■■ Architecture & Design Decisions – BrainyBursts (Quiz App)

1. Overall Architecture

- Modular React architecture with separation of concerns
- Pages: high-level screens (QuizPage, ResultsPage)
- Components: reusable UI elements (QuestionCard, ProgressBar, Select)
- Data: local fallback in questions.json
- Routing: React Router (/quiz, /results)

State Flow

- Load Questions → Track Current Question → Capture Answer → Update Score → Results Summary

2. Data Flow & State Management

- React Hooks (useState, useEffect)
- questions: array of quiz questions (API or local JSON)
- current: current question index
- answers: mapping questionId → selected option
- score: derived at the end
- difficulty: selected level
- timer: countdown per question
- Props for data passing into presentational components

3. Data Source

- Primary: Open Trivia DB API
- Fallback: local JSON file (/data/questions.json)
- Ensures robustness in offline/failed API scenarios

4. UI/UX Design Decisions

- Tailwind CSS for responsive, utility-first styling
- shadcn/ui for accessible dropdowns and UI components
- Lucide Icons for lightweight vector icons
- Responsive layout across desktop and mobile
- Accessibility: keyboard navigation, ARIA labels, focus states

5. Quiz Flow

- QuizPage: Loads 5–10 questions, one at a time
- Requires answer before moving forward
- Progress bar + timer
- ResultsPage: Calculates score, shows summary, restart option

6. Bonus Features

- Timer (30s per question) – auto-lock on timeout
- Difficulty levels (Easy, Medium, Hard)
- High scores persisted in localStorage
- Animations: hover scale, fade-ins
- Error handling with API fallback

7. Deployment

- Optimized for static hosting (Vercel/Netlify)
- Vite build tool for fast bundling

8. Design Trade-offs

- Chose Hooks over Redux/Zustand for simplicity
- Local JSON fallback provides offline reliability but static data
- Timer per question adds UX realism but increases state complexity

9. Conclusion

- Balanced simplicity and robustness
- Hooks-based state management, modular components
- Robust fallback, accessibility, error handling
- Meets all assignment requirements while being maintainable and user-friendly