**Day 1**

Node -> socket io-> chat app-> websocket Done:

<https://chatgpt.com/c/684d7609-a9c8-8008-82a2-6e0ff1dc4bb8>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

1.Login page using JWT Google Oauth

(node over Deno js)

Mongo Db required, schemas

JWT

Async Promises

(Vite over npx)

Creating react app and express server

( dotenv to read env variables)

<https://chatgpt.com/c/6857de2b-ed50-8008-89a9-b04153b82e53>

Mongo DB compass

(Schema, models)

Difficult to connect Compass with Atlas (used 0.0.0. as of now)

<https://chatgpt.com/c/6858b4cd-bf5c-8008-b937-234cdf08dd49>

Promises, fetch, async, await

Mongoose

**Features**

Slack’s workflow and feature set is designed for **team-based, real-time communication and collaboration**. Below is a breakdown of **Slack’s key features** and the **user workflow**, which you can use for inspiration in your *Collaborator’s Aura* project.

**🧭 Slack: Features & Workflow**

**🧩 1. Workspaces**

* **Definition**: A workspace represents an organization or team.
* Users **sign up or join** a specific workspace via invite.
* Each workspace is **isolated** — different teams, channels, files.

**👥 2. User Roles**

* **Workspace Owner / Admin**: Full control, manage users/settings.
* **Members**: Can send messages, create channels, etc.
* **Guests**: Limited access (to specific channels only).

**🔄 Core Workflow of Slack**

**🔐 1. Sign-Up / Login**

* Join via an **invitation link** or register directly.
* Can join **multiple workspaces**, switch between them from the sidebar.

**🏠 2. Workspace Dashboard**

* Shows:
  + **Sidebar**: Channels, direct messages, apps, and workflows.
  + **Main area**: Current conversation or app interface.
* **Top bar**: Search, notifications, status, and profile settings.

**💬 3. Channels**

* **Public channels**: Visible to all members in the workspace.
* **Private channels**: Invite-only.
* **Direct Messages (DMs)**: 1-on-1 or group chats.

Features inside channels:

* Threaded conversations
* Reactions and emoji support
* Mentions (@user, @channel, etc.)
* Pinned messages

**🛠️ 4. Workflows / Apps**

Slack Workflows:

* No-code automation like:
  + "Request leave"
  + "Daily standup reminder"
  + "Feedback collector"

Integrations:

* Apps like Google Drive, Trello, GitHub, Zoom, etc.

Advanced workflows:

* Triggered by form inputs, messages, or external APIs

**🔁 5. Notifications**

* Per-channel notification control
* Highlights for mentions
* Mobile & email notifications

**🗂️ 6. Files & Media**

* Upload documents/images in channels and DMs
* Searchable archive
* Integrations with cloud storage

**🔍 7. Search & History**

* Powerful search with filters (by user, channel, date, type)
* All messages and files are searchable (limited on free plan)

**⚙️ 8. Settings & Customization**

* Theme selection (light/dark)
* Custom emoji, reactions
* Profile management, status updates

**🛡️ 9. Security & Admin Controls**

* 2FA support
* Admin panel for managing:
  + Users
  + Permissions
  + Audit logs
* Data export for compliance (Enterprise plan)

Would you like a **side-by-side comparison** of Slack and your **Collaborator’s Aura** features for planning, or a **workflow diagram** to visualize this?

Perfect — you're aiming to add **AI-driven task summarization** and **Google Calendar sync** to enhance productivity in Collaborator’s Aura. Here's a summarized breakdown of your new feature scope, how it aligns with Slack-like workflows, and what you'll need technically.

**🔄 New Features to Add in Collaborator's Aura**

**1. ✅ Task Summarization from Chat or Task Page**

**Goal**:  
Automatically detect or let users summarize tasks from a conversation or a task submission page.

**Workflow**:

* In chat:
  + Select a message or thread → Click "Summarize Task"
  + AI extracts task title, description, due date (if mentioned), and assignees
* In task page:
  + AI assists in summarizing typed content into structured task fields

**Output**:

* Create a structured task (title, description, priority, deadline)
* Add it to the workspace's task board
* Show confirmation modal with "Add to Google Calendar?"

**Implementation Stack**:

* **Gemini API or OpenAI GPT**: For summarization logic
* Task model: Task { id, title, description, assignedTo[], createdBy, dueDate, status }
* Optional: Keywords trigger like /task, TODO, etc.

**2. 📆 Google Calendar Integration for All Workspace Members**

**Goal**:  
Automatically push created tasks to each member’s Google Calendar.

**Workflow**:

* On task creation, prompt: *“Add to everyone's Google Calendar?”*
* If confirmed:
  + Authenticate via Google OAuth (first time per user)
  + Use **Google Calendar API** to:
    - Create an event with title, description, and due date
    - Send invites to all assignedTo users

**Required Setup**:

* Google Cloud Project → Enable Calendar API
* OAuth 2.0 setup with consent screen
* Store tokens (securely) per user for future API calls
* Handle token refresh

**Event Format Example**:

{

"summary": "Fix real-time sync bug",

"description": "Assigned by Alice. Deadline: 8th Aug.",

"start": { "dateTime": "2025-08-08T10:00:00+05:30" },

"end": { "dateTime": "2025-08-08T11:00:00+05:30" },

"attendees": [

{ "email": "member1@xyz.com" },

{ "email": "member2@xyz.com" }

]

}

**🧠 Bonus Suggestions**

* Add a “My Calendar Sync” toggle in profile settings.
* Auto-detect deadlines from messages using NLP (e.g., “complete by Friday” → parse date).
* Show synced task status: ✅ Synced | 🔄 Pending Sync | ❌ Failed

**📋 Updated Requirements Summary**

Add this to your functional requirements:

**🆕 Smart Task + Calendar Features**

* AI-based task summarization from chat or task page
* Structured task creation with assignees and deadline
* Google OAuth & Calendar sync integration
* Option to add tasks to all assignees' calendars
* Handle calendar token revocation and refresh logic

Would you like a Notion-style doc, ERD update, or a UI mockup for this feature next?

Got it — you're aiming to build a **complete Google Docs-style real-time editor** *with all the core features of Docs* **plus software design architecture tools**.

That means replicating:

**✅ Full Google Docs Feature Set + Software Design Architecture**

Let’s break this down.

**🔹 Core Google Docs Features to Reproduce**

| **Feature** | **Your Implementation Plan** |
| --- | --- |
| 📝 Rich text editing | Use **TipTap** (based on ProseMirror) or **Lexical** |
| 🧑‍🤝‍🧑 Real-time collaboration | Use **Yjs** or **Automerge** with WebSockets |
| 🖱️ Cursor presence | Yjs awareness plugin (for real-time cursors) |
| 💬 Comments & suggestions | TipTap comment extensions or custom threaded comments |
| 📜 Version history | Snapshots with Yjs or backend-based changelogs |
| @Mentions | TipTap mention plugin |
| 🔍 Search inside document | TipTap + JavaScript regex + highlight overlays |
| ⌨️ Keyboard shortcuts | Custom handlers in TipTap |
| 📥 Export to PDF | Use **html2pdf.js** or **Puppeteer backend** rendering |
| 📑 Document organization | Use sidebar with folders/doc lists per workspace |

**🔹 Software Design Architecture Features to Add**

| **Diagram Type** | **Library/Tool** | **Integration Idea** |
| --- | --- | --- |
| **UML / Flowcharts** | [Mermaid.js](https://mermaid-js.github.io/) | Inline rendering or modal editor block |
| **Freehand Sketches** | [Excalidraw](https://excalidraw.com/) | Embed canvas in doc or open full screen |
| **draw.io** | Embed iframe + save XML/SVG in DB | For power users with diagram.net flow |
| **ERD / DB models** | Mermaid.js or dbdiagram.io export | Render in structured block |
| **Sequence Diagrams** | Mermaid or PlantUML | Text-based input with preview |

Add these as **block-level components inside your rich text editor**.

**🧩 Final Architecture Stack**

**⚛️ Frontend (React)**

* **Editor**: TipTap or Lexical
* **Real-time**: Yjs with WebSocketProvider
* **Diagram Blocks**:
  + Mermaid.js preview block
  + Excalidraw canvas (embedded or modal)
  + draw.io iframe (advanced use)
* **UI**:
  + Document title, sidebar nav
  + Comments popup on selection
  + Presence indicators
  + Diagram “insert” button in toolbar

**🖥️ Backend**

* **Node.js + Express**
* **WebSocket server** (Yjs sync)
* **MongoDB / PostgreSQL**:
  + Documents: Rich text + Yjs state
  + Diagrams: Separate JSON (Excalidraw), SVG/XML (draw.io)
  + Users, permissions, versions, comments

**🧠 Feature Idea: Smart Collaboration**

* 🧠 AI-assisted summarization of docs
* 🔗 Link architecture blocks to task system
* 📆 Export deadlines from diagrams to calendar
* 🧩 Insert live code snippets or code blocks with syntax highlighting

**🛠️ Sample UI Block Layout**

[ Toolbar: Bold | Italic | Heading | Insert Diagram | Comment | Export ]

──────────────────────────────────────────────────────────────

| Document Title 🔍 Search |

──────────────────────────────────────────────────────────────

| Alice 👩‍💻 is editing... Bob 🧑‍💻 is viewing... |

──────────────────────────────────────────────────────────────

| - This is the design for the User Auth system. |

| |

| 🧱 [Embedded Mermaid Diagram Block] |

| |

| - The sequence begins when the user logs in... |

| |

| 💬 [Comment on “logs in”] |

──────────────────────────────────────────────────────────────

**✅ Your Goal Summary:**

**Google Docs + Architecture Editor**

✅ Rich text editor  
✅ Real-time editing with cursor presence  
✅ Comments, suggestions, version history  
✅ Diagram block insertion (Mermaid, Excalidraw, draw.io)  
✅ Workspace-based document organization  
✅ Export to PDF/Markdown  
✅ Role-based access control (editor, viewer)

Would you like:

1. A **ready-made React component boilerplate** for TipTap + Mermaid + Excalidraw?
2. A **UI/UX wireframe mockup**?
3. A **modular backend API plan** for handling documents + diagrams?

Great! Here's a breakdown of:

**✅ Google Docs-Like Templates to Support**

You can pre-build or allow creation of templates for the following:

**📁 Project/Team Docs Templates**

1. **Meeting Notes**
2. **Review Tracker**
3. **Project Tracker**
4. **Sprint Planning / Retrospective**
5. **Design Decisions Record (ADR)**
6. **Product Requirements Document (PRD)**
7. **OKRs Template**
8. **Bug/Issue Tracker**
9. **Daily Standup Summary**

**📐 Software Design/Architecture Docs**

1. **Software Design Doc (SDD)**
2. **Architecture Overview (C4 Model-style)**
3. **API Contract Document**
4. **Component/Service Ownership Map**
5. **ERD (embed diagrams via draw.io, Excalidraw, Mermaid)**

**📌 Feature Requirements to Build Google Docs + Software Design Editor**

Here are the key features you should plan for your Collaborator's Aura project:

**🖊️ Rich Text Editor (Google Docs-Like)**

* Paragraphs, headings, bullet points, code blocks
* Inline formatting (bold, italics, links)
* Slash commands (/heading, /table, /checkbox, etc.)

**📊 Smart Tables (like Review Tracker)**

* Add/delete rows and columns with hover +
* Editable cells with:
  + Checkboxes
  + Status dropdowns
  + Date pickers
  + Assignee (select from workspace members)
* Sort by columns
* Hover toolbars for sorting/editing

**🧩 Custom Field Types**

* Dropdowns (e.g., Status, Priority)
* Calendar picker (Due Dates)
* Assignee lookup (workspace member selection)
* Tags (colored chips)

**✨ Templates System**

* Pre-built templates (stored in DB as TipTap JSON)
* Ability to "Use Template" and auto-create a new doc
* Editable metadata (title, icon, type, team)

**👥 Real-Time Collaboration**

* Cursor sync, edits visible live (Yjs + WebSockets)
* Presence indicators (avatar dots)
* Commenting / suggestions (optional)

**🔄 Revision History**

* Track document edit history (snapshot or versioned JSON)
* Restore from previous versions

**📐 Software Design Tools to Integrate**

**1. Embeddable Diagram Support**

* Embed [Excalidraw](https://excalidraw.com/) canvas (or use @excalidraw/excalidraw)
* Mermaid.js for sequence/flow/state diagrams
* UML templates
* Integration with Draw.io (optional)

**2. Document Structure & Navigation**

* Table of Contents sidebar
* Anchors for headings (like Notion)
* Navigation pane for multi-section design docs

**🚀 Optional Add-ons**

* AI Summarization of docs (Gemini/GPT API)
* One-click export to PDF
* Shareable public/private links
* Google Calendar sync for timelines/due dates
* Mention/tag users (@username)
* Attachments (upload design files, images, PDFs)

Would you like a **feature roadmap in table form** or a **modular implementation plan** (e.g., Sprint 1: Editor, Sprint 2: Templates, Sprint 3: Smart Tables, etc.)?

Let me know and I’ll tailor it accordingly.

Here’s your **complete, detailed roadmap** for building **Collaborator’s Aura**, your all-in-one platform combining **Google Docs + Slack + Notion + Software Architecture tools**.

**🗓️ Complete Roadmap: Collaborator’s Aura**

**Total Duration:** ~14 weeks (2–3 months of consistent solo work at ~30–40 hrs/week)

**✅ Phase 1: Core Setup (Week 1)**

| **Task** | **Tools** |
| --- | --- |
| Project scaffolding (React, Node.js, Express, MongoDB/Postgres) | Vite + Express + Mongoose/Prisma |
| User auth system (JWT + Google OAuth) | Clerk/Auth.js/Firebase Auth |
| Workspace management (create/join/switch) | DB schema for multi-tenant setup |
| Roles and permissions | Admin, Editor, Viewer |
| UI setup with Tailwind + shadcn/ui | Reusable layout, sidebar, nav |

**✍️ Phase 2: Google Docs-like Editor (Week 2–3)**

| **Task** | **Tools** |
| --- | --- |
| Integrate TipTap with Yjs (real-time) | TipTap + Yjs + WebSocket/Socket.IO |
| Text formatting, headings, links, lists | TipTap nodes/extensions |
| Comments & user presence | Collaborative cursors, presence panel |
| Document CRUD + autosave | Backend + DB support |

**📑 Phase 3: Templates & Smart Tables (Week 4)**

| **Task** | **Tools** |
| --- | --- |
| Template library (Review Tracker, Meeting Notes, PRD, SDD) | JSON-based templates |
| Smart tables (add row/col, dropdowns, checkboxes) | Custom TipTap table node |
| Field-type support (text, date, select, checkbox) | Inline menu or contextual popup |
| Template creation from UI | “Save as Template” button |

**🧠 Phase 4: Task Management + Google Calendar Sync (Week 5)**

| **Task** | **Tools** |
| --- | --- |
| Task creation (inline + task dashboard) | /task slash command or task page |
| Assignments, deadlines, status | Linked to users in workspace |
| Sync to Google Calendar API | Use OAuth2 flow + Calendar API |
| Tasks linked to docs or chat messages | Contextual popups and links |

**🏗️ Phase 5: Software Design + Architecture Tools (Week 6)**

| **Task** | **Tools** |
| --- | --- |
| Embed Excalidraw canvas in editor | Excalidraw open-source integration |
| Mermaid.js support for diagrams | TipTap code block renderer |
| Support for C4 model, PRD/ADR templates | Form-based doc creation |
| Documented design components | Microservice flow, infra, DB schemas |

**💬 Phase 6: Workspace Chat System (Week 7–8)**

| **Task** | **Tools** |
| --- | --- |
| Real-time chat per workspace | Socket.IO rooms + React |
| Threads and replies | Thread model with parent-child messages |
| Chat per doc (like Notion) | Sidebar or inline comments/chat |
| Mentions + Notifications | @mention handling + badge alerts |
| File/image upload | AWS S3, Firebase Storage, or local |
| Search chat history | Text index + filters |

**🧩 Phase 7: Collaboration Enhancements (Week 9)**

| **Task** | **Tools** |
| --- | --- |
| Slash commands (/task, /diagram) | Custom TipTap command handler |
| Workspace invite links | Invite via email or share link |
| Collaborative cursors, avatars | TipTap + Yjs presence system |
| Access control (role-based editing) | Viewer/Editor/Admin toggle |

**📖 Phase 8: Revision History, Export & Sharing (Week 10)**

| **Task** | **Tools** |
| --- | --- |
| Snapshot versioning | Save point-in-time JSONs of docs |
| Export to PDF/Markdown | TipTap to Markdown/HTML → PDF |
| Public link sharing (read-only) | Token-based access |
| Revert to previous versions | Diff viewer + rollback action |

**🤖 Phase 9: AI Add-ons (Week 11)**

| **Task** | **Tools** |
| --- | --- |
| Auto-summary of docs | Gemini/GPT API |
| Task/action extraction from docs or chat | NLP task parser |
| Daily chat digest (Slack-style summary) | Scheduled summarizer |
| AI-powered template suggestions | Suggest layout based on title |

**🧪 Phase 10: Polish, Testing & UX (Week 12)**

| **Task** | **Tools** |
| --- | --- |
| Responsive design, clean UI, dark mode | Tailwind media queries |
| Performance optimization (pagination, debounce) | Lazy loading, infinite scroll |
| Form validations, tooltips, modals | Zod/Yup + shadcn/ui |
| UX microinteractions | Framer Motion, transitions |

**🚀 Phase 11: Deployment, Demo & Resume Integration (Week 13–14)**

| **Task** | **Tools** |
| --- | --- |
| Deploy frontend & backend (Vercel/Render) | Vercel (React) + Render/Fly.io (Express) |
| Create a demo workspace for recruiters | Pre-filled dummy data |
| GitHub repository with README + badges | Live demo link + video |
| Write bullet points for resume/LinkedIn | STAR format highlights |

**📊 Summary Timeline (Visual):**

Week 01: Setup + Auth + Workspaces

Week 02–03: Real-time Editor (TipTap + Yjs)

Week 04: Templates + Smart Tables

Week 05: Task Management + Google Calendar

Week 06: Software Design Tools (Excalidraw, Mermaid)

Week 07–08: Real-time Chat + Doc Chat + Mentions

Week 09: Collab Layer (Slash cmd, Roles, Invite)

Week 10: Export + Version History

Week 11: AI Add-ons (Summary, Extraction)

Week 12: UI Polish + Testing

Week 13–14: Deployment + Demo + Resume

**🛠️ Optional Tools & Stack**

| **Category** | **Options** |
| --- | --- |
| **Frontend** | React, Tailwind, shadcn/ui, Zustand |
| **Backend** | Node.js, Express, MongoDB/PostgreSQL |
| **Real-time** | Socket.IO, Yjs |
| **AI** | Gemini, OpenAI GPT-4 |
| **Design Tools** | Excalidraw, Mermaid |
| **Editor** | TipTap (with custom nodes for smart features) |
| **Calendar** | Google Calendar API |
| **Deployment** | Vercel (frontend), Render/Fly.io (backend), MongoDB Atlas |
| **Auth** | Clerk, Firebase, OAuth2 |

Would you like this as a:

* 🗂️ Notion board template?
* 📄 PDF roadmap?
* 🧑‍💻 GitHub project board (Issues + Labels)?

Let me know, and I’ll prepare it accordingly.

| **Layer** | **Tool** |
| --- | --- |
| Frontend | React + Zustand + Tailwind + TipTap |
| Backend | Node.js + Express |
| Real-Time | Socket.IO |
| Database | MongoDB + Mongoose |
| Auth | JWT or Firebase Auth |
| Deployment | Vercel (frontend), Render/Railway (backend), MongoDB Atlas |

<https://chatgpt.com/c/68901086-4210-8008-a5d8-ab88586e94e9>

**Work:**

**Day1: 4/8/25**

Folder created,

Frontend – created react app

Backend – npm init, installed packages

npm install jsonwebtoken bcrypt body-parser dotenv mongoose cors nodemon

to start node js project (server) we run nodemon – so that reloads when changes occur

update scripts in package.json, use nodemon as start – to start entry point – index.js of node js server

create index.js to create express app() i.e. server program

.env file in backend – to store connection string, port (not hardCoded)

Create db.js to connect server to DB

Copy connection string with db name and connect using mongoose

Include the db.js in index.js to run it once server is started

Now creating schema for User using mongoose in Models folder

Mongoose.model() - so user becomes a collection with defined schema and now User obj can be used to manipulate this collection

Middlewares - Now getting the username, etc from client to server – body parser is used in index.js

(body-parser is now built into Express as express.json() and express.urlencoded())

App.use(cors()) allows your frontend and backend to talk to each other when they’re on different domains or ports. (i.e. we just run html file in browser like giving whole path but in react the server is started and renders the index.html)

Middlewares – are written above all the routes, app.use() is used to load the middleware

First middlewares are executed then routes after it.

Middlewares can be used for:

* 1. To execute certain code
  2. To modify req and res objects
  3. End the req-res cycle
  4. Call the next middleware in cycle

Fetch() is used to make API call, can be GET (get data from server), POST (to send data to server), etc.

Express routers are used to modularize multiple routes into a single route by mounting

Created rout auth to handle post requests made for signup and login

Syntax: router.post(path, middleware(s), handler)

Creating the controllers – for sign up first – controllers are used to handle the backend logic when a particular route/url is hit

| **Purpose** | **Joi** | **JWT** |
| --- | --- | --- |
| ✅ What it does | Validates incoming data (email, password, etc.) | Creates secure tokens after successful login |
| 📦 Used for | Input validation | Authentication |
| 🌐 When used | Before processing logic | After successful login |
| 💡 Example | "Is email valid?" | "Issue a token for this user" |

Creating Joi signup and login validation – as a middleware before going to signup or login logic

Checking on postman to test server logic and entries into the db

**Day 2: 5/8/25**

Created login logic (controller), sending JWT token to client for authentication purpose

When the user logs in, server sends the client the token. Whenver the client make any request to server after login, the header of url contains the authorization token which is sent to server for verification if token is valid or not.

Now creating the frontend ui for Login/signup and sending the data of form using fetch for api call

BrowserRouter is Context

(e) is the event object i.e. sent to handlechange whenever onchange event occurs

Installing tailwind css after creating sign up/ login

npm install react-router-dom

**Day 3: 6/8/25**

npm install -D tailwindcss@3.4.3 postcss@8.4.38 autoprefixer@10.4.19

installed newer version of node for compatibility with daisy ui

added tailwind.config and postcss.config for daisy ui

Ignore the warnings unless error (since plugin wasn’t recognized as rule, took time to resolve, so ignorance is solution)

We can use daisy ui but if we want to customize we can use tailwind css.

Setting the dark, light themes, daisy ui has dark-light support but for custom, we need persistent storage hence localstorage

Going with default light and dark theme

**Day 4: 8/8/25**

Added alert and successful modal after signup

Added Login frontend redirecting to chat

**Day 5: 10/8/25**

Learning SocketIO Websocket is a communication protocol like HTTP

HTTP client -> req -> server then server->res->client stop

Websocket client<->req,res<->server persistent connection

**Day 6: 11/8/25**

We can use standalone Socket.io library of node js or we can use it with express

To establish a websocket connection without any framework, we need the client to send – connection: upgrade Upgrade: websocket, this will tell the server that the connection is a websocket type of connection and does not end the req-res cycle (Like in HTTP)

We will use socket.io library

Creating a TestChat folder to test socket.io and create a chat