Guardian – A DSA Problem-Solving Companion Web App

---

1. Title of the Project

Guardian – A DSA Problem-Solving Companion Web App

---

2. Introduction of the Project

1. **Background & Motivation**
2. Data Structures and Algorithms (DSA) form the backbone of efficient software development, competitive programming, and technical interview success.
3. Many learners struggle to bridge the gap between understanding theoretical concepts and applying them to real-world problems.
4. Traditional solution repositories often provide full code, which can lead to rote memorization rather than genuine comprehension.
5. **Problem Statement**
6. Learners frequently get overwhelmed by complex problem statements and lack a step-by-step guide to derive solutions independently.
7. There is a need for a tool that nudges learners in the right direction without handing them complete answers.
8. **Proposed Solution**
9. The “Guardian” web app offers progressive hints, conceptual explanations, and strategic approaches tailored to each problem.
10. By mimicking the guidance of an experienced mentor, the tool helps users internalize problem-solving patterns, fostering deeper learning and confidence.

---

3. Objective of the Project

- **Guided Learning** – Provide incremental hints keyed to user input, helping users unravel complex problems in manageable steps.
- **Promote Analytical Thinking** – Encourage users to derive algorithms by understanding underlying principles rather than copying solutions.
- **Accessible Platform** – Deliver a responsive web interface compatible with desktops, laptops, tablets, and mobile browsers.
- **Customization & Scalability** – Support user profiles, track progress, and adapt hint levels over time.
- **Interview Preparation** – Include a curated set of problems categorized by difficulty and topic for focused practice.
- **Feedback Mechanism** – Allow users to rate hints and request additional guidance, enabling continuous improvement of hint quality.

---

4. Hardware and Software Requirements of the Project

4.1 Hardware Requirements

Component	Minimum Specification	Recommended Specification
Development Workstation	Intel Core i3, 4GB RAM, 128GB SSD	Intel Core i5/i7, 8GB+ RAM, 256GB SSD
Deployment Server (Cloud or On-Prem)	1 vCPU, 2GB RAM, 20GB Storage	2 vCPU, 4GB RAM, 50GB Storage
Network	2 Mbps broadband	10 Mbps or higher
Client Devices	Any device with a modern browser	Same as minimum

4.2 Software Requirements

- **Operating Systems (for Development & Deployment):** Windows 10/11, macOS Catalina or later, Ubuntu 20.04 LTS or higher.
- **Frontend Technologies:**
- React.js (v18+)
- Tailwind CSS (v3+) for utility-first styling
- Axios for HTTP requests
- **Backend Technologies:**
- Node.js (v16+) with Express.js (v4+) framework
- JSON Web Tokens (JWT) for secure authentication
- **Database:**
- MongoDB (v6+) hosted locally or via a cloud service (e.g., MongoDB Atlas)
- **Development Tools:**
- Git (v2+) for version control
- Visual Studio Code (or equivalent IDE)
- Postman or Insomnia for API testing
- **Browser Compatibility:** Latest versions of Chrome, Firefox, Edge, and Safari
- **Build Tools & Packages:**
- npm or yarn for package management
- ESLint and Prettier for code linting and formatting

---

5. Analysis of the Project

5.1 System Architecture

The application follows a **three-tier architecture**: 1. **Presentation Layer** (Client-side)
- React-based single-page application (SPA) delivers a dynamic user interface.
- Components include problem submission form, hint display panel, user dashboard, and progress tracker.
2. **Application Layer** (Server-side)
- RESTful API built with Express.js handles business logic, authentication, and request routing.
- Hint Engine module: Matches incoming problem identifiers or keywords to a curated hint database.

3. **Data Layer** (Database)
- MongoDB stores user profiles, problem metadata, hint sets, and user activity logs.

5.2 Functional Workflow

1. **User Registration & Login**
2. New users sign up with email and password; JWT tokens manage sessions.
3. **Problem Submission**
4. Users enter a problem title or description; the frontend sends an API call to the Hint Engine.
5. **Hint Generation**
6. Server queries MongoDB for matching problem hints; returns the first hint level.
7. **Progressive Guidance**
8. Users can request additional hints; each successive hint reveals deeper conceptual steps.
9. **Completion & Feedback**
10. Once a user solves or views all hints, they can mark completion and rate hint usefulness.
11. **Dashboard Analytics**
12. Displays solved problem count, average hint uses per problem, and performance trends over time.

5.3 Non-Functional Requirements

- **Performance:** API response time $\leq$ 200ms under typical load.
- **Scalability:** Modular design allows horizontal scaling of server instances.
- **Security:** HTTPS enforcement, input validation, and JWT-based authentication.
- **Usability:** Intuitive interface with accessible navigation and clear hint progression.
- **Maintainability:** Well-documented codebase, modular components, and CI/CD integration.

---

6. Enhancement and Limitations of the Project

6.1 Potential Enhancements

- **Multi-Domain Support:** Extend beyond DSA to include Database Design, Operating Systems, and System Design problem hints.
- **Machine Learning Integration:** Employ NLP techniques to auto-classify new problem statements and generate context-aware hints.
- **Mobile Application:** Develop Android and iOS native apps for offline hint caching and push notifications.
- **Analytics Dashboard:** Incorporate advanced data visualization for user performance, hint effectiveness, and topic-wise strengths/weaknesses.
- **Community Contributions:** Allow experienced users to submit and peer-review hint sets, moderated through a reputation system.

6.2 Current Limitations

Limitation	Impact	Workaround/Future Fix
Internet Dependency	Requires constant connectivity	Implement PWA capabilities for offline hint caching

Limitation	Impact	Workaround/Future Fix
Static Hint Repository	New problem hints must be manually curated	Introduce automated hint suggestion via ML
Single-Language Interface	Only supports English	Add multilingual localization support
No Real-Time Collaboration	Single-user experience only	Implement pair-programming or group study mode

---

7. Conclusion

The **Guardian Web App** addresses a critical gap in self-directed DSA learning by offering strategic, hint-driven guidance rather than full solutions. Its MERN stack foundation ensures a responsive interface, scalable backend, and flexible data management. While the current version excels at promoting analytical thinking, future enhancements—such as machine learning for hint generation and mobile support—could broaden its impact. By focusing on incremental learning and user-centric design, Guardian has the potential to become an indispensable tool for students and professionals preparing for technical assessments, academic coursework, and competitive programming.

End of Synopsis Document