Technologies

Front End	Back End	Devops
Framework: Next.js	**Framework:** Next.js API Routes	**CI/CD:** GitHub Actions
Styling: Tailwind CSS	**Database:** PostgreSQL	**Containerization:** Docker (if needed)
Component Library: Geist UI	**ORM:** Drizzle ORM	
	Authentication: NextAuth.js	
	Password Hashing: bcrypt-ts	

Standards and Best Practices

Front-End:

1. **Component Structure:**
   - Use a modular structure to keep components reusable and maintainable.
   - Utilize Next.js's file-based routing to organize pages and API routes.
2. **Styling:**
   - Use Tailwind CSS for utility-first styling.
   - Follow a consistent design system, leveraging Geist UI components for UI consistency.
3. **Error Handling:**
   - Implement error boundaries to catch JavaScript errors anywhere in the component tree.
   - Provide user-friendly error messages and feedback.
4. **Form Handling:**
   - Use form libraries like Formik for handling form state and validation.
   - Provide immediate feedback on validation errors.
5. **Type Safety:**
   - Use TypeScript to enforce type safety and reduce runtime errors.

Back-End:

1. **API Design:**
   - Follow RESTful principles for API design.
   - Use descriptive and consistent naming for endpoints and HTTP methods.
2. **Authentication:**
   - Implement JWT for stateless authentication using NextAuth.js.
   - Use secure password hashing (e.g., bcrypt-ts).
3. **Database:**
   - Use PostgreSQL for a robust relational database solution.
   - Structure data using schemas and ensure data validation with Drizzle ORM.
4. **Error Handling:**
   - Implement global error handling in API routes.
   - Return meaningful error messages and status codes.
5. **Security:**
   - Sanitize inputs to prevent SQL injection and other attacks.
   - Use HTTPS for secure data transmission.
   - Implement rate limiting to prevent abuse.

DevOps:

1. **CI/CD:**

    - Use GitHub Actions to automate testing and deployment pipelines.
    - Ensure that every commit triggers automated tests and builds.
2. **Containerization:**
    - Use Docker for consistent development, testing, and production environments.
    - Define services in Docker Compose for easier orchestration if necessary.
3. **Version Control:**
    - Use Git for source code management.
    - Follow Git Flow branching strategy for structured workflow.
4. **Code Quality:**
    - Use ESLint and Prettier for maintaining code quality and consistency.
    - Perform code reviews to ensure adherence to coding standards.
5. **Documentation:**
    - Use JSDoc for inline code documentation.
    - Maintain comprehensive README files for repositories.