Excellent. Now that we have a good high-level understanding of the application's structure, data, and flow, let's create some more detailed documentation for your backend developer friend.

The most critical piece of information for a backend developer is the API contract—what data the frontend needs, when it needs it, and what actions it can perform. Based on my analysis, I've created a list of suggested RESTful API endpoints. This will serve as a clear guide for what the backend needs to build.

API Endpoint Suggestions

Here is a proposed API structure. It assumes a user authentication system will be implemented, so all endpoints would be protected and operate on the data of the currently logged-in user.

Feature	HTTP Method	Endpoint	Description	Request Body (JSON)	Success Response (JSON)
Notes					
Get All Notes	`GET`	`/api/notes`	Retrieves all notes for the user.	(none)	`[Note]`
Get Single Note	`GET`	`/api/notes/:id`	Retrieves a single note by its ID.	(none)	`Note`
Create Note	`POST`	`/api/notes`	Creates a new note. The backend should generate the `id` and `date`.	`{ title: string, content: string, isMarkdown: boolean }`	`Note` (with new id/date)
Update Note	`PUT`	`/api/notes/:id`	Updates an existing note. The backend should update the `date`.	`{ title: string, content: string, isMarkdown: boolean }`	`Note` (updated)

Feature	HTTP Method	Endpoint	Description	Request Body (JSON)	Success Response (JSON)
Delete Note	`DELETE`	`/api/notes/:id`	Deletes a note by its ID.	(none)	`{ message: "Note deleted" }`
Settings					
Get Settings	`GET`	`/api/settings`	Retrieves the user's settings.	(none)	`Settings`
Update Settings	`PUT`	`/api/settings`	Updates the user's settings.	`Settings`	`Settings` (updated)
Data Mgmt					
Export Notes	`GET`	`/api/notes/export`	Exports all notes as a JSON file.	(none)	`[Note]`
Import Notes	`POST`	`/api/notes/import`	Imports notes from a JSON file.	`[Note]`	`{ message: "Import successful" }`
AI Feature					
Get Summary	`POST`	`/api/ai/summary`	Triggers AI summary generation for all notes.	(none)	`{ summary: string }`

Next, I'll outline the data validation rules. This ensures data integrity and prevents errors.

Data Validation and Schema

This builds upon the Data Model diagram to provide clear rules for the backend.

`Note` **Object:**

Field	Type	Validation Rules	Description
`id`	`Number` / `UUID`	Required, Unique, Generated by backend	Primary key for the note.
`title`	`String`	Optional, Max length: 255 chars	The title of the note.
`content`	`String`	Required, Cannot be empty	The main body of the note.
`date`	`String` (ISO 8601)	Required, Valid date format, Generated by backend	The creation/last modified date.
`isMarkdown`	`Boolean`	Required, Must be `true` or `false`	Flag to indicate if content is Markdown.
`userId`	`Number` / `UUID`	Required, Foreign Key	Associates the note with a user.

`Settings` Object:

Field	Type	Validation Rules	Default Value
`theme`	`String`	Required, Must be 'light' or 'dark'	'dark'
`font`	`String`	Required, Must be 'handwriting', 'serif', or 'sans'	'handwriting'
`userId`	`Number` / `UUID`	Required, Foreign Key	Links settings to a user.

These diagrams and tables should give your friend a very solid foundation for building the backend. Is there anything else you'd like me to generate?