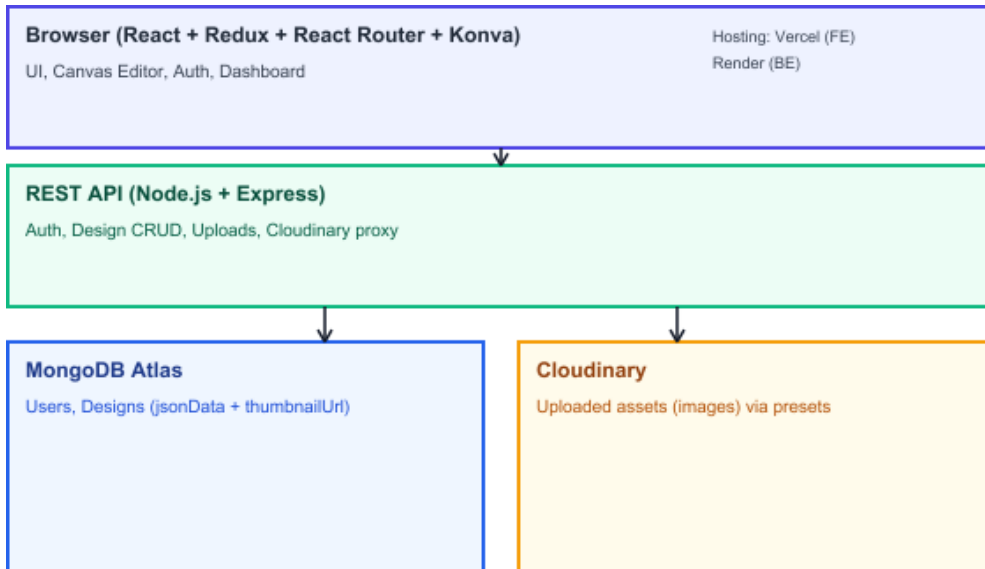


# Final Project Report

## Matty – Online Graphic Design Tool

<b>Client</b>	Confidential
<b>Prepared by</b>	Global Next Consulting India Pvt. Ltd. (GNCIPL)
<b>Prepared on</b>	12 August 2025
<b>Product Owner</b>	Ravi Kant (GNCIPL)
<b>Dev Team</b>	Shivanshu Kumar Gupta (Lead Dev), Hemlata (Lead Dev), Chitra Gautam (UI/UX Designer), Shivam Patel (Tester), Tasleem Ahmad (Deployment Engineer)



# Table of Contents

1	Certificate & Declarations
2	Abstract & Introduction
3	Problem Statement & Objectives
4	Background (Literature Review)
5	Architecture & Technologies
6	Modules / Features
7	Database Design (ERD & Schema)
8	Frontend & Backend Overview
9	Implementation Details
10	Test Cases & Results
11	Deployment Process
12	Limitations & Future Enhancements
13	Conclusion & References
14	Appendix: Screenshots

# Certificate by Company

This is to certify that the project titled “Matty – Online Graphic Design Tool” has been carried out by the team under the guidance of GNCIPL during the stated period. The report is a bona fide record of original work.

## Acknowledgment

We thank the Product Owner, Ravi Kant, for domain guidance and timely clarifications. We thank our Lead Developers, **Shivanshu Kumar Gupta (Team Lead)** and **Hemlata**, for their leadership and engineering contributions, with special appreciation to **Shivanshu** for leading the team. We acknowledge UI/UX inputs from **Chitra Gautam**, deployment support from **Tasleem Ahmad**, and QA diligence by **Shivam Patel**. Special thanks to stakeholders and early pilot users whose feedback shaped the MVP.

## Declaration

We declare that this submission is our own effort. All external libraries and assets have been used in compliance with their licenses.

## Abstract

Matty is a web-based canvas editor that allows non-designers to create posters, banners, and social media creatives using drag and drop, templates, and exports. The MVP targets single-user workflows with JWT authentication, design persistence in MongoDB (jsonData), and Cloudinary-backed uploads for images.

## Introduction

The project addresses the gap between heavy professional design tools and quick lightweight editors needed by marketing teams and students. Built using the MERN stack, Matty emphasizes speed, zero-install access, and graceful UX on both desktop and mobile.

## Problem Statement

Users need a simple browser-based editor to compose layered designs, re-edit later, and export as PNG/PDF—without complex learning curves or expensive licenses.

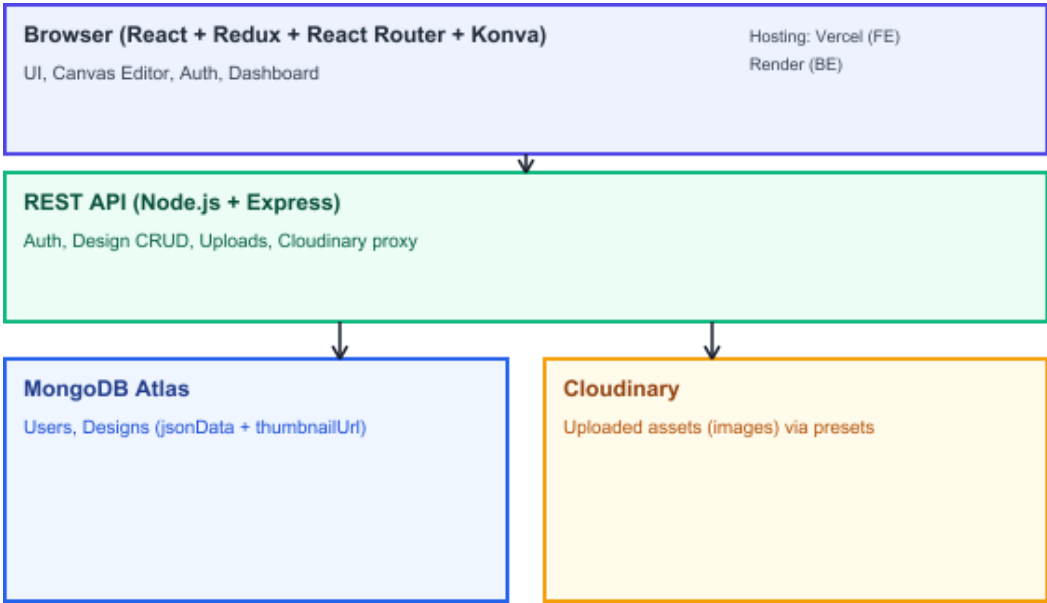
## Objectives

- Deliver a MVP canvas editor with drag-drop shapes, images, and text
- Enable auth, design CRUD (save/load/update/delete) tied to user accounts
- Generate thumbnails and exports; persist full design structure as Konva JSON
- Ensure scalable hosting on Vercel (FE) and Render (BE) with MongoDB Atlas

## Literature Review / Background Study

We reviewed modern web editors (e.g., Canva-lite patterns) and open-source canvas libraries (Konva, Fabric). Konva’s retained-mode scene graph, node transforms, and serialization via toJSON/toDataURL aligned well with our persistence and thumbnail requirements. For uploads, Cloudinary provides secure, CDN-backed storage and transformations.

# System Architecture



Request Flow: React SPA → Express API (JWT middleware) → MongoDB for documents; Cloudinary for image assets. Exports are generated client-side from Konva’s Stage toDataURL and saved as thumbnails for quick rendering in dashboard.

## Tools and Technologies Used

Frontend	React, Redux, React Router, TailwindCSS, react-konva
Backend	Node.js, Express.js
Database	MongoDB Atlas
Auth	JWT (Admin OAuth planned)
Storage	Cloudinary
Hosting	Vercel (Frontend), Render (Backend)

## Modules / Features

- Authentication (register/login) with protected routes
- Design Editor: add/move/resize/rotate shapes, images, text; text styling; undo/redo
- Uploads: user images via Cloudinary; public\_id stored for later deletion
- Persistence: Konva Stage JSON stored as jsonData with thumbnailUrl
- Dashboard: grid of thumbnails, edit/delete actions
- Exports: client-side thumbnail generation; PDF planned

## Frontend & Backend Overview

**Frontend:** Modular pages (Home, Login, Register, Dashboard, Editor), global auth state via Redux, protected routes, Konva-based canvas with custom Draggable components.

**Backend:** Express routes for auth and designs, JWT middleware, Mongoose schemas (User, Design), uploads controller proxying to Cloudinary, and cleanup on design deletion.

## Implementation Highlights

- Canvas serialization with `Stage.toJSON()` and thumbnails with `Stage.toDataURL()`
- Design CRUD scoped by `userId`; optimistic UI updates on delete
- Cloudinary delete-by-public\_id when image elements are removed
- Undo/Redo via snapshots of elements array (history stack)

## Database Design (ERD & Schema)

Entities: User( `_id`, name, email, passwordHash, role ), Design( `_id`, `userId`, title, `jsonData`, `thumbnailUrl`, `createdAt` ). Each Design.jsonData holds the serialized Konva scene graph, including per-node attrs (e.g., `id`, `x`, `y`, `width`, `height`, `text`, `fontSize`, `fill`, `rotation`, `url`, `public_id`).

## Test Cases & Results

Case	Endpoint	Expected	Result
Auth – valid login	POST /api/auth/login	JWT + user returned	Pass
Auth – invalid login	POST /api/auth/login	401	Pass
Design create	POST /api/designs	201 + saved id	Pass
Design fetch	GET /api/designs	200 + list	Pass
Upload image	POST /api/uploads/image	200 + url + public_id	Pass
Delete design	DELETE /api/designs/:id	200 + image cleanup	Pass

## Deployment Process

Frontend deployed to Vercel with environment variable `VITE_REACT_APP_API_URL`; Backend deployed to Render with Mongo Atlas URI, JWT secret, and Cloudinary credentials. Branch protections on GitHub enforce PR-based merges and CI checks.

## Limitations

No real-time collaboration in MVP; limited template library; PDF export not server-side; no granular roles beyond user/manager.

## Future Enhancements

- Multi-user live editing with Socket.io
- Templating marketplace and premium assets
- Server-side PDF rasterization
- Version history with diffs

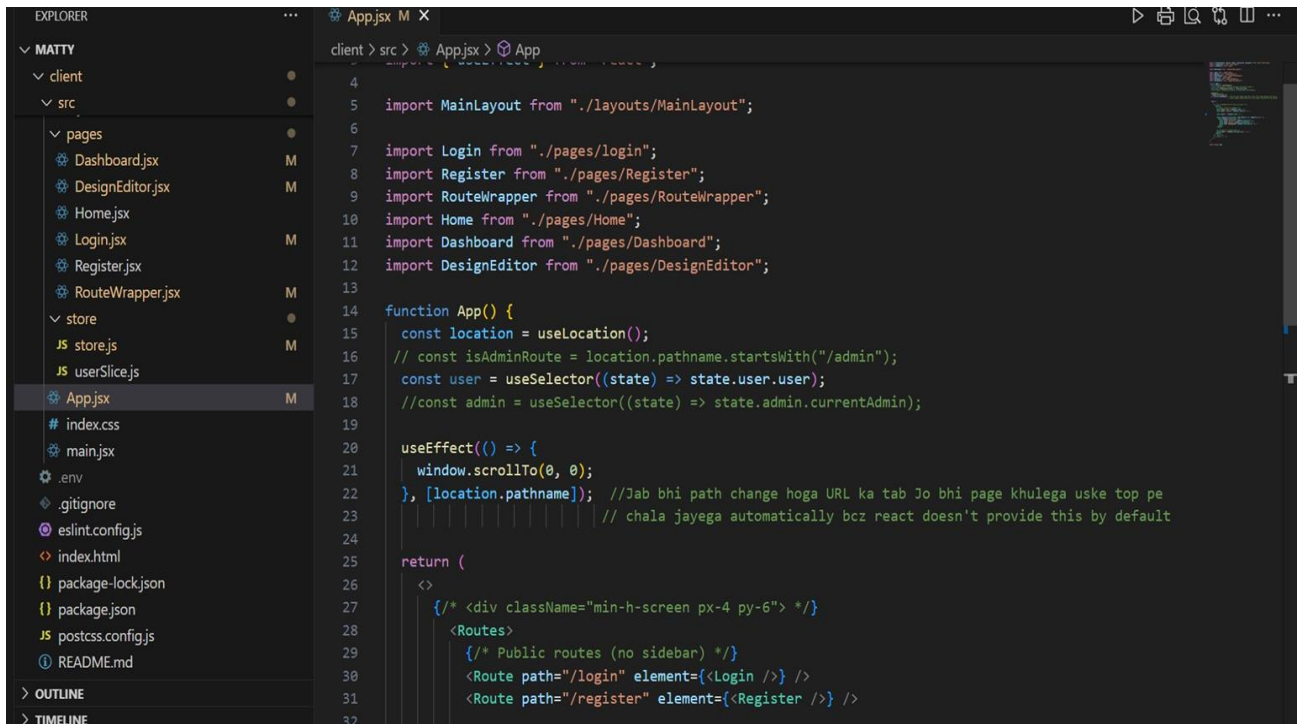
# Conclusion

Matty delivers a practical, performant MVP that enables quick graphic creation in the browser. The architecture scales horizontally and the data model supports rich editing and rehydration.

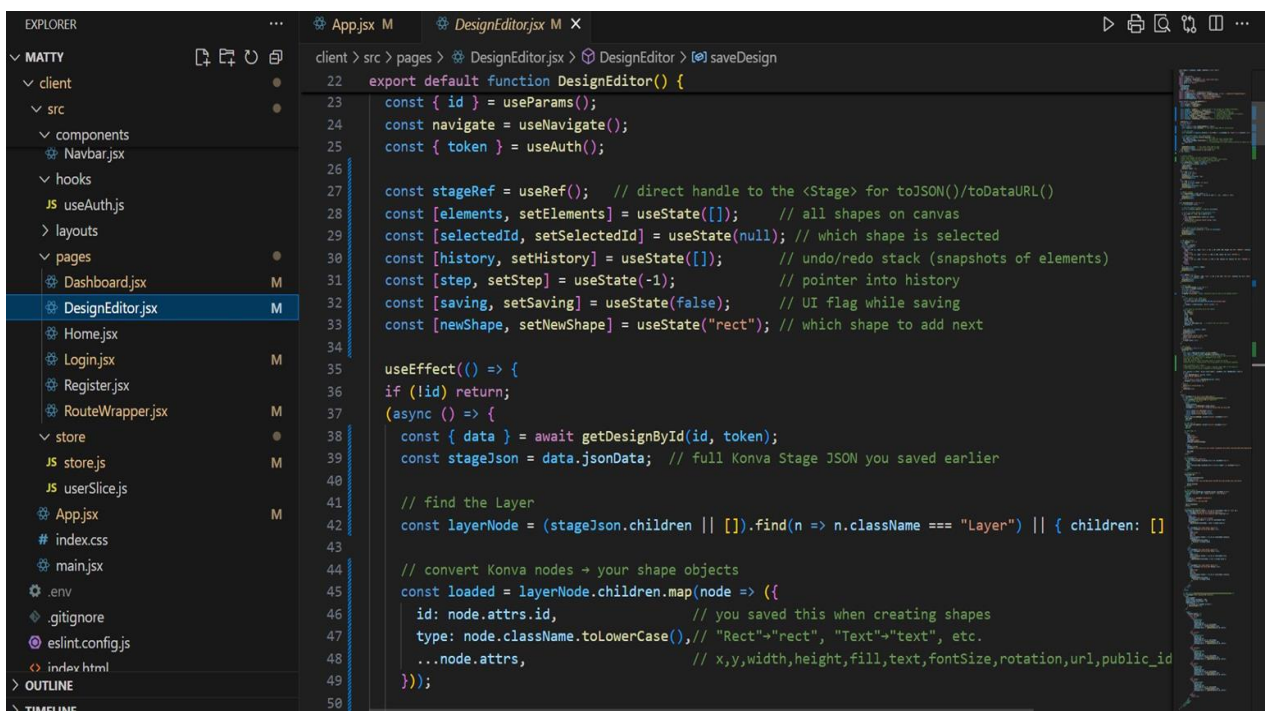
# References

Konva.js docs; React, Redux, Express, MongoDB official docs; Cloudinary developer guides; Vercel and Render deployment docs.

# Appendix: Screenshots



```
client > src > App.jsx > App
4
5 import MainLayout from "../layouts/MainLayout";
6
7 import Login from "../pages/login";
8 import Register from "../pages/Register";
9 import RouteWrapper from "../pages/RouteWrapper";
10 import Home from "../pages/Home";
11 import Dashboard from "../pages/Dashboard";
12 import DesignEditor from "../pages/DesignEditor";
13
14 function App() {
15   const location = useLocation();
16   // const isAdminRoute = location.pathname.startsWith("/admin");
17   const user = useSelector((state) => state.user.user);
18   //const admin = useSelector((state) => state.admin.currentAdmin);
19
20   useEffect(() => {
21     window.scrollTo(0, 0);
22   }, [location.pathname]); //Jab bhi path change hoga URL ka tab Jo bhi page khulega uske top pe
23                               // chala jayega automatically bec react doesn't provide this by default
24
25   return (
26     <>
27     <div className="min-h-screen px-4 py-6"> */>
28     <Routes>
29       <Route path="/login" element={<Login /> } />
30       <Route path="/register" element={<Register /> } />
31     </Routes>
32   )
33 }
```



```
client > src > pages > DesignEditor.jsx > DesignEditor > saveDesign
22 export default function DesignEditor() {
23   const { id } = useParams();
24   const navigate = useNavigate();
25   const { token } = useAuth();
26
27   const stageRef = useRef(); // direct handle to the <Stage> for toJSON()/toDataURL()
28   const [elements, setElements] = useState([]); // all shapes on canvas
29   const [selectedId, setSelectedId] = useState(null); // which shape is selected
30   const [history, setHistory] = useState([]); // undo/redo stack (snapshots of elements)
31   const [step, setStep] = useState(-1); // pointer into history
32   const [saving, setSaving] = useState(false); // UI flag while saving
33   const [newShape, setNewShape] = useState("rect"); // which shape to add next
34
35   useEffect(() => {
36     if (!id) return;
37     (async () => {
38       const { data } = await getDesignById(id, token);
39       const stageJson = data.jsonData; // full Konva Stage JSON you saved earlier
40
41       // find the Layer
42       const layerNode = (stageJson.children || []).find(n => n.className === "Layer") || { children: [] }
43
44       // convert Konva nodes to your shape objects
45       const loaded = layerNode.children.map(node => ({
46         id: node.attrs.id, // you saved this when creating shapes
47         type: node.className.toLowerCase(), // "Rect"-"rect", "Text"-"text", etc.
48         ...node.attrs, // x,y,width,height,fill,text,fontSize,rotation,url,public_id
49       }));
50     });
51   });
52 }
```

EXPLORER

MATTY

- client
  - src
    - components
      - Navbar.jsx
    - hooks
      - useAuth.js
    - layouts
    - pages
      - Dashboard.jsx
      - DesignEditor.jsx
      - Home.jsx
      - Login.jsx
      - Register.jsx
      - RouteWrapper.jsx
    - store
      - store.js
      - userSlice.js
    - App.jsx
    - index.css
    - main.jsx
    - .env
    - .gitignore
    - eslint.config.js
    - index.html

OUTLINE

TIMELINE

App.jsx M DesignEditor.jsx M X

client > src > pages > DesignEditor.jsx > DesignEditor > saveDesign

```
22 export default function DesignEditor() {
215   <select
216     value={newShape}
217     onChange={e => setNewShape(e.target.value)}
218     className="w-full px-3 py-2 rounded-md bg-gray-700 text-gray-200"
219   >
220     <option value="rect">Rectangle</option>
221     <option value="circle">Circle</option>
222     <option value="ellipse">Ellipse</option>
223   </select>
224   <Button onClick={addShape} variant="outline" className="w-full">
225     Add Shape
226   </Button>
227
228   { /* Add Text */ }
229   <Button onClick={addText} variant="outline" className="w-full">
230     Add Text
231   </Button>
232
233   { /* Add Image */ }
234   <div>
235     <input
236       type="file"
237       accept="image/*"
238       id="upload"
239       className="hidden"
240       onChange={handleFileChange}
241     />
242     <label
```

EXPLORER

MATTY

- client
  - src
    - components
      - Navbar.jsx
    - hooks
      - useAuth.js
    - layouts
    - pages
      - Dashboard.jsx
      - DesignEditor.jsx
      - Home.jsx
      - Login.jsx
      - Register.jsx
      - RouteWrapper.jsx
    - store
      - store.js
      - userSlice.js
    - App.jsx
    - index.css
    - main.jsx
    - .env
    - .gitignore
    - eslint.config.js
    - index.html

OUTLINE

TIMELINE

App.jsx M DesignEditor.jsx M X

client > src > pages > DesignEditor.jsx > DesignEditor > saveDesign

```
22 export default function DesignEditor() {
112   const addShape = () => {
113     // ...
114     const id_ = nanoid();
115     switch (newShape) {
116       case "rect":
117         shape = { id: id_, type: "rect", x: 50, y: 50, width: 100, height: 60, fill: "#00D2FF", rotati
118         break;
119       case "circle":
120         shape = { id: id_, type: "circle", x: 100, y: 100, radius: 50, fill: "#FF7F50" };
121         break;
122       case "ellipse":
123         shape = { id: id_, type: "ellipse", x: 150, y: 150, radiusX: 70, radiusY: 40, fill: "#8A2BE2"
124         break;
125       default:
126         return;
127     }
128     const next = [...elements, shape];
129     setElements(next);
130     pushHistory(next);
131   };
132
133   const addText = () => {
134     const txt = { id: nanoid(), type: "text", x: 50, y: 50, text: "New Text", fontSize: 20, fill: "#33
135     const next = [...elements, txt];
136     setElements(next);
137     pushHistory(next);
138   };
139
140   // Upload image
```



EXPLORER

MATTY

- client
  - postcss.config.js
  - README.md
  - tailwind.config.js
  - vite.config.js
- server
  - node\_modules
  - src
    - controllers
      - adminAuth.js
      - authController.js
      - designController.js
      - uploadController.js
    - database
    - middleware
    - models
      - Admin.js
      - BlacklistedToken.js
      - Design.js
      - User.js
    - routes
    - utils
    - app.js
  - env
  - OUTLINE
  - TIMELINE

App.jsx M JS designController.js M X

```
server > src > controllers > JS designController.js > deleteDesign
24 export const createDesign = async (req, res, next) => {
25   const { title, jsonData, thumbnailUrl = '' } = req.body;
26   if (!title || !jsonData) {
27     return next(createError(400, 'Title and jsonData are required'));
28   }
29   const design = await Design.create({
30     userId: req.userId,
31     title,
32     jsonData,
33     thumbnailUrl
34   });
35   res.status(201).json(design);
36 } catch (err) {
37   next(err);
38 }
39 };
40
41 /**
42  * GET /api/designs/:id
43  * Fetch a single design by its ID, but only if it belongs to the current user.
44  */
45 export const getDesignById = async (req, res, next) => {
46   try {
47     const { id } = req.params;
48     // Find the design that matches both the ID and the current user
49     //console.log(id, " ", "User id:", req.userId);
50     const design = await Design.findOne({ _id: id, userId: req.userId });
51     if (!design) {
52       return next(createError(404, 'Design not found'));
53     }
```

EXPLORER

MATTY

- client
  - postcss.config.js
  - README.md
  - tailwind.config.js
  - vite.config.js
- server
  - node\_modules
  - src
    - controllers
      - adminAuth.js
      - authController.js
      - designController.js
      - uploadController.js
    - database
    - middleware
    - models
      - Admin.js
      - BlacklistedToken.js
      - Design.js
      - User.js
    - routes
    - utils
    - app.js
  - env
  - OUTLINE
  - TIMELINE

App.jsx M JS User.js X

```
server > src > models > JS User.js > comparePassword
1 import mongoose from 'mongoose';
2 import bcrypt from 'bcryptjs';
3
4 const userSchema = new mongoose.Schema({
5   name: { type: String, required: true },
6   email: { type: String, required: true, unique: true },
7   password: { type: String, required: true },
8   role: { type: String, enum: ['user', 'manager'], default: 'user' },
9 }, { timestamps: true });
10
11 // password hashing
12 userSchema.pre('save', async function(next) {
13   if (this.isModified('password')) {
14     const salt = await bcrypt.genSalt(10);
15     this.password = await bcrypt.hash(this.password, salt);
16   }
17   next();
18 });
19 userSchema.methods.comparePassword = function(candidate) {
20   return bcrypt.compare(candidate, this.password);
21 };
22
23 export default mongoose.model('User', userSchema);
24
```

EXPLORER

- MATTY
  - client
    - postcss.config.js
    - README.md
    - tailwind.config.js
    - vite.config.js
  - server
    - node\_modules
    - src
      - controllers
        - adminAuth.js
        - authController.js
        - designController.js
        - uploadController.js
      - database
      - middleware
      - models
        - Admin.js
        - BlacklistedToken.js
        - Design.js
        - User.js
      - routes
      - utils
      - app.js
    - .env
  - OUTLINE
  - TIMELINE

server > src > models > JS Design.js > designSchema > thumbnailUrl

```
1 import mongoose from 'mongoose';
2
3 const designSchema = new mongoose.Schema(
4   {
5     userId: {
6       type: mongoose.Schema.Types.ObjectId,
7       ref: 'User',
8       required: true,
9     },
10    title: {
11      type: String,
12      required: true,
13      trim: true,
14    },
15    thumbnailUrl: {
16      type: String,
17      default: '',
18      trim: true,
19    },
20    jsonData: {
21      type: Object,
22      required: true,
23    },
24  },
25  { timestamps: true }
26 );
27
28 export default mongoose.model('Design', designSchema);
29
```

EXPLORER

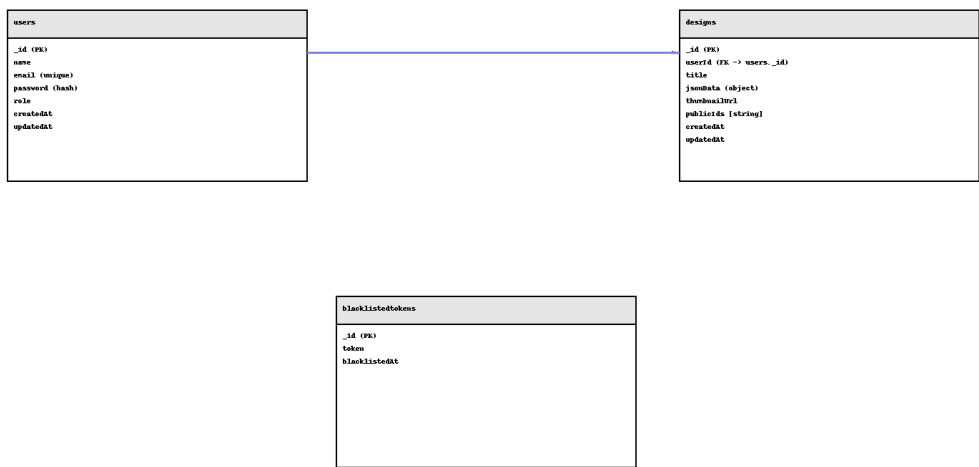
- MATTY
  - server
    - src
      - controllers
        - authController.js
        - designController.js
        - uploadController.js
      - database
      - middleware
      - models
        - Admin.js
        - BlacklistedToken.js
        - Design.js
        - User.js
      - routes
        - adminAuth.js
        - auth.js
        - design.js
        - uploads.js
      - utils
      - app.js
    - .env
    - package-lock.json
    - package.json
  - OUTLINE
  - TIMELINE

server > src > JS app.js > ...

```
27 // Auth routes
28 app.use('/api/auth', authRoutes);
29
30 // Design CRUD routes (protected inside router)
31 app.use('/api/designs', designRoutes);
32
33 app.use('/api/uploads', uploadsRoutes);
34
35 // Health-check
36 app.get('/', (req, res) => {
37   res.json({ activeStatus: true, error: false });
38 });
39
40 // — ERROR HANDLER —
41 app.use((err, req, res, next) => {
42   console.error("Error:", err.message);
43   res.status(err.status || 500).json({ message: err.message });
44 });
45
46 // — DATABASE CONNECTION —
47 main().catch(err => console.error('DB connection failed', err));
48
49 // — SERVER STARTUP —
50 if (process.env.NODE_ENV !== 'production') {
51   const PORT = process.env.PORT || 4000;
52   app.listen(PORT, () => {
53     console.log("Server running on http://localhost:${PORT}")
54   });
55 }
```



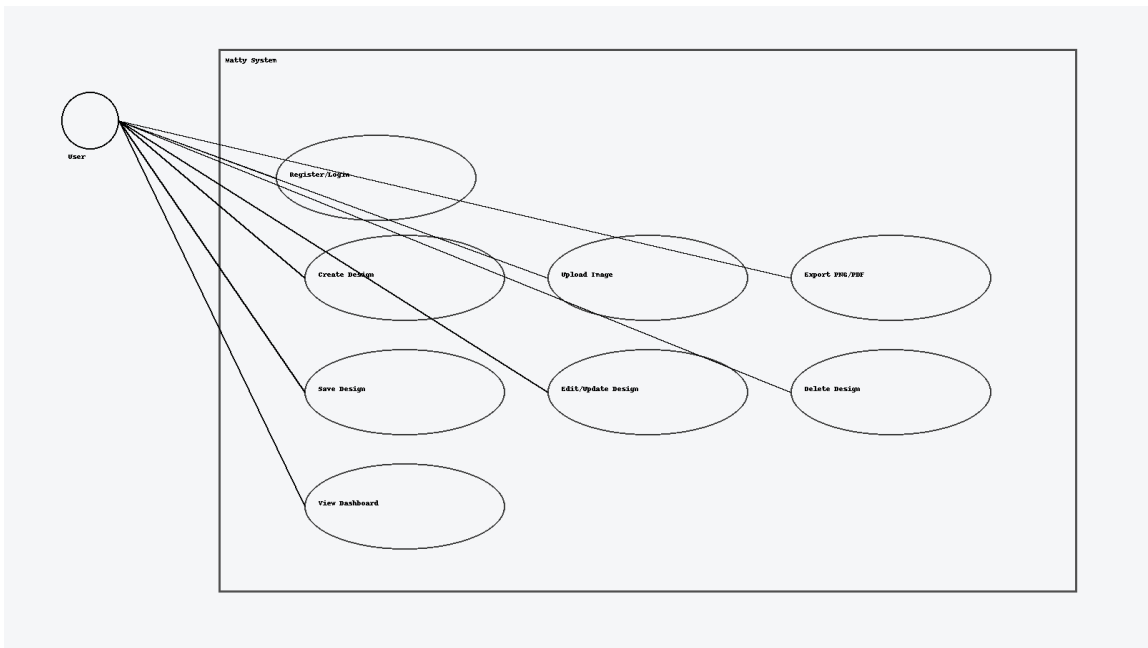
## 4. Data Model & ER Diagram



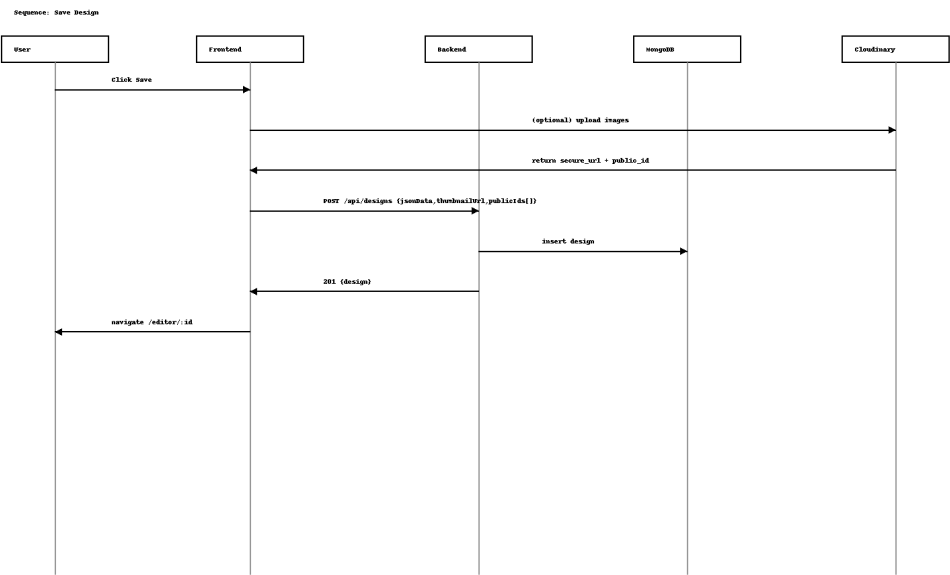
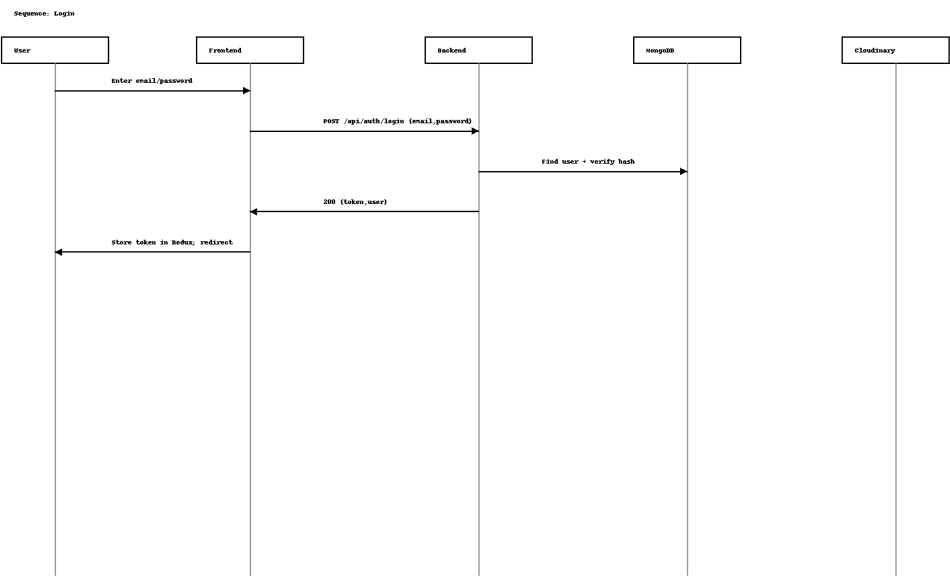
## 5. Data Dictionary

See Data\_Model/Data\_Dictionary.xlsx for all collections, fields, types, and semantics.

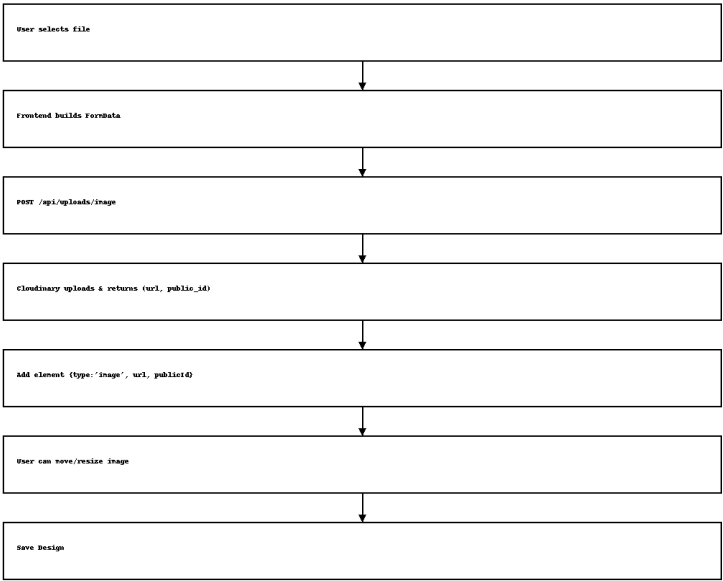
## 6. UML: Use Case



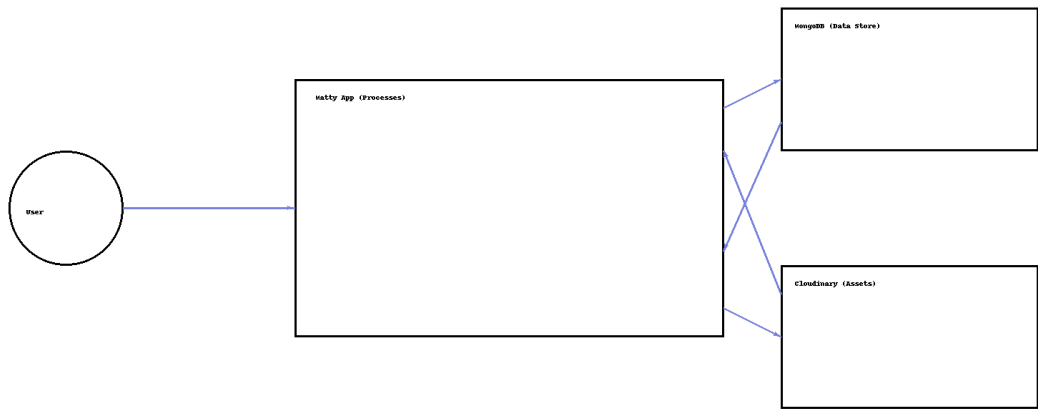
# 7. UML: Sequence (Login / Save Design)



# 8. Activity Diagram (Upload Image)



9. Data Flow Diagram (Level 0)



## 10. API Contracts

Endpoints grouped under /api/auth, /api/designs, /api/uploads. See System\_Design/API/API\_Contracts.md for full request/response schemas and auth requirements.

## 11. State Management

- Redux slice: user { token, user }
- Selectors: useAuth() returns { token, user, isLoggedIn }
- Axios instance attaches Authorization header dynamically

## 12. Security Design

- JWT signed with SERVER JWT\_SECRET; 1d expiry
- Auth middleware verifies token; userId injected into req
- CORS restricts origins to FRONTEND\_URL
- Cloudinary uses signed API keys on server; publicId-only exposed
- Server-side validation on payloads; limit file size & types

## 13. Performance & Reliability

- Thumbnails stored as data URLs reduce initial latency from Cloudinary
- Konva stage uses minimal layers; avoid unnecessary re-renders
- Indexes on users.email and designs.userId improve queries
- Graceful error handler returns JSON {message}; logs server-side

## 14. Deployment Topology

- Frontend: Vercel build (Vite) with env var VITE\_REACT\_APP\_API\_URL
- Backend: Render, Node 18+, env vars (JWT\_SECRET, MONGO\_URI, CLOUDINARY\_\*)
- DB: MongoDB Atlas (shared cluster)
- DNS: Custom domain optional; HTTPS enforced