Firstly, congratulations on having such a well-thought-out project idea. It's evident that you've poured a lot of thought into it and it serves a genuine purpose, based on your own experiences. This personal connection will likely make it even more compelling to potential employers.

**1. Advice on MVP Scope:**

Considering the purpose of your app as a portfolio project to land a job and keeping in mind the ADHD challenges, it's essential to focus on a Minimum Viable Product (MVP) that shows the core features of your project without overwhelming you.

For the MVP, I'd recommend:

1. **Authentication:** This is already part-done and is a fundamental feature. Completing it gives credibility to your project.
2. **Dashboard with Project Management:** Allowing users to create projects and tasks is a core feature based on your description. The UI here can leverage **shadcn** components.
3. **Notes Feature:** This seems relatively straightforward and would provide tangible value. Start with text-based notes. You can consider enhancing them with images in later versions.
4. **Task Setter or Goal Setter:** As this was partially complete, integrating it into the new project should be relatively direct.
5. **Calculators & Unit Conversion:** If these were already implemented, integrating them into the new design should be a priority, given they're industry-specific and show off a unique aspect of your app.

For the MVP, consider postponing:

1. **Cost Tracking & Estimate Generator:** While these are fantastic features, they are complex and could take a considerable amount of time.
2. **Extended Media in Notes:** Images in notes, while valuable, can be added in a later version after the MVP is stable.

**2. Roadmap for GitHub:**

Here's a sample roadmap that outlines the phases of your project, starting with the MVP:

**Phase 1: MVP**

* **Authentication**
  + Complete Login & Registration Functionality.
  + JWT Authentication & Authorization.
* **Dashboard**
  + Project creation.
  + Task & sub-task creation.
* **Notes Feature**
  + Text-based notes linked to projects.
* **Task Setter/Goal Setter**
  + CRUD operations for tasks/goals.
* **Calculators & Unit Conversion**
  + Reintegrate the already developed calculators.
  + Unit conversion feature.

**Phase 2: Enhancements**

* **Cost Tracking**
  + User input for building & business costs.
  + Track material and vendor costs.
  + Visual representation of cost increase over time.
* **Estimate Generator**
  + Contact list integration.
  + Previous customer data & billing.
  + Suggested price increases based on cost tracking data.
* **Notes Enhancement**
  + Ability to add images to notes.

**Phase 3: Advanced Features & Optimizations**

* **Mobile or PWA version**
  + Ensure mobile responsiveness.
  + Research and possibly develop a PWA version for easier mobile access and functionality like adding pictures directly from the phone.
* **User Feedback & Iterations**
  + Collect user feedback.
  + Implement changes based on the feedback.
* **Optimizations & Performance Enhancements**
  + Refactor code for better performance.
  + Improve UI/UX based on user behavior and feedback.

This roadmap can be expanded upon or contracted based on the development pace and any new features you think of along the way. It's meant to be dynamic, but having a clear structure will undoubtedly help guide the development process.

Now, with the roadmap in place and the MVP features identified, I'm ready to assist you with breaking down the next actionable steps when you're ready. Let me know the additional details you'd like to provide!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

questions about JWT and authentication:

-I know there an issue with JWT auth like mine is that the tokens expire, so users will need to be issued new tokens eventually.

-Is JWT auth like this used in backends of professional production apps?

-If not, are there alternatives I would be bettor off using, such as O auth or google auth?

-if you think im better off sticking with this JWT for my current MVP, is it possible to add some code that reissues tokens, and if so, how complicated is it and how long would it take?

1. **JWT Expiry**:
   * Yes, JWTs often have an expiration time, which is a security measure to ensure that if a token is compromised, it can't be used indefinitely. When a token expires, the user needs to be re-authenticated to get a new token.
2. **Is JWT used in professional production apps?**
   * Absolutely! JWT is widely used in many professional production applications. It's a standard method for securely transmitting information between parties as a JSON object. Its stateless nature makes it particularly useful for scalable applications.
3. **Alternatives to JWT**:
   * OAuth and Google Auth (which is a form of OAuth) are authentication protocols, whereas JWT is a token format. They can be used together. For example, when you use Google Auth, the response can be a JWT.
   * OAuth is more about authorizing third-party applications to access user data without exposing user credentials. If your application doesn't need third-party access, then implementing OAuth might be overkill for an MVP.
   * Google Auth or other social logins can be a great addition to your app as they provide users with a convenient way to sign in without creating a new account. However, they are not replacements for JWT but can be used in conjunction with it.
4. **Reissuing Tokens**:
   * There are a couple of common strategies to handle token expiration:
     1. **Short-lived Access Token + Long-lived Refresh Token**: The application issues two tokens: a short-lived access token (e.g., 15 minutes) and a long-lived refresh token (e.g., 7 days). When the access token expires, the application uses the refresh token to get a new access token. If the refresh token expires, the user needs to log in again. This strategy combines the security of short-lived tokens with the user experience of long-lived sessions.
     2. **Sliding Sessions**: Every time the user makes a request with a valid token, issue them a new token. This way, as long as the user is active, they won't be logged out. However, if they're inactive and the token expires, they'll need to log in again.
   * Implementing either strategy isn't overly complex, but it does require careful handling to ensure security. Depending on your familiarity with the concepts, it could take a few hours to a couple of days to implement and test thoroughly.

**Recommendation for Your MVP**: Given the context of your project and the goal to expedite the MVP, I'd recommend sticking with JWT for now. It's a robust and widely-accepted method for handling authentication. If you're concerned about token expiration, you might consider implementing the "Short-lived Access Token + Long-lived Refresh Token" strategy, as it offers a good balance between security and user experience. However, if you feel that this might slow down your MVP development, you can initially set a longer expiration time for the JWT (e.g., 24 hours) and then implement the refresh token mechanism in the next phase of your roadmap.

okay, so token expiration won't have any effects on this app being used down the road, such as users needing to reregister??

ChatGPT

Correct! Token expiration will not require users to re-register. The expiration of a token only means that the token itself is no longer valid for authentication. The user's account and all associated data remain intact in the database.

When a token expires, the user will simply need to log in again to obtain a new, valid token. This is a common practice in many applications to ensure security. The user's credentials (username, password, etc.) and any other data associated with their account remain unaffected by token expiration.

So, to reiterate, token expiration will not have any adverse effects on the app being used down the road. Users will not need to re-register; they will only need to log in again if their token has expired.  
  
Thank you for your patience. I've reviewed the relevant files from your repository. Here's what happens when a user registers or logs in:

1. **Registration**:
   * The **registerUser** function in **userController.js** first checks if the user already exists in the database using the provided email.
   * If the user doesn't exist, it hashes the password and creates a new user in the database.
   * After successfully creating the user, it generates a JWT token using the **generateToken** function and sends it back in the response.
2. **Login**:
   * The **loginUser** function in **userController.js** checks if the provided email exists in the database.
   * If the email exists and the password matches, it generates a JWT token using the **generateToken** function and sends it back in the response.

Both the registration and login processes use the same **generateToken** function to create the JWT token, which has an expiration time of 30 days (**expiresIn: '30d'**).

To answer your question: The token generated during registration will expire after 30 days, just like the one generated during login. However, this expiration **does not affect the user's registration or account in any way**. The user's account remains active and intact in the database. When the token expires, the user will simply need to log in again to obtain a new token. They won't need to re-register or create a new account.

In summary, your current setup ensures that token expiration only requires users to log in again, and it won't have any effects on their registration or account status.