Complete Guide to URL Shortener React Application

Table of Contents

Project Overview

This is a **URL Shortener** application built with React that allows users to:

- Create shortened versions of long URLs
- Track click analytics for their URLs
- Manage their shortened URLs through a dashboard
- View graphical analytics of URL performance
- Register and login to save their URLs

The application has a **multi-subdomain architecture** where:

- Main app runs on the primary domain (www.domain.com)
- Shortened URLs work on a subdomain (url.domain.com)

Project Structure

```
url-shortener-react/
├── public/                 # Static assets
│   ├── images/            # Image files
│   ├── vite.svg           # Vite logo
│   └── _redirects         # Netlify redirects
├── src/                    # Source code
│   ├── api/               # API configuration
│   ├── assets/            # React assets
│   ├── components/        # React components
│   │   └── Dashboard/     # Dashboard-specific components
│   ├── contextApi/        # State management
│   ├── dummyData/         # Sample data
│   ├── hooks/             # Custom React hooks
│   ├── utils/             # Utility functions
│   ├── App.jsx            # Main App component
```

```
│   ├── AppRouter.jsx     # Routing configuration
│   ├── main.jsx          # Application entry point
│   └── PrivateRoute.jsx  # Route protection
├── index.html            # HTML template
├── package.json          # Dependencies and scripts
├── tailwind.config.js    # Tailwind CSS configuration
└── vite.config.js        # Vite build configuration
```

Technologies Used

Core Technologies

- **React 18.3.1** - Frontend framework
- **Vite** - Build tool and development server
- **React Router DOM 7.1.1** - Client-side routing
- **Axios 1.7.9** - HTTP client for API calls

UI and Styling

- **Tailwind CSS 3.4.17** - Utility-first CSS framework
- **Material-UI 6.3.0** - React component library
- **React Icons 5.4.0** - Icon library
- **Framer Motion** - Animation library

State Management and Data

- **React Query 3.39.3** - Server state management
- **React Context API** - Global state management
- **React Hook Form 7.54.2** - Form handling

Charts and Analytics

- **Chart.js 4.4.7** - Chart library
- **React ChartJS 2** - React wrapper for Chart.js

Utilities

- **Day.js 1.11.13** - Date manipulation
- **React Copy to Clipboard** - Copy functionality
- **React Hot Toast** - Notifications

Application Flow

User Journey

1. **Landing Page** - User sees the homepage with app description
2. **Registration** - New users can create an account
3. **Login** - Existing users can sign in
4. **Dashboard** - Authenticated users can:
    - View their shortened URLs
    - Create new shortened URLs
    - View analytics for each URL
    - See overall click statistics
5. **URL Redirection** - Shortened URLs redirect to original URLs

Technical Flow

1. **App Initialization** - React app starts and checks for subdomain
2. **Router Selection** - Based on subdomain, loads main app or redirect app
3. **Authentication Check** - Context API checks for stored JWT token
4. **Route Protection** - PrivateRoute component controls access
5. **Data Fetching** - React Query handles API calls and caching
6. **State Updates** - Context API manages global authentication state

---

File-by-File Explanation

Configuration Files

`package.json`

Purpose: Defines project dependencies, scripts, and metadata.

Key Dependencies:

- `react`, `react-dom`: Core React libraries
- `react-router-dom`: For routing between pages
- `axios`: For making HTTP requests to backend
- `react-query`: For data fetching and caching
- `tailwindcss`: For styling
- `chart.js`, `react-chartjs-2`: For analytics graphs

Scripts:

- `dev`: Start development server with Vite
- `build`: Build production version
- `lint`: Check code quality with ESLint

`index.html`

Purpose: The HTML template where React app mounts.

Key Elements:

- `<div id="root">`: Where React app renders
- `<script src="/src/main.jsx">`: Loads React app
- Meta tags for responsive design

`vite.config.js`

Purpose: Configuration for Vite build tool.

- Enables React plugin for JSX support
- Configures development server

`tailwind.config.js`

Purpose: Configuration for Tailwind CSS.

- Defines custom colors and gradients
- Sets up responsive breakpoints

Core Application Files

`src/main.jsx`

Purpose: Entry point of the React application.

What it does:

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'
import { ContextProvider } from './contextApi/ContextApi.jsx'
import { QueryClient, QueryClientProvider } from 'react-query'
```

Flow:

1. Creates a QueryClient for React Query
2. Wraps app in providers:
    - `QueryClientProvider` : Enables React Query throughout app
    - `ContextProvider` : Provides global state (authentication)
    - `StrictMode` : Enables React's strict mode for better development

`src/App.jsx`

Purpose: Main App component that determines which router to use.

Logic:

```
function App() {
  const CurrentApp = getApps(); // Determines main app or subdomain app
  return (
    <Router>
      <CurrentApp />
    </Router>
  )
}
```

How it works:

1. Calls `getApps()` helper function
2. Based on current domain/subdomain, returns appropriate router
3. Wraps in `BrowserRouter` for routing

`src/utils/helper.js`

Purpose: Contains utility functions for app logic.

Key Functions:

1. `getApps()` :
    - Checks current subdomain
    - Returns main app router for www/main domain
    - Returns subdomain router for URL redirects
2. `getSubDomain(location)` :
    - Extracts subdomain from URL

- Handles both localhost and production domains

Example:

- `www.example.com` → Returns main app
- `url.example.com` → Returns redirect app

`src/utils/constant.js`

Purpose: Defines which router to use for each subdomain.

```
export const subDomainList = [
    {subdomain:"www", app: AppRouter, main: true},
    {subdomain:"url", app: SubDomainRouter, main: false}
];
```

Routing System

`src/AppRouter.jsx`

Purpose: Main application routing configuration.

Routes Defined:

- `/` - Landing page (public)
- `/about` - About page (public)
- `/register` - Registration (public only - redirects to dashboard if logged in)
- `/login` - Login (public only - redirects to dashboard if logged in)
- `/dashboard` - User dashboard (private - requires authentication)
- `/s/:url` - URL redirection (public)
- `/error` - Error page
- `*` - 404 error page

Special Features:

- `hideHeaderFooter`: Hides navigation on redirect pages (`/s/*`)
- Uses `PrivateRoute` component for route protection
- Includes `Toaster` for notifications

`src/PrivateRoute.jsx`

Purpose: Protects routes based on authentication status.

Logic:

```
export default function PrivateRoute({ children, publicPage}) {
    const { token } = useStoreContext();

    if (publicPage) {
        return token ? <Navigate to="/dashboard" /> : children;
    }

    return !token ? <Navigate to="/login" /> : children;
}
```

How it works:

- `publicPage=true` : Only show if NOT logged in (login/register pages)
- `publicPage=false` : Only show if logged in (dashboard)
- Automatically redirects based on authentication state

State Management

`src/contextApi/ContextApi.jsx`

Purpose: Global state management using React Context API.

State Managed:

- `token` : JWT authentication token
- `setToken` : Function to update token

Features:

- Persists token in localStorage
- Automatically loads token on app start
- Provides authentication state globally

Usage Pattern:

```
const { token, setToken } = useStoreContext();
```

API Communication

`src/api/api.js`

Purpose: Configures Axios for backend communication.

```
import axios from "axios";

export default axios.create({
    baseURL: import.meta.env.VITE_BACKEND_URL,
});
```

Features:

- Sets base URL from environment variable
- All API calls use this configured instance

`src/hooks/useQuery.js`

Purpose: Custom hooks for data fetching using React Query.

Hooks Provided:

1. **`useFetchMyShortUrls(token, onError)`** :
    - Fetches user's shortened URLs
    - Sorts by creation date (newest first)
    - Requires authentication token
2. **`useFetchTotalClicks(token, onError)`** :

    - Fetches click analytics data
    - Transforms object data to array format
    - Used for dashboard graphs

React Query Benefits:

- Automatic caching
- Background refetching
- Loading states
- Error handling

Components

Landing Page (`src/components/LandingPage.jsx`)

Purpose: Homepage that introduces the application.

Features:

- Hero section with app description
- Call-to-action buttons
- Feature cards explaining benefits
- Responsive design with animations

Technologies Used:

- Framer Motion for animations
- Tailwind CSS for styling
- React Router for navigation

Authentication Components

`src/components/LoginPage.jsx`

Purpose: User login interface.

Features:

- Form validation using React Hook Form
- Email and password fields
- Error handling and loading states
- JWT token storage on successful login

Flow:

1. User enters credentials
2. Form validates input
3. API call to `/api/auth/public/login`
4. Token stored in Context and localStorage
5. Redirect to dashboard

`src/components/RegisterPage.jsx`

Purpose: User registration interface.

Features:

- Form with username, email, password
- Input validation
- API call to create new account

- Redirect to login on success

Navigation (`src/components/NavBar.jsx`)

Purpose: Main navigation bar.

Features:

- Responsive mobile menu
- Dynamic navigation based on authentication
- Logout functionality
- Active page highlighting

Conditional Rendering:

- Shows "SignUp" button if not logged in
- Shows "Dashboard" and "LogOut" if logged in

Dashboard Components

`src/components/Dashboard/DashboardLayout.jsx`

Purpose: Main dashboard interface for authenticated users.

Features:

- Analytics graph showing click trends
- List of user's shortened URLs
- "Create New Short URL" button
- Loading states and empty states

Data Sources:

- `useFetchMyShortUrls` : Gets user's URLs
- `useFetchTotalClicks` : Gets analytics data

`src/components/Dashboard/ShortenItem.jsx`

Purpose: Individual shortened URL item display.

Features:

- Shows original and shortened URL
- Click count and creation date
- Copy to clipboard functionality
- Individual analytics view
- Collapsible analytics graph

Interactions:

- Copy button copies shortened URL
- Analytics button fetches and displays click data
- External link opens shortened URL

`src/components/Dashboard/CreateNewShorten.jsx`

Purpose: Form to create new shortened URLs.

Features:

- URL validation
- API call to create short URL

- Auto-copy to clipboard on success
- Modal popup interface

Flow:

1. User enters long URL
2. Form validates URL format
3. API call to `/api/urls/shorten`
4. Generated short URL automatically copied
5. Modal closes and list refreshes

`src/components/Dashboard/Graph.jsx`

Purpose: Chart component for analytics visualization.

Features:

- Bar chart using Chart.js
- Shows clicks over time
- Responsive design
- Placeholder data when no analytics available

Utility Components

`src/components/TextField.jsx`

Purpose: Reusable form input component.

Features:

- Integrated with React Hook Form
- Validation for email, URL, password
- Error message display
- Consistent styling

Validation Types:

- Required fields
- Minimum length (passwords)
- Email format validation
- URL format validation

`src/components/ShortenUrlPage.jsx`

Purpose: Handles URL redirection.

Function:

```
useEffect(() => {
    if (url) {
        window.location.href = import.meta.env.VITE_BACKEND_URL + `/${url}`;
    }
}, [url]);
```

How it works:

1. Extracts short URL code from route parameter
2. Immediately redirects to backend for tracking
3. Backend handles the actual redirect to original URL

Component Relationships

Hierarchy Diagram

```
App
├── Router
    ├── AppRouter (Main App)
    │   ├── NavBar
    │   ├── Routes
    │   │   ├── LandingPage
    │   │   │   └── Card (multiple)
    │   │   ├── AboutPage
    │   │   ├── PrivateRoute (Login/Register)
    │   │   │   ├── LoginPage
    │   │   │   │   └── TextField
    │   │   │   └── RegisterPage
    │   │   │       └── TextField
    │   │   ├── PrivateRoute (Dashboard)
    │   │   │   └── DashboardLayout
    │   │   │       ├── Graph
    │   │   │       ├── ShortenUrlList
    │   │   │       │   └── ShortenItem (multiple)
    │   │   │       │       └── Graph (analytics)
    │   │   │       └── ShortenPopUp
    │   │   │           └── CreateNewShorten
    │   │   │               └── TextField
    │   │   ├── ShortenUrlPage
    │   │   └── ErrorPage
    │   ├── Footer
    │   └── Toaster
    └── SubDomainRouter (Redirect App)
        └── ShortenUrlPage
```

Data Flow

```
ContextApi (Global State)
    ↓ token, setToken
Components
    ↓ API calls with token
React Query Hooks
    ↓ cached data
Components render with data
```

Frontend-Backend Communication

Environment Variables

The app uses these environment variables:

- `VITE_BACKEND_URL` : Backend API base URL

- `VITE_REACT_FRONT_END_URL` : Frontend domain for generating short URLs

API Endpoints Used

Authentication Endpoints

1. **POST** `/api/auth/public/register`
    - Creates new user account
    - Body: `{username, email, password}`
    - Response: Success message
2. **POST** `/api/auth/public/login`
    - Authenticates user
    - Body: `{username, password}`
    - Response: `{token}` (JWT)

URL Management Endpoints

1. **GET** `/api/urls/myurls`
    - Fetches user's shortened URLs
    - Headers: `Authorization: Bearer {token}`
    - Response: Array of URL objects
2. **POST** `/api/urls/shorten`
    - Creates new shortened URL
    - Headers: `Authorization: Bearer {token}`
    - Body: `{originalUrl}`
    - Response: `{shortUrl, originalUrl, id}`
3. **GET** `/api/urls/totalClicks?startDate=X&endDate=Y`
    - Fetches click analytics for all user's URLs
    - Headers: `Authorization: Bearer {token}`
    - Response: Object with date->count mapping
4. **GET** `/api/urls/analytics/{shortUrl}?startDate=X&endDate=Y`
    - Fetches click analytics for specific URL
    - Headers: `Authorization: Bearer {token}`
    - Response: Object with date->count mapping

URL Redirection

1. **GET** `/{shortUrl}`
    - Redirects to original URL
    - Increments click count
    - No authentication required

Authentication Flow

```
1. User logs in → API returns JWT token
2. Token stored in Context API and localStorage
3. All subsequent API calls include:
   Headers: { Authorization: "Bearer " + token }
```

```
4. Backend validates token for protected routes
5. Token persists across browser sessions
```

Error Handling

- API errors trigger navigation to `/error` page
- Form validation prevents invalid API calls
- Toast notifications show success/error messages
- Loading states shown during API calls

State Management

React Context API

Global State:

- Authentication token
- User login status

Why Context API?:

- Simple authentication state
- No complex state transformations needed
- Easy to access from any component

React Query

Server State Management:

- User's shortened URLs
- Analytics data
- API call caching

Benefits:

- Automatic background refetching
- Loading and error states
- Data caching reduces API calls
- Optimistic updates

Local State

Component-level state:

- Form inputs (React Hook Form)
- UI toggles (modals, dropdowns)
- Loading states
- Copy status

Routing System

Route Types

Public Routes

- `/` - Landing page

- `/about` - About page
- `/s/:url` - URL redirection

Public-Only Routes (redirect if logged in)

- `/login` - Login page
- `/register` - Registration page

Private Routes (require authentication)

- `/dashboard` - User dashboard

Error Routes

- `/error` - Error page
- `*` - 404 page

Route Protection Logic

```
// For public-only routes (login/register)
if (publicPage) {
    return token ? <Navigate to="/dashboard" /> : children;
}

// For private routes (dashboard)
return !token ? <Navigate to="/login" /> : children;
```

Multi-Subdomain Architecture

Main App (www.domain.com)

- Full application with all features
- Uses `AppRouter` component
- Shows header and footer

Subdomain App (url.domain.com)

- Minimal redirect functionality
- Uses `SubDomainRouter` component
- No header/footer
- Only handles `/:url` route

Why This Architecture?:

- Clean, short URLs for sharing
- Separates redirect traffic from main app
- Better SEO and user experience

Authentication System

JWT Token Flow

1. **Login**: User provides credentials
2. **Token Generation**: Backend creates JWT token
3. **Token Storage**: Frontend stores in Context API + localStorage
4. **Token Usage**: Included in all authenticated API calls
5. **Token Persistence**: Survives browser refresh/restart

6. **Logout**: Token removed from storage and context

Route Protection

- `PrivateRoute` component wraps protected routes
- Checks authentication state from Context API
- Automatically redirects unauthorized users
- Prevents authenticated users from accessing login/register

Security Considerations

- Token stored in memory (Context) and localStorage
- All API calls use HTTPS (in production)
- Token expiration handled by backend
- No sensitive data stored in frontend

How to Run the Application

Prerequisites

- Node.js (version 18+)
- npm or yarn package manager
- Backend API server running

Environment Setup

Create a `.env` file in the root directory:

```
VITE_BACKEND_URL=http://localhost:8080
VITE_REACT_FRONT_END_URL=http://localhost:3000
```

Installation and Running

1. **Install Dependencies**:

```
cd url-shortener-react
npm install
```

2. **Start Development Server**:

```
npm run dev
```

3. **Build for Production**:

```
npm run build
```

4. **Preview Production Build**:

```
npm run preview
```

Development Workflow

1. Start backend server first
2. Start React development server
3. Access main app at `http://localhost:3000`
4. Test URL redirection with shortened URLs

Deployment Considerations

- Set correct environment variables for production
- Configure subdomain routing on hosting platform
- Set up SSL certificates for both main domain and subdomains
- Configure CORS on backend for frontend domain

Key React Concepts Used

Hooks Used

- `useState` : Component state management
- `useEffect` : Side effects and lifecycle
- `useContext` : Global state access
- `useNavigate` : Programmatic navigation
- `useLocation` : Current route information
- `useParams` : URL parameters
- `useForm` : Form handling (React Hook Form)
- `useQuery` : Data fetching (React Query)

React Patterns

- **Component Composition**: Building complex UIs from simple components
- **Conditional Rendering**: Showing/hiding elements based on state
- **Event Handling**: User interactions trigger state changes
- **Props Passing**: Data flow between parent and child components
- **State Lifting**: Moving state up to common parent components

Modern React Features

- **Functional Components**: All components use function syntax
- **Custom Hooks**: Reusable stateful logic (`useQuery.js`)
- **Context API**: Global state without prop drilling
- **Error Boundaries**: Error handling with React Query
- **Code Splitting**: Dynamic imports for better performance

This guide provides a complete understanding of how the URL Shortener React application works, from basic structure to advanced concepts. Each file serves a specific purpose in creating a modern, scalable web application with authentication, data management, and a great user experience.