**SmartTasker – Full-Stack Task Management Application**

**Project Overview**

SmartTasker is a full-stack task management application aimed at demonstrating modern web development practices to potential employers. It will showcase a robust architecture featuring secure user authentication, real-time collaborative updates, responsive design, and fully functional RESTful API endpoints. The goal is to build, test, and deploy a high-quality application—all within one week.

**Functional Requirements**

1. **User Authentication & Authorization**
   * **Secure Registration/Login:** Users must be able to register, log in, and log out securely.
   * **Role-Based Access:** Implement role-based control (admin vs. regular user), where admin users have broader permissions, such as managing all tasks.
2. **Task Management**
   * **CRUD Operations:** Provide full Create, Read, Update, and Delete functionalities for tasks.
   * **Task Details:** Allow users to assign tasks, set deadlines, define priorities, and mark completion statuses.
   * **Assignment & Tracking:** Enable tasks to be assigned to different users and track progress over time.
3. **Real-Time Collaboration**
   * **Live Updates:** Integrate real-time notifications and updates (e.g., using Socket.IO) so that any change in task status immediately reflects on all connected clients.
   * **Communication:** Offer notifications for task assignments, updates, and changes as they occur.
4. **Responsive Front-End**
   * **User Interface:** Develop a modern, mobile-first design using React.js.
   * **Accessibility:** Ensure the design adapts to various screen sizes (desktop, tablet, mobile) and follows basic accessibility guidelines for optimal usability.
5. **Reporting & Metrics**
   * **Basic Analytics:** Generate simple statistics such as completed vs. pending tasks.
   * **Filtering/Sorting:** Allow users to filter and sort tasks based on priority, status, or assignee.

**Non-Functional Requirements**

1. **Performance**
   * **Rapid Response:** The application should respond to most user interactions within 2 seconds.
   * **Real-Time Speed:** Real-time updates need to be pushed to users in under 500 milliseconds.
2. **Scalability**
   * **Concurrent Users:** Design to efficiently support up to 1000 concurrent users.
   * **Optimized Data Handling:** Use efficient database queries that scale for large datasets (e.g., 10,000+ tasks).
3. **Security**
   * **Data Protection:** Use HTTPS for secure data transmission and protect user credentials with industry-standard hashing (e.g., bcrypt).
   * **Prevent Abuse:** Implement input validation and rate limiting to safeguard the system.
4. **Maintainability**
   * **Clean Codebase:** Organize code into distinct, modular sections for the frontend, backend, and database layers.
   * **Documentation & Testing:** Include comprehensive documentation (such as a well-structured README and Swagger for API docs) and a testing suite that covers at least 80% of the code.
5. **Availability**
   * **Reliable Uptime:** Deploy the application on a cloud platform targeting at least 99.9% uptime.
   * **CI/CD Pipelines:** Set up continuous integration and deployment workflows for seamless, error-free updates.
6. **Usability**
   * **User-Friendly:** Design an intuitive user interface with clear navigation, helpful tooltips, and accessible error messages.
   * **Consistent UX:** Ensure that users receive immediate, useful feedback when actions succeed or fail.

**One-Week Project Schedule**

**Day 1: Initialization**

* **Setup:** Create the GitHub repository; initialize the project architecture for both the frontend (using create-react-app) and backend (using Express.js).
* **Environment:** Set up your development environment and database connections.

**Day 2: Authentication & Basic Backend**

* **User System:** Develop secure user registration and login functionality using JWT.
* **Middleware:** Implement role-based access control.
* **Backend Skeleton:** Establish basic CRUD endpoints for task management.

**Day 3: Task Management Operations**

* **CRUD Implementation:** Flesh out full Create, Read, Update, Delete operations for tasks.
* **Database Integration:** Set up the database models (using MongoDB and Mongoose) and ensure data persistence.

**Day 4: Real-Time Features**

* **Live Updates:** Integrate Socket.IO (or similar technology) to facilitate real-time updates and notifications.
* **Syncing:** Ensure that any changes in task data instantly reflect on the user interface.

**Day 5: Frontend Development & Integration**

* **UI/UX:** Develop the main user interface components (task list, forms, dashboard) using React.js.
* **Data Binding:** Connect the frontend with backend APIs to allow dynamic data flow and interactivity.

**Day 6: Testing & Optimization**

* **Testing:** Write and run unit tests (with Jest) and API tests (using Supertest) to cover all functionalities.
* **Optimization:** Tweak database queries and front-end performance to enhance responsiveness.

**Day 7: Deployment & Finalization**

* **Deployment:** Deploy the application to a cloud platform (Heroku, AWS, etc.) and configure the CI/CD pipeline.
* **Documentation & Review:** Finalize all documentation (README, API docs with Swagger, architecture diagrams) and perform a final round of polishing before the official project release.

**Next Steps & Accountability**

I’ll be checking in with you at the close of each day to assess progress, conduct code reviews, and ensure we’re aligning with the deadlines. When you push code to GitHub, I’ll evaluate it, ask questions about your design decisions, and provide constructive feedback to keep the momentum going. Remember, this is a sprint, so there’s no room for delays—consistency, focus, and meeting daily targets is key.

Now, let’s get this show on the road. Start by setting up your environment and initializing your repository. I'll expect a status update by the end of Day 1. No excuses—let’s do this!

This is your complete project description and schedule. Time to execute and bring SmartTasker to life!