subscriptions and add-ons, confirms successful purchases, prevents duplicate add-on grants, and enforces generation and moodboard limits with monthly limits plus banked add-on balances.

Pinterest implementation: Pinterest OAuth tokens are stored in the PinterestIntegration model. Users can connect an account, browse boards and pins, and add pin images to the moodboard. The backend uses a rate-limited Pinterest helper, while the frontend routes pin images through a Next.js proxy so imported inspiration can display reliably in the canvas.

Deployment implementation: Docker Compose runs MySQL, the Express backend, and the Next.js frontend. GitLab CI/CD validates dependencies, runs lint/test jobs, builds the frontend, verifies backend health endpoints, builds Docker images with Kaniko, and deploys to the production VM through an SSH-triggered deploy script.

7. Ethics and Professional Responsibility

7.1. Areas of Professional Responsibility/Codes of Ethics

Selected Code: ACM Code of Ethics (2018)

Area of Responsibility	**Definition**	**Relevant ACM Code Item**	**How Our Team Applies It**
Public Welfare & Safety	Build tools that help users and avoid foreseeable harm	1.1 Contribute to society; 1.2 Avoid harm	Renova provides visual previews of renovation ideas before real-world purchases, helping users avoid costly mistakes.

Honesty & Transparency	Accurately represent what the system can/can't do	1.3 Be honest and trustworthy	The user interface clearly states that generated previews are approximations and not guaranteed real-world results.
Privacy & Data Protection	Protect user-uploaded photos and personal data	1.6 Respect privacy; 2.9 Design secure systems	Passwords are securely hashed before storage. Authentication uses access tokens and refresh tokens, where refresh tokens are stored as hashed session records in the database and sent as HttpOnly cookies to prevent client-side access. The backend validates sessions and rotates refresh tokens for improved security. All protected media routes require authenticated backend requests.
Fairness & Inclusion	Ensure equitable access and avoid bias	1.4 Be fair and non-discriminatory	Users can upload their own images and references instead of relying on fixed templates.
Professional Competence	Stay within areas of expertise, seek review	2.2 Maintain professional competence	The system uses established frameworks including Next.js, Express, Prisma, MySQL, AWS S3, and external AI APIs.

Our team is performing well in Privacy & Data Protection by hashing passwords, requiring authenticated access for private workflows, enforcing role checks for admin actions, and routing stored media through controlled access paths. A remaining improvement is to harden refresh-token handling by moving toward HttpOnly cookies and hashed server-side session records, and to continue refining access control for private media resources.

An area needing improvement is Fairness & Inclusion. Our current datasets and style presets may under-represent cultural diversity. To improve, we will expand our templates, audit AI outputs for bias, and incorporate manual refinement tools to enable users to correct errors as needed.

7.2. Four Principles

Broader Context	Beneficence (do good)	Nonmaleficence (avoid harm)	Respect for Autonomy	Justice (fairness)
Public health, safety, welfare	Reduces stress and financial risk by letting users preview designs virtually	Clear disclaimers so users don't mistake previews for construction-ready blueprints	Allow users to delete accounts and export data on request	Free tier access ensures that even small-budget users can benefit
Global, cultural, social	Style presets include global design traditions (e.g., Japanese minimalism, Mediterranean styles)	Avoid implying one cultural style is "default" or superior	Users select their own preferred aesthetics instead of being forced into defaults	Represent multiple cultural aesthetics equally in UI
Environmental	Helps reduce wasteful purchases, encouraging sustainable design choices	Avoid features that promote unnecessary consumption	Let users explore reuse/upcycle options when planning designs	Equal access to sustainable suggestions regardless of budget
Economic	Provides an affordable alternative to hiring designers	Ensure no hidden costs or dark patterns in pricing	Show cost per AI render clearly; allow cancellation of jobs	Transparent and consistent subscription/ pricing for all users

Our team is strongest in Economic × Beneficence, where the system directly reduces renovation costs and stress by providing affordable previews before purchase. These coefficients demonstrate a clear value to users.

An area for improvement is the intersection of Global/Cultural and Nonmaleficence. Current datasets may under-represent non-Western design traditions, which risks biasing outputs. To address this, we plan to expand the training data, introduce culturally diverse templates, and enable manual adjustments within the UI.

7.3. Virtues

Team Virtues:

- **Integrity** – Be transparent about limitations of AI outputs and costs, so users trust the system.
- **Diligence** – Carefully test subsystems, validate AI results, and review code to maintain quality.
- **Collaboration** – Share knowledge across frontend, backend, and ML, ensuring all members contribute and learn from one another.

Prakeerth Regunath

- **Demonstrated Virtue: Collaboration**
  Collaboration has been a key component of senior design so far. It is important to me because no project truly succeeds without a team that can communicate effectively and support each other when my teammates need it the most. I have demonstrated this virtue by actively participating in and engaging with group discussions, sharing resources when required, and respectfully agreeing with others.
- **Virtue Not Yet Demonstrated: Confidence**
  Confidence is an essential virtue to me because it allows many ideas to be shared clearly and taken seriously. Without confidence, even exemplary contributions to the project may go unheard, which limits the value I can bring to the team. Sometimes, I hold back my ideas or second-guess them before speaking up about them. To demonstrate greater confidence, I plan to prepare thoroughly for the weekly meetings, trust the knowledge I have gained, and voice my thoughts, even if they aren't perfect. Doing this will help me grow personally and also benefit the team with fresh ideas and perspectives.

Onur Onal

- **Demonstrated Virtue: Integrity**
  Integrity is essential to me because it builds trust and keeps the team aligned. I have demonstrated this by being clear and transparent when defining project goals and delegating tasks, and by supporting teammates with both shared work and their individual responsibilities.

- **Virtue Not Yet Demonstrated: Patience**
  Patience is essential because technical progress can be slower than anticipated, and frustration can negatively impact teamwork and collaboration.

I have not consistently demonstrated this virtue, but I can show it by giving teammates the time and flexibility to work through challenges, while offering guidance and encouragement instead of pressure.

Bae Meh

- **Demonstrated Virtue: Patience**
  Patience is essential because it allows me to approach problems calmly and thoughtfully, rather than rushing and making mistakes. I have demonstrated patience when learning new tools and concepts for the project, as well as when tasks required extra time to understand fully. Instead of getting frustrated, I've stayed composed and worked through issues step by step.
- **Virtue Not Yet Demonstrated: Leadership**
  I can contribute to this by checking in with my teammates, providing additional input on the project's direction, and helping the group stay on track.

Erroll Barker

1. **Demonstrated Virtue: Honesty**

   Honesty/Accountability
   Integrity has been crucial in my senior design work so far, as it has limited my ability to contribute during times of conflict. I was open and honest with our project leader (Onur) about my situation and took accountability for my inactivity, rather than making excuses.
2. **Virtue Not Yet Demonstrated: Responsibility**

   Responsibility
   Patience and responsibility are essential to me, and I have not demonstrated them as much as I would like. I am committed to taking on greater responsibilities and contributing more once I resolve my personal challenges related to my ongoing wedding issues.

Kemal Yavuz

- **Demonstrated Virtue: Collaboration**
  Collaboration is vital to me because it ensures that everyone contributes effectively and the team moves forward together as a cohesive unit. I have demonstrated this by actively engaging with both frontend and backend members, asking questions to clarify dependencies, and ensuring my work integrates smoothly with others. This strategy has helped keep the team aligned across subsystems.
- **Virtue Not Yet Demonstrated: Diligence/Hard Work at Full Capacity**
  Usually, I give 100% effort and push myself to deliver at my peak, but I haven't fully locked into that version of myself yet in this project. This form matters to me because consistent hard work is what ensures the team's long-term success. I plan to demonstrate it by becoming more disciplined with my weekly contributions, diving deeper into tasks, and holding myself accountable to reach the level of effort I know I can sustain.

Om Sanghvi

- **Demonstrated Virtue: Responsibility**
  Responsibility has been a key virtue for me so far in senior design. It is important to me because being accountable for both technical quality and user trust ensures the project's sustainability. I have demonstrated responsibility by ensuring that our backend and deployment choices prioritize the secure handling of user data. For example, during our meetings, I raised points about password protection and signed URL access. I also suggested a feature that would enable multiple people to collaborate on the same project board while maintaining access controls. These contributions demonstrate my approach to not only building features but also protecting the customer experience.

- **Virtue Not Yet Demonstrated: Diligence**
  Diligence is crucial to me because consistent, detail-oriented effort is what ensures that a system as complex as Renova works reliably. While I've contributed ideas and focused on security concerns, I have yet to demonstrate the whole level of diligence I expect from myself in areas such as testing, documentation, and refining deployment pipelines. I plan to demonstrate diligence by carefully reviewing backend code, validating infrastructure configurations, and putting extra effort into thorough testing. This plan will help strengthen the system's reliability and make sure our users can trust the product.

8. Conclusions

8.1. Summary of Progress

Renova has progressed from an early concept into a functioning full-stack interior design platform centered around moodboards and AI-assisted visualization. At this stage, the project includes user authentication, dashboard navigation, moodboard creation and editing, autosave and manual save behavior, direct image upload, Pinterest-based inspiration import, AI generation using selected board assets and optional base images, and account-level management features. The implemented system is now best understood as a Next.js frontend paired with an Express backend, Prisma/MySQL for structured persistence, S3 for media storage, and Seedream integration through Fal for image generation.

A major improvement since earlier design iterations is that the architecture is now grounded in what the team has actually built rather than what was initially assumed. Instead of relying on a speculative separate rendering pipeline, the current design uses a more practical backend orchestration flow that is simpler to maintain and better aligned with the project timeline. This shift improved feasibility while preserving the core value of the platform: helping users explore interior design ideas visually before making real-world decisions.

The remaining work is less about redefining the system and more about refining it. Key focus areas include strengthening automated test reliability, improving generation reliability and progress feedback, tightening CI gates, hardening authentication/media access, and polishing the user experience around billing, credits, and long-running AI tasks.

8.2. Value Provided

Renova effectively addresses the core problem identified at the beginning of the project: the difficulty users face when trying to visualize interior design ideas before committing time, money, or effort in real life. By combining moodboard-based planning with AI-assisted generation, the platform allows users to experiment with layouts, styles, and inspirations in a low-risk, interactive environment.

From a usability standpoint, the system supports both structured and exploratory workflows. Users can manually upload images, import inspiration from Pinterest, and arrange assets on a flexible canvas, which mirrors how people naturally approach design ideation. The addition of AI generation enhances this process by transforming static inspiration into more cohesive visual outputs, helping bridge the gap between concept and realization.

The value of the system is evident in user testing observations. Users were able to complete key tasks, such as creating moodboards, modifying elements, and generating designs, without extensive guidance, indicating that the platform is intuitive and aligns well with user expectations. Additionally, feedback highlighted that the ability to iterate quickly on design ideas was particularly useful, reinforcing the platform's role as a planning and experimentation tool.

In a broader context, Renova fits into the growing space of AI-assisted creative tools, similar to platforms that support design prototyping or content generation. However, it differentiates itself by focusing specifically on interior design workflows and integrating multiple stages, collection, organization, and generation, into a single cohesive system. This integration reduces friction compared to using multiple disconnected tools.

Even in cases where AI generation may vary in quality or speed, the platform still provides meaningful value through its moodboard and organization features. This ensures that the system remains useful regardless of external service variability, making the design both practical and resilient.

8.3. Next Steps

While the current implementation provides a strong foundation, there are several important next steps that could further enhance the system.

One key area is improving the reliability and performance of AI generation. This includes optimizing request handling, introducing better progress indicators for long-running operations, and potentially caching or reusing previous results to reduce redundant computation. These improvements would directly enhance user experience by making the system feel faster and more responsive.

Another important direction is expanding collaboration features. Currently, moodboards are primarily single-user experiences. Future work could introduce shared boards, real-time collaboration, and commenting functionality, allowing multiple users to work together on design projects. This would significantly broaden the platform's applicability, especially for teams or client-designer interactions.

Enhancing personalization and recommendation systems is also a valuable next step. By analyzing user preferences, past moodboards, and interaction patterns, the system could suggest relevant

design elements, layouts, or styles. This would make the platform more intelligent and reduce the effort required for users to get started.

From a technical perspective, further strengthening security and scalability would be important for production readiness. This includes hardening refresh-token storage, refining access control for media resources, improving rate limiting and abuse prevention, tightening CI test gates, and ensuring the system can handle increased user load.

Finally, a natural follow-up project would be exploring deeper integration with real-world applications, such as linking designs to purchasable furniture or materials. This would extend the platform from a planning tool into a more end-to-end solution, connecting inspiration directly to execution.

Overall, these next steps build on the current system's strengths while addressing its limitations, and they represent realistic and impactful directions for future development.

9. References

The following technical references support the implemented design:

https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf[1] Vercel, "Next.js Documentation," https://nextjs.org/docs.

[2] Meta Platforms, "React Documentation," https://react.dev/.

[3] Konva, "Konva and React-Konva Documentation," https://konvajs.org/docs/react/.

[4] Prisma, "Prisma ORM Documentation," https://www.prisma.io/docs.

[5] Express.js, "Express Documentation," https://expressjs.com/.

[6] Stripe, "Stripe API and Checkout Documentation," https://docs.stripe.com/.

[7] Amazon Web Services, "Amazon S3 Documentation," https://docs.aws.amazon.com/s3/.

[8] Fal.ai, "Fal API Documentation," https://fal.ai/docs.

[9] Pinterest Developers, "Pinterest API Documentation," https://developers.pinterest.com/docs/.

[10] GitLab, "GitLab CI/CD Documentation," https://docs.gitlab.com/ee/ci/.

10. Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics/layout, PCB testing issues, software bugs, etc.

10.1. Appendix 1 – Operation Manual

- At least 2-3 pages
- Step-by-step instructions on how to setup/demo/test/use the system

10.1.1 Operation Manual (Stripe Setup + Demo Script)

To run Renova for demo or development, configure backend environment variables for DATABASE_URL, JWT_SECRET, Google OAuth, AWS/S3, FAL_KEY, Pinterest OAuth, Stripe secret key, and APP_URL. Configure frontend variables such as NEXT_PUBLIC_API_BASE_URL, NEXT_PUBLIC_GOOGLE_CLIENT_ID, and the Stripe publishable key when billing features are used. After pulling the repository, install dependencies in both backend and frontend, run Prisma generate/migrate, and seed preset moodboards and plan records. The project can be started locally with the backend on port 8080 and the frontend on port 3000, or through Docker Compose, which runs MySQL, the backend, and the frontend together. A demo flow should include: register or log in, create or clone a preset moodboard, upload an image, add text or drawing items, crop/mirror/duplicate/layer items, remove an image background, save and autosave the board, select items in the AI panel, choose a model, generate an image, view the result in the Generations panel, and then walk through plan/add-on checkout and invoice viewing with Stripe test cards.

10.2. Appendix 2 – Alternative/initial version of design

- Versions considered before client's specifications have changed
- Versions considered before learning more about the project
- Versions that resulted in failure to achieve specifications, etc.
- Describe each major design version and why they were scrapped/revised

10.3. Appendix 3 – Team Contract

Team Contract

Team Members:

1) Prakeerth Regunath 2) Onur Onal

3) Kemal Yavuz 4) Bae Meh

5) Erroll Barker 6) Om Sanghvi

Team Procedures

1) Day, time, and location (face-to-face or virtual) for regular team meetings:
    a) Day of the week: Tuesday
    b) Time: 5 - 6 PM
    c) Location: Mainly in-person. If there are any issues, we will hold a virtual meeting (via Discord), depending on availability.
2) Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    a) Primary: Discord (Group Chat + Meeting calls through phone/computer)
    b) Secondary: Email (File-Sharing)
3) Decision-making policy (e.g., consensus, majority vote):
    a. Majority vote when possible; otherwise, Consensus.
4) Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    a. One rotating member will take meeting minutes each session.
    b. Minutes will then be shared in our dedicated Discord group server

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    a. Attend all scheduled meetings unless excused in advance.
    b. Arrive on time and stay engaged during discussions.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    a. Each member is responsible for completing assigned work before deadlines.
    b. If issues arise, notify the team early so adjustments can be made.
3. Expected level of communication with other team members:
    a. Frequent and transparent updates through Discord.
    b. Respond to messages within 24 hours.
4. Expected level of commitment to team decisions and tasks:
    a. Support group decisions once made.
    b. Commit to high-quality work and follow through on tasks.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
    a. Prakeerth Regunath - Frontend development lead, responsible for UI features, dashboard components, and coordination of frontend tasks.
    b. Bae Meh - Backend development support, responsible for API logic, database operations, and AI generation integration.
    c. Kemal Yavuz - Backend development support, responsible for API logic, database operations, and AI generation integration.
    d. Onur Onal - Project lead and architecture lead, responsible for overall system design, technical decisions, and full-stack development.
    e. Erroll Barker - Frontend / UX lead, responsible for layout, styling, and user experience consistency across the application.
    f. Om Sanghvi - Infrastructure / CI-CD / backend support, responsible for deployment, testing pipeline, and environment configuration.
2. Strategies for supporting and guiding the work of all team members:
    a. Pair programming or peer review for complex tasks.
    b. Offer help when someone falls behind.
    c. Share resources, tutorials, and references to build up weaker skills.
    d. Encourage open discussions to ask for clarification without judgment.
3. Strategies for recognizing the contributions of all team members:
    a. Acknowledge contributions during meetings.
    b. Post shoutouts in Discord for completed milestones.

c. Celebrate major project milestones together (e.g., small team social).

Collaboration

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
    a. Prakeerth Regunath - Strong coding background and project management skills.
    b. Bae Meh - Clear communicator, strong organizational abilities, effective at bridging ideas.
    c. Erroll Barker - Detail-oriented, excels at documentation and tracking progress.
    d. Onur Onal - Experience in frontend and backend development through internships. Came up with the project idea.
    e. Kemal Yavuz - Skilled in system design and architecture, bringing a technical vision.
    f. Om Sanghvi - Strong developer, skilled in integration and code quality assurance.
2. Strategies for encouraging and supporting contributions and ideas from all team members:
    a. Encourage brainstorming without immediate criticism.
    b. Use Discord polls for anonymous input on conflicting decisions.
    c. Respect different working styles and schedules.
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
    a. Team members should first raise the concern privately with the meeting organizer.
    b. If unresolved, bring it up in a group meeting for open discussion.
    c. If the issue remains unresolved, escalate it to the instructor/TA for mediation.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    a. Deliver a high-quality project on time by meeting the requirements, writing clean and maintainable code, and incorporating user feedback where possible.
    b. Maintain consistent communication and collaboration to ensure transparency and alignment on project progress.
    c. Ensure all team members understand and contribute to each part of the project so knowledge is shared and the workload is balanced.

2. Strategies for planning and assigning individual and team work:
    a. Break large tasks into smaller pieces.
    b. Assign based on individual strengths while rotating to build skills.
    c. Use shared tools (Google Drive/Trello/Discord) for tracking progress.
3. Strategies for keeping on task:
    a. Weekly check-ins on progress.
    b. Clear deadlines and reminders in Discord.
    c. Adjust workload distribution if someone falls behind.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
    a. First offense: Private reminder and discussion.
    b. Second offense: Group discussion and redistribution of tasks if needed.
    c. Third offense: Report to instructor/TA.
2. What will your team do if the infractions continue?
    a. The team will document the issue and request intervention from course staff.

****

a. I participated in formulating the standards, roles, and procedures as stated in this contract.
b. I understand that I am obligated to abide by these terms and conditions.

c. I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Prakeerth Regunath DATE 5/4/2026
2) Bae Meh DATE 5/4/2026
3) Kemal Yavuz DATE 5/4/2026
4) Om Sanghvi DATE 5/4/2026
5) Onur Onal DATE 5/4/2026
6) Erroll Barker DATE 5/4/2026