Base Converter Pro

Program & Process Documentation

Advanced Number System Conversion Tool

10/2/2025

Table of Contents

1. Overview
2. Core Conversion Engine
3. Storage Module
4. Validation Module
5. State Management
6. UI Component System
7. Type System
8. Application Process Flow

1. Overview

Base Converter Pro is a modern web application designed for converting numbers between different numeral systems including Binary (Base 2), Octal (Base 8), Decimal (Base 10), and Hexadecimal (Base 16).

The application features:
• Real-time number conversion
• Input validation for each base system
• Persistent conversion history
• Modern, responsive user interface
• Complete offline functionality

2. Core Conversion Engine

The heart of the application - handles number system conversions

Key Features:

- Multi-base conversion (Binary, Octal, Decimal, Hexadecimal)
- Input validation for each base system
- Error handling and user feedback
- Decimal intermediary conversion approach

Implementation:

```
const convertNumber = (input: string, fromBase: number, toBase: number) => {
  // Step 1: Convert to decimal
  const decimal = parseInt(input, fromBase);

  // Step 2: Convert from decimal to target base
  return decimal.toString(toBase).toUpperCase();
};
```

3. Storage Module

Manages persistent data using browser localStorage

Key Features:

- Automatic history saving
- Data persistence across sessions
- JSON serialization/deserialization
- Error handling for storage operations

Implementation:

```
const saveHistory = (history: ConversionRecord[]) => {
  localStorage.setItem('conversionHistory', JSON.stringify(history));
};

const loadHistory = (): ConversionRecord[] => {
  const saved = localStorage.getItem('conversionHistory');
  return saved ? JSON.parse(saved) : [];
};
```

4. Validation Module

Ensures input integrity and prevents invalid conversions

Key Features:

- Character validation per base
- Regular expression patterns
- Real-time input checking
- User-friendly error messages

Implementation:

```
const isValidInput = (value: string, base: number): boolean => {
  const patterns = {
    2: /^[01]+$/,
    8: /^[0-7]+$/,
    10: /^[0-9]+$/,
    16: /^[0-9A-Fa-f]+$/
  };
  return patterns[base].test(value);
};
```

5. State Management

React hooks for managing application state

Key Features:

- useState for local state
- useEffect for side effects
- Reactive UI updates
- Efficient re-rendering

Implementation:

```
const [inputValue, setInputValue] = useState('');
const [fromBase, setFromBase] = useState(10);
const [toBase, setToBase] = useState(2);
const [result, setResult] = useState('');
const [history, setHistory] = useState<ConversionRecord[]>([]);
```

6. UI Component System

Reusable components with shadcn/ui

Key Features:

- Card-based layouts
- Responsive design
- Accessibility support
- Consistent design system

7. Type System

TypeScript interfaces for type safety

Key Features:

- Strong typing
- Interface definitions
- IntelliSense support
- Compile-time error checking

Implementation:

```
interface ConversionRecord {
  input: string;
  fromBase: number;
  output: string;
  toBase: number;
  timestamp: string;
}
```

8. Application Process Flow

Step-by-step execution of conversion operations:

Step 1: User Input

User enters a number and selects source base system

Step 2: Validation

Input is validated against allowed characters for selected base

Step 3: Conversion

Number is converted to decimal, then to target base

Step 4: Result Display

Converted value is displayed with formatting

Step 5: History Storage

Conversion record is saved to localStorage

Technology Stack

- React 18.3 - UI Library
- TypeScript - Type Safety
- Tailwind CSS - Styling
- shadcn/ui - Component Library
- Vite - Build Tool
- localStorage - Data Persistence
- Lucide React - Icons