Project Documentation

Project Structure

```
src/
├── api/                # API related files
│   ├── config.js       # API endpoints and payload definitions
│   └── index.js        # API implementation
├── components/         # React components
│   ├── common/         # Reusable UI components
│   └── features/       # Feature-specific components
├── containers/         # Container components and logic
├── hooks/              # Custom React hooks
└── utils/              # Utility functions
```

Code Guidelines

1. Component Structure

- Use functional components with hooks
- Follow single responsibility principle
- Keep components small and focused
- Extract reusable logic into custom hooks
- Use TypeScript-style JSDoc comments for props

```
/**
 * @param {Object} props
 * @param {string} props.title - Component title
 * @param {Function} props.onAction - Callback function
 */
const Component = ({ title, onAction }) => {
  // Component logic
};
```

2. API Integration

- All API calls are centralized in src/api/
- Use the provided payload types from api/config.js
- Handle errors consistently
- Use async/await for API calls

```
try {
  const data = await questionsAPI.fetchQuestion();
  // Handle success
} catch (error) {
  // Handle error
}
```

3. State Management

- Use hooks for local state (useState, useReducer)
- Extract complex state logic into custom hooks
- Keep state close to where it's used

- Use context for global state when needed

4. Component Locations

UI Components (src/components/)

- QuestionCard.jsx - Main question display
- Timer.jsx - Countdown timer
- Controls.jsx - Recording and submission controls
- VoiceRecorder.jsx - Voice recording interface

Containers (src/containers/)

- QuestionContainer.jsx - Question logic and state management

Hooks (src/hooks/)

- useVoiceToText.js - Voice recognition logic
- useSpeechRecognition.js - Speech recognition setup
- useRecognitionState.js - Recognition state management

Utils (src/utils/)

- recognitionSetup.js - Speech recognition configuration
- toast.js - Toast notification utilities

Best Practices

1. **File Organization**
    - One component per file
    - Use index files for cleaner imports
    - Group related components in feature folders
2. **Naming Conventions**
    - PascalCase for components
    - camelCase for functions and variables
    - kebab-case for file names
    - Use descriptive, meaningful names
3. **Code Style**
    - Use consistent formatting (ESLint/Prettier)
    - Write self-documenting code
    - Add JSDoc comments for complex functions
    - Use meaningful variable names
4. **Performance**
    - Memoize callbacks with useCallback
    - Memoize expensive calculations with useMemo
    - Use proper React key props
    - Avoid unnecessary re-renders
5. **Error Handling**
    - Use try-catch blocks for async operations

- Provide meaningful error messages
- Handle edge cases gracefully
- Show user-friendly error notifications

Component Usage Examples

```
// Using the Question component
import { QuestionView } from './components/QuestionView';
function App() {
  return <QuestionView />;
}
// Using API calls
import { questionsAPI } from './api';
const handleSubmit = async (answer) => {
  await questionsAPI.submitAnswer({
    questionId: 1,
    answer
  });
};
```

Future Development

1. Add new features:
    - Create new components in appropriate directories
    - Follow existing patterns and conventions
    - Update documentation as needed
2. Extend API:
    - Add new endpoints to api/config.js
    - Implement handlers in api/index.js
    - Document payload types
3. Add new hooks:
    - Place in hooks/ directory
    - Follow existing naming conventions
    - Add proper documentation

This documentation serves as a reference for maintaining consistency and quality in the codebase as it grows.