TUNG DUC VU

DeepSeek Clone

DeepSeek interface - Google Gemini API

THYNK UNLIMTED

Structure

- app
  - api
    - chat
      - ai
      - create
      - delete
      - get
      - rename
    - clerk
  - layout.js
  - page.js
- components
  - ChatLabel.jsx
  - Message.jsx
  - PromptBox.jsx
  - Sidebar.jsx
- config
  - db.js
- context
  - AppContext.jsx
- models
  - Chat.js
  - User.js

THYNK UNLIMTED

app/api

- chat
  - ai
  - create
  - delete
  - get
  - rename
- clerk

chat/ai/route.js

- **Auth Check:** Clerk *getAuth* to verify *userId*
- **DB Fetch:** *connectDB*, find Chat by *userId* & *chatId*
- **User Prompt:** Add user message to messages array
- **Gemini API:** Call *gemini-2.5-flash* model, get response
- **Save & Respond:** Append AI reply, save chat, return JSON response

AI answers generation

chat/create/route.js

- **Auth Check** – Clerk *getAuth* to verify *userId*
- **Validate User** – Return error if *userId* is missing
- **Prepare Data** – Create empty chat object with *userId*, messages, name
- **DB Connect & Save** – *connectDB*, *Chat.create(chatData)*
- **Response** – Return success message as JSON

Chat creation

chat/delete/route.js

- **Auth Check** – Clerk *getAuth* to verify *userId*
- **Parse Input** – Extract *chatId* from request JSON
- **Validate User** – Return error if *userId* is missing
- **DB Connect & Delete** – *connectDB*, *Chat.deleteOne({ _id, userId })*
- **Response** – Return success or error as JSON

Chat deletion

chat/get/route.js

- **Auth Check** – Clerk *getAuth* to verify *userId*
- **Validate User** – Return error if *userId* is missing
- **DB Connect** – Call *connectDB()*
- **Fetch Chats** – *Chat.find({ userId })*
- **Response** – Return chat list as JSON

Chat fetching

chat/rename/route.js

- **Auth Check** – Clerk *getAuth* to verify *userId*
- **Parse Input** – Extract *chatId* and name from request JSON
- **Validate User** – Return error if *userId* is missing
- **DB Connect & Update** – *connectDB*, *Chat.findOneAndUpdate({ _id, userId }, { name })*
- **Response** – Return success message as JSON

Chat renaming

clerk/route.js

- **Webhook Verify** – Use svix to verify incoming payload with headers
- **Parse Event** – Extract data and type from verified payload
- **Prepare User Data** – Format user info (id, email, name, image)
- **DB Connect & Handle** – connectDB, perform action based on event type: create, update, or delete
- **Response** – Return confirmation message as JSON

Clerk Webhook User Sync Handler

app/layout.js

- **Font Setup** – Import and configure Inter font
- **Clerk Auth** – Wrap with *ClerkProvider* for authentication
- **Global State** – Use *AppContextProvider* for app context
- **Styling** – Import *globals.css* and *prism.css*
- **Toast Alerts** – Configure *react-hot-toast* with custom styles

Next.js Layout with Clerk Auth

app/page.js

- **Sidebar Toggle** – Controls mobile and desktop sidebar expansion
- **Load Messages** – Loads messages from *selectedChat* using *context*
- **Auto Scroll** – Scrolls to bottom when messages update
- **Render Chat** – Displays messages or welcome screen based on chat state
- **Prompt Input** – Uses *PromptBox* for sending messages and shows loading

Main Chat UI

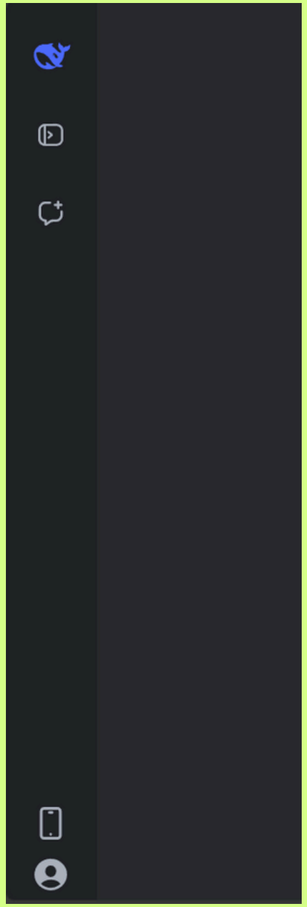
THYNK UNLIMTED

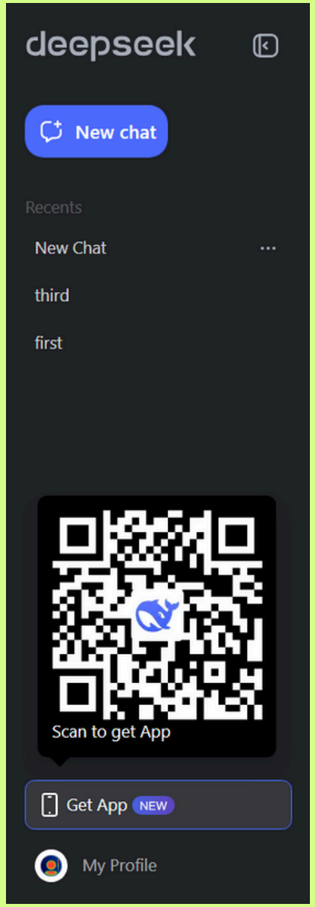
Components

- ChatLabel.jsx
- Message.jsx
- PromptBox.jsx
- Sidebar.jsx

Sidebar.jsx

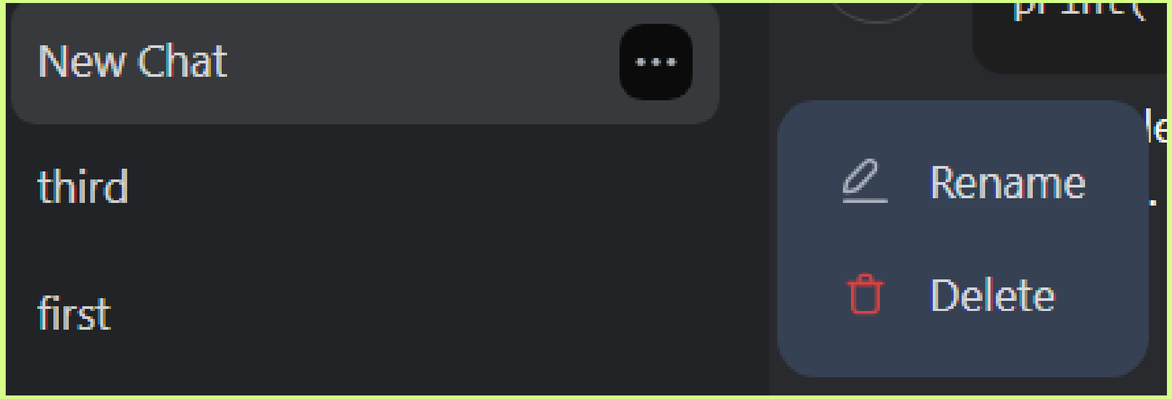
- **Sidebar Toggle** – Expand/collapse sidebar with icon click
- **Logo Display** – Show *logo_text* or *logo_icon* based on state
- **New Chat Button** – Trigger *createNewChat*, adjust style by expand
- **Recents List** – Map and render recent chats using *ChatLabel*
- **QR Code Tooltip** – Hoverable QR tooltip for "Get App" section
- **Auth/Profile** – Show *UserButton* if logged in, or open sign-in
- **Responsive Design** – Handle mobile/desktop view with Tailwind classes





ChatLabel.jsx

- **Select Chat** – On click, fetch chat by ID and set as selected
- **Rename Chat** – Prompt for new name, call API, show toast feedback
- **Delete Chat** – Confirm deletion, call API, refresh chat list
- **Menu**
  - Toggle 3-dot options menu per chat
  - Dynamic – Show rename/delete options only for active chat
- **Context Integration** – Uses AppContext for state and chat actions

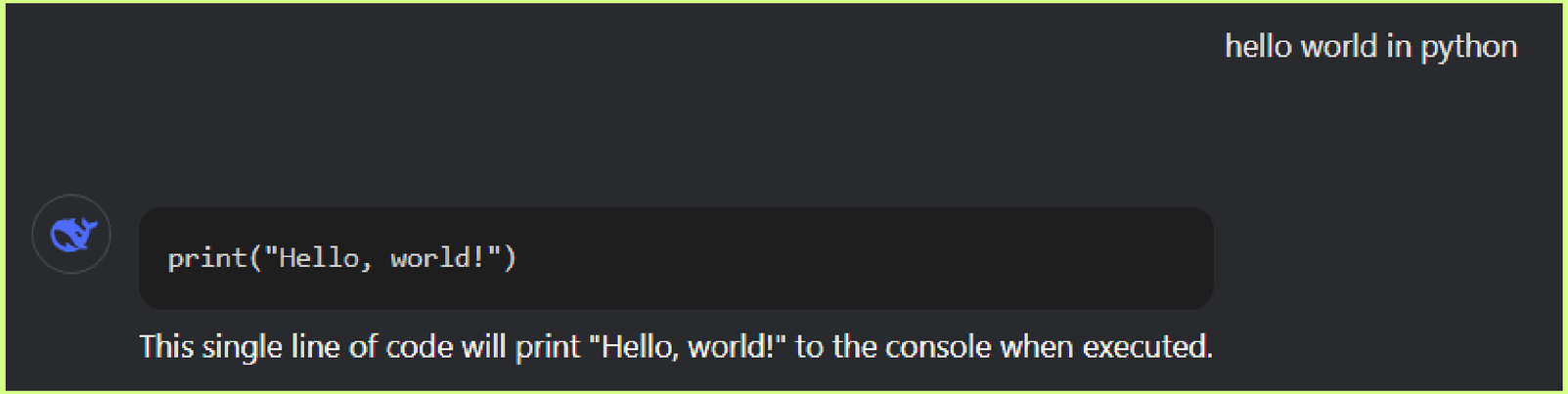


PromptBox.jsx

- **Prompt State** – *useState* to track user input
- **Validation Checks** – Ensure user is logged in, chat selected, and not loading
- **Send Message** – Add user prompt to UI and selected chat state
- **API Request** – POST to */api/chat/ai* for Gemini response
- **Simulated Typing** – Animate assistant reply word by word
- **Key Handling** – Submit on Enter without Shift
- **UI Elements** – *Textarea*, tag, and submit button with dynamic styling

Message.jsx

- **Syntax Highlighting** – Uses *Prism.highlightAll()* on message render
- **Markdown Support** – Renders assistant message with *ReactMarkdown + rehype-prism-plus*
- **Role-Based Styling** – Different layout and colors for user vs assistant
- **Copy to Clipboard** – Copies message content with a toast alert
- **Hover Actions** – Icons shown on hover (copy, edit, like, etc.)
- **Assistant Avatar** – Displays logo icon for assistant messages
- **Scrollable Output** – Assistant message container supports vertical scrolling



THYNK UNLIMTED

config/db.js

MongoDB connection

- **Mongoose Import** – Load mongoose and prepare connection cache
- **Cache Check** – Return existing connection if available
- **Connection Promise** – Create new connection if not already initiated
- **Error Handling** – Catch and log connection errors
- **Return Connection** – Return active or newly created mongoose connection

THYNK UNLIMTED

context/ AppContext.js

Global State Management

- **Clerk Auth** – Use *useUser* and *useAuth* to access user and token
- **App Context** – Create *AppContext* and *useAppContext* hook
- **State Management** – Manage *chats* and *selectedChat* via *useState*
- **Create Chat** – *createNewChat*() calls API with Bearer token
- **Fetch Chats** – *fetchUserChat*() gets and sorts user chats
- **Auto Fetch** – *useEffect*() loads chats when user logs in
- **Provider Setup** – Wrap children with *AppContext.Provider* and values

THYNK UNLIMTED

models

- Chat.js
- User.js

THYNK UNLIMTED

Chat.js

- **Schema Definition** – Defines *ChatSchema* with name, messages, and userId
- **Messages Array** – Each message has *role*, *content*, *timestamp*
- **User Association** – *userId* links the chat to a specific user
- **Timestamps Enabled** – Automatically tracks createdAt and updatedAt
- **Model Export** – Reuses or creates *Chat* model for Mongoose

THYNK UNLIMTED

User.js

- **Schema Definition** – *UserSchema* with *_id*, *name*, *email*, *image*
- **Required Fields** – *_id*, *name*, and *email* are mandatory
- **Optional Field** – *image* field is not required
- **Timestamps Enabled** – Automatically adds *createdAt* and *updatedAt*
- **Model Export** – Uses existing or creates new *User* model with schema

THYNK UNLIMTED

.env requirement

- NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY
- CLERK_SECRET_KEY
- MONGODB_URI
- SIGNING_SECRET
- GOOGLE_GEMINI_API_KEY

THYNK UNLIMTED

Thank You

visit

https://deepseek-henna-nu.vercel.app/