Tech Glossary Hub (React)

Project Documentation

Project Duration: 23 July 2026 – 26 August 2026

1. Introduction

Tech Glossary Hub (React) is a modern developer-focused glossary application built using React and Vite. The application provides an organized collection of technical terms across programming languages, frameworks, libraries, and web technologies.

The project is designed to make technical concepts easier to discover, understand, and reference through structured categories, detailed term pages, search, filtering, sorting, favorites, related terms, and responsive navigation.

The React edition uses a dark theme by default and follows a reusable, component-based architecture.

2. Project Overview

Tech Glossary Hub is a multi-version learning and reference project developed across different technologies. The current version documented here is the React edition.

The React application uses JSON-based glossary data, React Router for navigation, reusable components for the user interface, React Context for favorites, custom hooks for search and filtering, and browser Local Storage for persistence.

The project also provides a Versions page through which users can explore the React, HTML, Django, Angular, and Flask editions of Tech Glossary Hub.

3. Objectives

- Build a modern technical glossary using React and Vite.
- Organize programming concepts into technology-based categories.
- Provide beginner-friendly definitions, explanations, examples, tags, and related terms.
- Implement efficient search, filtering, and sorting for glossary content.
- Allow users to save and manage favorite terms.
- Create reusable and maintainable React components.

- Implement responsive layouts for desktop, tablet, and mobile screens.
- Maintain a dark-themed developer-focused interface.
- Practice React Router, Context API, custom hooks, JSON data management, and Local Storage.
- Maintain a clean Git history through phase-based development and commits.

4. Features

- Dark Theme by Default
- Responsive Navigation Bar and Footer
- Home Page with Hero, Search, Featured Categories, and Featured Terms
- Category Listing and Category Details
- Glossary Listing
- Detailed Term Pages
- Search Functionality
- Category and Difficulty Filtering
- Alphabetical Sorting
- Favorite Terms with Local Storage Persistence
- Related Terms
- About Page
- Contact Page
- Project Versions Page
- 404 and Invalid Route Handling
- Reusable Empty State Handling
- Responsive Design
- JSON-Based Data Architecture
- Production Build with Vite

5. System Architecture

The application follows a component-based React architecture. Routing, page-level components, reusable UI components, application state, data, hooks, utilities, and global styling are separated into dedicated folders.

Architecture Flow:

React Application

↓

main.jsx

↓

App.jsx

↓

AppRouter.jsx

↓

Pages + Shared Layout

↓

Reusable Components

↓

Context + Custom Hooks + Utilities

↓

JSON Data

Major Architectural Components

- Pages: Provide route-level screens such as Home, Categories, Glossary, Search, Favorites, About, Versions, Contact, and NotFound.
- Components: Provide reusable category, glossary, home, layout, search, contact, favorites, and common UI components.
- Context: FavoritesContext manages favorite-term state.
- Hooks: useGlossarySearch and useGlossaryFilters encapsulate search and filtering logic.
- Data: JSON files store glossary terms and categories, while glossaryData.js centralizes the combined dataset.
- Routes: AppRouter.jsx defines application routes.
- Utils: relatedTerms.js resolves related glossary entries.
- Styles: Global and component/page CSS provide the visual system and responsive behavior.

6. Technologies Used

Technology	Purpose

React 19	Frontend framework
Vite 7	Development server and production build tool
JavaScript ES6+	Application programming language
React Router DOM 7	Client-side routing and navigation
CSS3	Styling and responsive design
React Icons	Interface icons
JSON	Glossary and category data
Local Storage	Persistent favorite-term storage
Git	Version control
GitHub	Source code repository
Vercel	Production deployment

7. Database Design

The React edition does not use a traditional relational database. Glossary information is maintained through JSON files inside the src/data directory.

This data architecture is suitable for the current static/reference-oriented application because the glossary content is packaged with the frontend and consumed by React components.

Data Sources

- categories.json — category information.
- html.json — HTML glossary terms.
- css.json — CSS glossary terms.
- javascript.json — JavaScript glossary terms.
- react.json — React glossary terms.
- typescript.json — TypeScript glossary terms.
- python.json — Python glossary terms.
- node.json — Node.js glossary terms.
- next.json — Next.js glossary terms.
- vue.json — Vue glossary terms.
- tailwindcss.json — Tailwind CSS glossary terms.
- glossaryData.js — central module that combines glossary datasets.

Typical Glossary Record

A glossary term can contain identifying information, category information, difficulty, definition, explanation, examples, tags, and references to related terms. Related terms are resolved through their stored IDs.

8. Project Workflow

The project was developed incrementally through defined development phases. Each completed phase was intended to be committed to GitHub to maintain a clear development history.

1. Project setup with Vite and React.
2. Router and initial project structure.
3. Shared application layout, navigation, footer, and reusable common components.
4. Home page development.
5. Category listing and category details.
6. Glossary and term details.
7. Search implementation.
8. Filtering and sorting.
9. Favorites and Local Storage persistence.
10. Related terms.
11. About and Contact pages.
12. Error handling and 404 behavior.
13. Responsive design.
14. Performance optimization.
15. Testing and bug fixing.
16. Deployment preparation and production deployment.
17. Final project documentation.

9. Implementation Details

Routing

React Router is used to provide client-side navigation between application pages. Dynamic routes are used for category and term details.

Component Architecture

The interface is divided into reusable components to reduce duplication and make individual sections easier to maintain.

Search

Search functionality uses the centralized glossary dataset and the custom useGlossarySearch hook to locate matching terms.

Filtering and Sorting

Glossary filtering and sorting are handled through useGlossaryFilters. Category and difficulty selections can be combined with sorting operations.

Favorites

Favorites are managed through FavoritesContext and persisted using browser Local Storage so selected terms remain available after refresh.

Related Terms

Related glossary entries are represented through term relationships and resolved using the relatedTerms utility.

Error Handling

Invalid application routes display the NotFound page. Invalid category and term routes are redirected to the appropriate listing pages. Reusable empty-state handling is used for cases where no results or saved terms are available.

Responsive Design

The interface uses responsive CSS media queries to adapt layouts for desktop, tablet, and mobile screen sizes while preserving the default dark theme.

Versions Page

The Versions page presents the different Tech Glossary Hub implementations. React, HTML, and Angular live projects can be previewed through embedded frames when supported. Django and Flask versions are presented with direct-link messaging because they may not load reliably inside an embedded preview.

10. Learning Outcomes

- Improved practical knowledge of React component architecture.
- Hands-on experience with React Router and client-side navigation.
- Experience creating reusable UI components.
- Experience managing shared state with React Context.
- Experience developing custom React hooks.
- Improved understanding of JSON-based frontend data architecture.
- Experience implementing search, filtering, sorting, and related-content logic.
- Experience using Local Storage for client-side persistence.
- Improved responsive CSS and mobile-first interface development.
- Experience debugging React runtime and routing errors.
- Experience optimizing repeated data processing and component rendering.
- Experience preparing and testing a Vite application for production deployment.
- Experience maintaining structured Git history during phased development.

11. Conclusion

Tech Glossary Hub (React) demonstrates the development of a structured, responsive, and maintainable glossary application using modern React practices. The project combines reusable components, client-side routing, JSON-based data, shared state, custom hooks, search, filtering, sorting, favorites, related terms, error handling, and responsive design.

The project also represents the React stage of a broader multi-technology Tech Glossary Hub evolution, providing a practical foundation for learning and comparing different implementation approaches.

12. Project Links

GitHub Repository

https://github.com/charanepuri/tech-glossary-hub-react

Live Demo

https://tech-glossary-hub-react.vercel.app/

13. Author

Charan Teja EPURI

Aspiring Python Full Stack Developer

- Django Portfolio: https://portfolio-site-django.onrender.com
- GitHub: https://github.com/charanepuri
- React Portfolio: https://charan-react-portfolio.vercel.app
- LinkedIn: https://www.linkedin.com/in/charan-teja-972aa9231
- Flask Portfolio: https://flask-developer-dashboard-portfolio.onrender.com/

14. Project Timeline

The project was developed from 23 July 2026 to 26 August 2026, covering the planned implementation phases, refinements, testing, deployment preparation, and documentation.

Total development period: approximately 35 calendar days.