**Whiteboard**

**Contents**

1. Project snapshot & goals
2. Quick start (dev)
3. Architecture overview
4. Frontend
   * Tech stack
   * Folder map
   * Routing & auth flow
   * Key components & whiteboard flow
   * State management & networking
   * Styling & assets
   * Dev commands
   * Known front-end limitations
5. Backend
   * Tech stack
   * Folder map
   * Environment variables
   * Data models
   * Authentication flow
   * API endpoints (reference)
   * File uploads
   * Real-time (Socket.IO)
   * Security hardening
   * Dev & deployment notes
6. Socket.IO event contract
7. Production checklist
8. Testing plan
9. Troubleshooting FAQ
10. Changelog
11. Appendix

**1 — Project snapshot & goals**

**Goal:** Real-time collaborative digital whiteboard with user accounts, avatar & image storage, and live drawing rooms. The presenter broadcasts quickly-generated PNG frames and viewers receive a lightweight live stream. The product is built as a two-folder repo (backend/ and frontend/) and can be deployed together behind a reverse proxy.

**High-level tech:** Node 18 (Express) + MongoDB + Socket.IO for realtime; React 18 + Vite + Rough.js for drawing; Tailwind CSS for styling; Multer for uploads; JWT (http-only cookie) for auth.

**2 — Quick start (development)**

Prerequisites: Node 18+, npm, MongoDB running (or Atlas URI)

# clone repo

git clone <repo>

# backend

cd backend

npm install

npm run dev # starts server on PORT 5000 (default)

# frontend

cd ../frontend

npm install

npm run dev # starts Vite dev server on port 5173

# open http://localhost:5173

Notes: CORS on backend is configured to allow the dev origin http://localhost:5173. Axios instances in the frontend use withCredentials:true to include the http-only cookie.

**3 — Architecture overview**

* **Frontend**: React + Vite with a single-page app. The whiteboard uses Rough.js to draw shapes and canvas is serialized to PNG frames which are broadcast via Socket.IO to room participants.
* **Backend**: Express server serves REST APIs (/api/\*) and serves static files from /img. Socket.IO runs on the same HTTP server and mediates room joins and broadcast frames. MongoDB stores users and references to uploaded filenames.
* **Auth**: Email + password registration. Passwords are bcrypt-hashed. Login returns a JWT set as an http-only cookie. Protected REST routes read the cookie.

**4 — Frontend**

**4.1 Tech stack**

* Vite 5
* React 18 (functional components + hooks)
* React Router DOM v6
* Tailwind CSS 4 (via official @tailwindcss/vite plugin)
* Axios (global instance with withCredentials:true)
* Socket.IO-client 4
* Rough.js (rough-esm)
* React-Toastify for notifications

**4.2 Folder map (high level)**

frontend/

├─ public/

├─ index.html

├─ vite.config.js

├─ src/

│ ├─ main.jsx

│ ├─ index.css

│ ├─ App.jsx

│ ├─ Auth/

│ ├─ Components/

│ ├─ pages/

│ │ ├─ Dashboard/

│ │ ├─ Profile.jsx

│ │ ├─ main/

│ │ ├─ canvas/

│ │ ├─ Room/

│ │ └─ sidebar/

│ ├─ assets/img/

│ └─ utils/

└─ package.json

**4.3 Routing & auth flow**

Routes (summary):

* / — public landing
* /login, /register — public only routes
* /dashboard, /profile, /JoinCreateRoom, /main — protected routes

Guarding: ProtectedRoute and PublicOnlyRoute components call GET /api/users/profile (reads cookie) to resolve the user state. App.jsx maintains loggedIn, userId, and username in top-level state and provides handleLogout which clears auth state and cookie.

**4.4 Key components & whiteboard flow**

**Main flow**

1. JoinCreateRoom - user supplies display name + room id (create or join).
2. Main.jsx - creates single Socket.IO client. Based on user.presenter flag it renders Room (editor) or ClientRoom (viewer).
3. Room.jsx - editor UI: toolbar (tool selector, color, undo/redo, clear, download PNG, save to DB) and <Canvas/> (Rough.js). Emits drawing event with {roomId, imageData} (base64 PNG) throttled to ~200ms.
4. ClientRoom.jsx - listens for drawing events and updates an <img src> to show the latest frame (~5 fps experience).
5. Sidebar - slide-out user list sourced from socket users event; keyboard shortcut Ctrl/Cmd+K opens overlay.

**Canvas features**

* Auto-resizes to container and device pixel ratio
* Pencil, line, rectangle strokes
* Touch support (with passive/active toggle)
* Emits PNG base64 on mouse-up and throttled during move for live preview

**4.5 State & networking**

* Simple lifted React state (useState) at root; hooks & props for communication.
* Single socket singleton created in Main.jsx and passed through context or props.
* Global Axios instance configured with base URL (see utils/axios.js) and withCredentials:true.

**4.6 File uploads & gallery**

* Avatars: POST /api/users/avatar as multipart form-data — after successful upload the UI updates preview and the backend stores the filename in user.avatar.
* Canvas save: canvas.toBlob() → FormData file → POST /api/upload (Multer disk) → backend stores filename and reference in user.images[]. Dashboard loads user images from GET /api/images.
* Deletion: DELETE /api/images/:filename removes DB ref and physical file on disk.

**4.7 Styling & dev commands**

* Tailwind 4 via official Vite plugin; some pages include custom CSS for sticky-note animation in auth screens.

Dev commands:

npm i

npm run dev

npm run build

**4.8 Known front-end limitations**

* Single presenter per room (hard-coded)
* Broadcasts full PNG frames — bandwidth heavy
* No pointer labeling or per-user cursors on canvas
* Mobile/touch support tested lightly; Apple Pencil may need extra plugin

**5 — Backend**

**5.1 Tech stack**

* Node.js 18 (ES Modules)
* Express 4
* MongoDB 6+ (Mongoose 8)
* Socket.IO 4
* Multer (disk storage)
* bcrypt + JWT for auth
* dotenv for environment variables

**5.2 Folder map**

backend/

├─ server.js

├─ app.js

├─ config/db.js

├─ controllers/

├─ middleware/

│ ├─ auth.js

│ ├─ errorHandler.js

│ ├─ multer.js

│ └─ socket.js

├─ models/

│ └─ User.js

├─ routes/

├─ utils/

│ └─ users.js

├─ img/

└─ .env

**5.3 Environment variables**

Example .env (DO NOT commit real secrets):

PORT=5000

DB=mongodb://localhost:27017/WhiteboardDB

JWT\_SECRET\_KEY=<YOUR\_SECRET>

NODE\_ENV=development

**5.4 Data model — User (summary)**

{

username: String (3-30 chars),

email: String (unique),

password: String (bcrypt hash),

images: [String],

avatar: String,

timestamps: createdAt/updatedAt

}

Model behavior:

* Pre-save hook to hash password
* comparePassword(candidate) helper
* toJSON() strips password hash before send

**5.5 Authentication flow**

1. POST /api/auth/register – create user with {username,email,password}.
2. POST /api/auth/login – validate credentials; on success set http-only cookie token=JWT (1 day by default).
3. Protected routes use middleware/auth.js to verify cookie.
4. POST /api/auth/logout – clears cookie.

**5.6 API endpoints (reference)**

Base path: http://localhost:5000/api

| **Method** | **Route** | **Auth** | **Description** |
| --- | --- | --- | --- |
| POST | /auth/register | ❌ | Create account |
| POST | /auth/login | ❌ | Login & set JWT cookie |
| POST | /auth/logout | ✅ | Clear auth cookie |
| GET | /users/profile | ✅ | Get own profile |
| PUT | /users/profile | ✅ | Update username |
| POST | /users/avatar | ✅ | Upload avatar (multipart avatar) |
| POST | /upload | ✅ | Upload whiteboard PNG (multipart file) |
| GET | /images | ✅ | List user's images |
| DELETE | /images/:filename | ✅ | Delete file + DB ref |
| GET | /img/:filename | ❌ | Static serve of uploaded image |

Response conventions: JSON with {status,message,data} or resource objects. Error handler returns standardized JSON via middleware/errorHandler.js.

**5.7 File upload details**

* Multer stores files to /img using filename <field>-<timestamp>.<ext> to prevent collisions.
* After upload, the server pushes the filename to the current user’s images[] or sets user.avatar accordingly.
* Deletion removes both DB reference and the disk file.
* Add fileFilter / limits.fileSize for production.

**5.8 Real-time server (Socket.IO)**

* Socket.IO shares the same HTTP server (created in server.js).
* CORS is configured for http://localhost:5173 during dev.
* Socket middleware wiring is handled in middleware/socket.js and uses utils/users.js to maintain in-memory room participant lists.
* Expected events (see next section) include join-room, draw (or drawing), users, and message.
* In-memory helpers are not persisted — consider backing room state if you need history.

**5.9 Security hardening**

Implemented / recommended:

* Password hashing (bcrypt 12 rounds)
* JWT secret in env
* http-only cookies; in production set secure=true and appropriate sameSite
* CORS whitelist with credentials enabled

Recommended additions before production:

* Rate limiting (express-rate-limit)
* Helmet and other header hardening
* Input sanitization (mongo-sanitize / xss-clean)
* Move uploads to S3/GCS with pre-signed URLs and remove direct static serving of /img

**5.10 Dev & deployment notes**

Dev:

npm install

npm run dev # nodemon

Deployment:

* Build frontend with vite build → dist/ and serve as static files behind Nginx or the same Express server.
* Use reverse proxy (Nginx) and enable WebSocket upgrade headers for socket connections.
* Use PM2 or systemd for process management.
* Persist uploads or switch to cloud storage.

**6 — Socket.IO event contract (canonical)**

| **Event** | **Direction** | **Payload** | **Description** |
| --- | --- | --- | --- |
| user-joined / join-room | client → server | {userName, roomId} | join room / register participant |
| drawing / draw | client → server | {roomId, imageData} | base64 PNG frame (throttled) |
| drawing | server → clients | {imageURL or imageData} | broadcast latest frame |
| users | server → clients | [{id, username}, ...] | current active user list |
| message | server → clients | {message} | generic toast/info |

Notes:

* Frames are full PNG images (bandwidth heavy). Consider a more efficient format (vector ops, diffs, binary frames) for scale.
* Throttling: client throttles frames to ~200ms so viewers see ~5 fps.

**7 — Production checklist (short)**

**Backend**

* Move uploads to S3/GCS and remove direct static serving of /img.
* Enforce rate limits and security headers (helmet).
* Use NODE\_ENV=production to enable secure cookies and other settings.
* Use a process manager (PM2) and reverse proxy (Nginx) with SSL and gzip.

**Frontend**

* Set VITE\_API\_URL and update axios baseURL accordingly.
* vite build and serve dist/ via CDN/Nginx with SPA fallback to index.html.
* Configure WebSocket proxy in Nginx (upgrade headers).

**Monitoring & testing**

* Logging (winston/pino), uptime probes, and application metrics.
* Add E2E and load testing (Cypress, k6).

**8 — Testing plan (recommended)**

* **Unit tests**: Jest + supertest for backend controllers and middleware.
* **Integration tests**: Postman collection / Newman to run all REST endpoints.
* **Socket tests**: socket.io-client mocks to assert join/leave/draw behavior.
* **E2E**: Cypress: register → create room → draw → save → delete.
* **Load**: k6 / Artillery: simulate many viewers and presenters, watch frame latency.

**9 — Troubleshooting quick-FAQ**

**Q: Cookie not set in browser**

* Ensure frontend requests use withCredentials:true and that CORS origin matches exactly.

**Q: 401 on calls after login**

* Verify JWT\_SECRET\_KEY is the same for login & auth middleware and cookie domain/path is correct.

**Q: Uploaded images return 404**

* Ensure app.use('/img', express.static(...)) is mounted before any catch-all /\* route.

**10 — Changelog**

* **v1.0.0 — 29 Oct 2025**: Initial full-stack release — JWT auth, avatar uploads, canvas PNG upload, live room drawing with Rough.js + Socket.IO, dashboard and gallery.

**11 — Appendix**

* Source documents used to build this unified doc: frontend docs, backend docs, and fullstack summary (original project docs). The combined doc consolidates routes, contracts and steps needed to run & deploy the system.

*End of document.*