Features Breakdown

Feature 1: Chart of Accounts (COA)

- 🔹 1.1 View all accounts
- 🔹 1.2 Add new account
  - 🔸 Required fields:
    - Name (e.g., "Cash", "Bank Loan")
    - Type (Asset / Liability / Equity / Expense / Revenue)
- 🔹 1.3 Edit account
- 🔹 1.4 Delete account

Feature 2: Journal Entry Module

- 🔹 2.1 View all journal entries
  - View all journal entry from a page and user can view journal entry from account page.
- 🔹 2.2 Create journal entry
  - 🎯 Requirements:
    - Date (default: today)
    - Description
    - At least 2 lines (min one debit, one credit)
    - Total Debit === Total Credit

➕ **Each Line Includes:**

- Account selection (from COA)
- Type: Debit / Credit
- Amount

API Planing

1. Chart of Accounts APIs

- `GET /api/accounts`
  → Get all accounts

`POST /api/accounts`
→ Create new account

Body:

```
{
  "name": "Cash",
  "type": "Asset"}
```

`PATCH /api/accounts?id=1`

→ Update account info

Body (partial allowed):

```
{
  "name": "Cash in Hand"

}
```

- `DELETE /api/accounts?id=1`
  → Delete account

2. Journal Entry APIs

- `GET /api/journal-entries`
  → Get all journal entries

`POST /api/journal-entries`

→ Create new journal entry

Body:

```
{
  "date": "2024-05-29",         // optional
  "description": "Office Rent",
  "lines": [
    {
      "accountId": 1,
      "type": "debit",
      "amount": 5000
    },
    {
      "accountId": 2,
      "type": "credit",
      "amount": 5000
    }
  ]
```

- `}`

Validation:

- Minimum 2 lines
- At least one debit and one credit
- Total debit === total credit

File & Folder Structure

```
/app
  └── api/
    └── accounts/
      ├── route.ts          → GET & POST handlers
      └── [id]/
        └── route.ts        → PATCH & DELETE handlers

    └── journal-entries/
      ├── route.ts          → GET & POST handlers

/lib
  └── prisma.ts             → Prisma client instance
  └── validations/
    ├── account.schema.ts   → zod or custom validation for accounts
    └── journal.schema.ts   → validation for journal entries

/prisma
  └── schema.prisma         → Prisma schema

/types
  └── index.ts              → Shared TypeScript types

/utils
  └── helpers.ts            → Common utilities (e.g., sum, format, etc.)

/components
  └── AccountForm.tsx
  └── JournalEntryForm.tsx
  └── EntryList.tsx

/app/(pages)
  ├── accounts/
    └── page.tsx            → Account list + form
  └── journal-entries/
    └── page.tsx            → Journal entries list + create form

/styles
  └── globals.css

.env
```

next.config.js
package.json
tsconfig.json

Task Assignment

Junior Dev 1: Backend Developer

Responsibility:

API Development + Validation + Prisma Integration

Assigned Tasks:

- **Chart of Accounts (COA) APIs:**
  - `GET /api/accounts` – Get all accounts
  - `POST /api/accounts` – Create new account (validate `name`, `type`)
  - `PATCH /api/accounts?id=1` – Update account (partial fields)
  - `DELETE /api/accounts?id=1` – Delete account

- **Journal Entries APIs:**
  - `GET /api/journal-entries` – Get all entries
  - `POST /api/journal-entries` – Create entry with:
    - Validation:
      - At least 2 lines
      - At least 1 debit and 1 credit
      - Total debit === credit

- **Validation (Zod or custom):**
  - `lib/validations/account.schema.ts`

- `lib/validations/journal.schema.ts`

🔹 **Prisma & DB:**

- Setup `schema.prisma` (Accounts, JournalEntry, EntryLine models)
- Write & test all necessary Prisma queries
- File: `lib/prisma.ts`

Junior Dev 2: Frontend Developer

Responsibility:

UI Pages, Forms, and API Integration

Assigned Tasks:

🔹 **COA Frontend:**

- `/accounts/page.tsx`
  - Fetch and list all accounts
  - Add/edit form using `AccountForm.tsx`
  - Delete button with confirmation

🔹 **Journal Entries Frontend:**

- `/journal-entries/page.tsx`
  - List journal entries
  - Form to create new entry (`JournalEntryForm.tsx`)
    - Dynamic line items (Add/Remove line)
    - Show error if debit ≠ credit

Components:

- `AccountForm.tsx`
- `JournalEntryForm.tsx`
- `EntryList.tsx`

API Integration:

- Use `fetch()` or `axios` to call the API routes created by backend dev
- Basic form validation before sending to server
- Display error/success messages

Coordination Point:

- Backend dev must provide API response format & endpoint availability.
- Frontend dev should use those exact formats and keep backend in loop for data structure.