Bank of React

Project Document

Assignment:	Assignment 3 - Bank of React
Start Date:	October 15, 2025
Due Date:	November 3, 2025, 11:59 PM
Project Duration:	3 weeks (19 days)
Team Size:	2 developers (David & Kevin)
Document Version:	1.0
Last Updated:	November 2, 2025

A React-based banking application with dual bank support, client-side routing, and authentication.

Contents

1 Feature Requirements

1.1 Core Features

1.1.1 Landing Page

- **Multi-Bank Access**: Landing page allowing users to choose between David's Bank and Kevin's Bank
- **Navigation**: Clean interface with links to both banking portals
- **User-Friendly Design**: Simple, intuitive card-based layout

1.1.2 Bank Portal Features (Per Bank)

- **Home Page**: Display account balance and navigation menu
- **User Profile**: Display user information (username, email, member since date)
- **Login System**: Form-based authentication with validation
- **Credits Management**: View credit history, add new credits
- **Debits Management**: View debit history, add new debits
- **Account Balance**: Dynamic calculation based on credits and debits

1.1.3 Authentication System

- **Login Required**: Users must authenticate before accessing bank features
- **Form Validation**: Username and password validation
- **Email Support**: Optional email field in login form
- **Separate Auth States**: Independent authentication for each bank
- **Automatic Redirect**: Unauthenticated users redirected to login

1.1.4 State Management

- **Separate Bank States**: Independent credits, debits, and balances for each bank
- **User Data Isolation**: Separate user profiles per bank
- **Independent Calculations**: Balance calculations separate for each bank
- **Transaction History**: Separate transaction histories maintained per bank

1.2 Technical Requirements

1.2.1 React Router Implementation

- Use of `BrowserRouter` for client-side routing
- Use of `Route` component for defining routes
- Use of `Link` component for navigation
- Use of `Redirect` component for programmatic navigation
- Route protection based on authentication state

1.2.2 React Components

- Component-based architecture
- Props passing for data flow
- State management in App component
- Lifecycle methods (`componentDidMount`)
- Event handlers for user interactions
- Controlled components for form inputs

1.2.3 Data Fetching

- API integration using `fetch`
- Async/await for asynchronous operations
- Error handling for API requests
- Data initialization on component mount

1.2.4 Component Organization

- Prefixed components (`kevin_` and `david_`)
- Shared core logic with different styling
- Modular component structure
- Reusable component patterns

1.3 Version Control Requirements

1.3.1 Git Workflow

- **Feature Branches**: Separate branch for each major feature
- **Pull Requests**: Merge feature branches via pull requests
- **Commit Practices**: Small, frequent commits with descriptive messages
- **Branch Naming**: Clear, descriptive branch names (e.g., feat/my-home-page, feat/user-auth-enhancements)

1.4 Deployment Requirements

- Website deployed to GitHub Pages
- Live URL accessible and functional
- README includes link to deployed site
- All features working in production environment
- Proper basename configuration for GitHub Pages

2 Application Architecture Description and Diagram

2.1 Architecture Overview

The Bank of React application follows a **React-based Single-Page Application (SPA) architecture** using React Router for client-side routing. The architecture emphasizes **component-based development**, **state management**, and **authentication-based route protection**.

2.2 System Components

2.2.1 Presentation Layer (React Components)

- **Landing Page**: Entry point with bank selection
- **Home Components**: Display account balance and navigation (separate for each bank)
- **User Profile Components**: Display user information (separate for each bank)
- **Login Components**: Authentication forms with validation (separate for each bank)
- **Credits Components**: Credit management interface (separate for each bank)
- **Debits Components**: Debit management interface (separate for each bank)
- **Account Balance Components**: Balance display (separate for each bank)

2.2.2 Routing Layer (React Router)

- **Router**: BrowserRouter wrapper for entire application
- **Routes**: Defined routes for all page views
- **Protected Routes**: Authentication-based access control
- **Navigation**: Link components for user navigation
- **Redirects**: Programmatic navigation after login

2.2.3 State Management Layer (App Component)

- **Kevin's Bank State**:
  - `kevinCredits`: Array of credit transactions
  - `kevinDebits`: Array of debit transactions
  - `kevinBalance`: Calculated account balance
  - `kevinAuthenticated`: Authentication status
  - `kevinUser`: User profile data
- **David's Bank State**:
  - `davidCredits`: Array of credit transactions
  - `davidDebits`: Array of debit transactions
  - `davidBalance`: Calculated account balance
  - `davidAuthenticated`: Authentication status
  - `davidUser`: User profile data

2.2.4 Business Logic Layer

- **Balance Calculation**: Separate functions for each bank
- **Transaction Management**: Add credit/debit functions per bank
- **Authentication Handlers**: Login functions per bank
- **API Integration**: Data fetching on component mount

2.3 Architecture Diagram

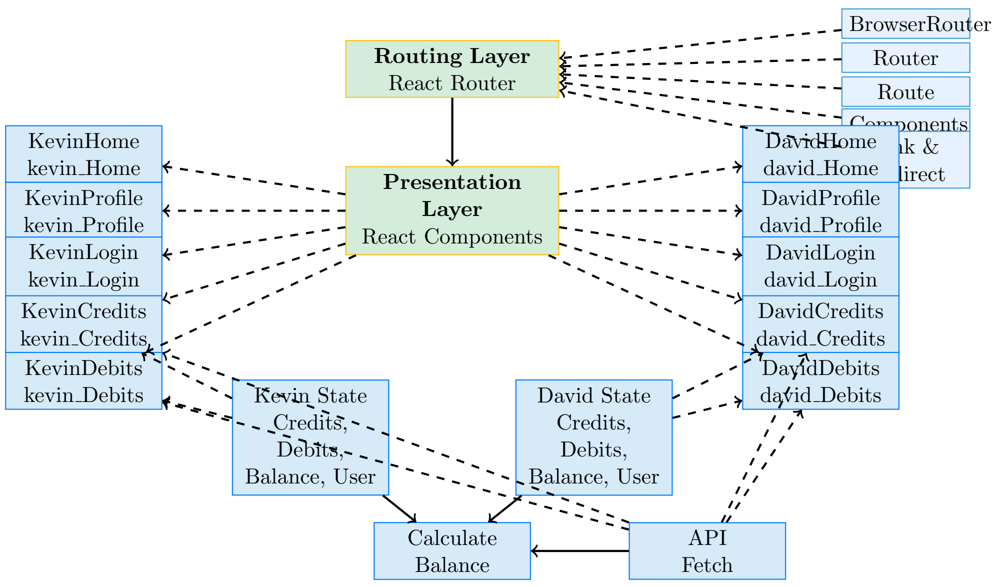

Figure 1: Bank of React Application Architecture

2.4 Data Flow

1. **User Navigation**: User selects bank from landing page
2. **Route Matching**: React Router matches URL to route definition
3. **Authentication Check**: System checks authentication state
4. **Component Rendering**: Appropriate component rendered (bank portal or login)
5. **State Access**: Component accesses relevant bank state from App
6. **User Interaction**: User performs actions (add transaction, login, etc.)
7. **State Update**: App component state updated via handler functions
8. **UI Update**: Component re-renders with new data

2.5 Technology Stack

- **React 17.0.2**: Component-based UI library
- **React Router 5.3.0**: Client-side routing
- **JavaScript (ES6+)**: Application logic

- **Fetch API**: HTTP requests for data
- **Git**: Version control
- **GitHub**: Repository hosting
- **GitHub Pages**: Static website hosting

2.6 Key Design Patterns

- **Component Composition**: Breaking UI into reusable components
- **State Lifting**: Centralized state management in App component
- **Unidirectional Data Flow**: Props down, events up
- **Route Protection**: Conditional rendering based on authentication
- **Separation of Concerns**: Logic separated from presentation
- **DRY Principle**: Shared logic with different styling per bank

3 Epics, User Stories, and Acceptance Criteria

3.1 Epic 1: Landing Page and Multi-Bank Access

3.1.1 User Story 1.1: Landing Page with Bank Selection

As a user
I want to see a landing page with options to access different banks
So that I can choose which banking portal to use
Acceptance Criteria:

- Landing page displays at root route (/)
- Page shows options for David's Bank and Kevin's Bank
- Each bank option has a clear call-to-action button
- Navigation links correctly route to respective bank portals
- Design is clean and user-friendly

3.2 Epic 2: Authentication System

3.2.1 User Story 2.1: Login Form with Validation

As a user
I want to log in with username and password
So that I can access my bank account
Acceptance Criteria:

- Login form includes username and password fields
- Form includes optional email field
- Username and password are required fields
- Validation errors display when fields are empty

- Error messages clear when user starts typing
- Form submission triggers authentication

3.2.2 User Story 2.2: Authentication Protection

As a user
I want to be required to log in before accessing bank features
So that my account is secure
Acceptance Criteria:

- Unauthenticated users accessing bank routes see login form
- Login sets authentication state for respective bank
- Authentication state is separate for each bank
- After login, users can access bank features
- Login redirects to appropriate bank home page

3.3 Epic 3: Home Page and Navigation

3.3.1 User Story 3.1: Home Page Display

As a user
I want to see my account balance on the home page
So that I can quickly view my financial status
Acceptance Criteria:

- Home page displays after successful authentication
- Account balance is prominently displayed
- Balance is calculated correctly (Credits - Debits)
- Balance shows 2 decimal places
- Navigation links to other pages are available

3.3.2 User Story 3.2: Navigation Menu

As a user
I want to navigate between different bank pages
So that I can access all bank features easily
Acceptance Criteria:

- Navigation links present on home page
- Links to User Profile, Login, Credits, and Debits
- Links use React Router's Link component
- Navigation works correctly and updates URL
- Users can return to home from other pages

3.4 Epic 4: User Profile

3.4.1 User Story 4.1: View User Profile

As a user
I want to view my profile information
So that I can see my account details
Acceptance Criteria:

- Profile page displays username
- Profile page displays email (if provided)
- Profile page displays member since date
- Information is styled and easy to read
- Link to return to home is available

3.5 Epic 5: Credits Management

3.5.1 User Story 5.1: View Credits History

As a user
I want to view all my credit transactions
So that I can see my income history
Acceptance Criteria:

- Credits page displays list of all credits
- Each credit shows description, amount, and date
- Credits from API are loaded on page load
- Amounts display with 2 decimal places
- Dates display in yyyy-mm-dd format

3.5.2 User Story 5.2: Add New Credit

As a user
I want to add new credit transactions
So that I can record income
Acceptance Criteria:

- Form to add new credit with description and amount fields
- Form validation ensures required fields are filled
- New credit added to credit list immediately
- New credit includes current date (yyyy-mm-dd)
- Account balance updates automatically after adding credit
- Form clears after successful submission

3.6 Epic 6: Debits Management

3.6.1 User Story 6.1: View Debits History

As a user
I want to view all my debit transactions
So that I can see my spending history
Acceptance Criteria:

- Debits page displays list of all debits
- Each debit shows description, amount, and date
- Debits from API are loaded on page load
- Amounts display with 2 decimal places
- Dates display in yyyy-mm-dd format

3.6.2 User Story 6.2: Add New Debit

As a user
I want to add new debit transactions
So that I can record expenses
Acceptance Criteria:

- Form to add new debit with description and amount fields
- Form validation ensures required fields are filled
- New debit added to debit list immediately
- New debit includes current date (yyyy-mm-dd)
- Account balance updates automatically after adding debit
- Form clears after successful submission

3.7 Epic 7: Account Balance Calculation

3.7.1 User Story 7.1: Dynamic Balance Calculation

As a user
I want to see my account balance calculated from credits and debits
So that I know my current financial status
Acceptance Criteria:

- Balance = Total Credits - Total Debits
- Balance updates automatically when transactions added
- Balance can be negative if debits exceed credits
- Balance displays with 2 decimal places
- Balance displayed on Home, Credits, and Debits pages

3.8 Epic 8: Dual Bank System

3.8.1 User Story 8.1: Separate Bank States

As a developer
I want to maintain separate state for each bank
So that banks operate independently
Acceptance Criteria:

- Kevin's and David's banks have separate credit arrays
- Kevin's and David's banks have separate debit arrays
- Kevin's and David's banks have separate balance calculations
- Transactions in one bank don't affect the other
- Each bank maintains independent authentication state

3.9 Epic 9: Code Quality and Version Control

3.9.1 User Story 9.1: Feature Branch Workflow

As a developer
I want to create feature branches for each feature
So that I can develop features in isolation
Acceptance Criteria:

- Each major feature has its own branch
- Branch names follow convention (feat/feature-name)
- Feature branches merged via pull requests
- Commit messages are descriptive and clear
- Code is well-organized and commented

4 Project Schedule Chart (Gantt Chart)

4.1 Project Timeline Overview

Project Duration: 3 weeks (19 days)
Start Date: October 15, 2025
Due Date: November 3, 2025, 11:59 PM
Team Size: 2 developers (David & Kevin)

4.2 Detailed Gantt Chart

4.3 Phase Breakdown

4.3.1 Phase 1: Planning & Setup (Days 1-3, Oct 15-17)

Tasks:

- Review assignment requirements
- Analyze starter code structure

- Set up GitHub repository
- Review React Router documentation
- Draft project document structure

Deliverables:

- Project document outline
- GitHub repository configured
- Starter code imported and reviewed

4.3.2 Phase 2: Landing Page & Routing (Days 3-6, Oct 17-20)

Tasks:

- Create LandingPage component
- Set up React Router
- Configure routes for both banks
- Test navigation between pages

Deliverables:

- Working landing page
- Router configuration
- Basic routing between banks

Branches:

- `feat/my-home-page`

4.3.3 Phase 3: Authentication System (Days 6-9, Oct 20-23)

Tasks:

- Create Login components (Kevin and David versions)
- Implement form validation
- Add authentication state management
- Implement route protection
- Test authentication flow

Deliverables:

- Working login forms
- Form validation
- Protected routes
- Authentication state management

Branches:

- `feat/user-auth-enhancements`

4.3.4 Phase 4: Credits & Debits (Days 9-14, Oct 23-28)

Tasks:

- Create Credits components (Kevin and David versions)
- Create Debits components (Kevin and David versions)
- Integrate API endpoints
- Implement add transaction functionality
- Implement balance calculation
- Test transaction flows

Deliverables:

- Working Credits pages
- Working Debits pages
- API integration complete
- Transaction addition working
- Balance calculation working

Branches:

- Feature branches for Credits and Debits

4.3.5 Phase 5: Dual Bank System (Days 14-16, Oct 28-30)

Tasks:

- Separate state management per bank
- Split components with prefixes
- Independent balance calculations
- Separate authentication states
- Test bank independence

Deliverables:

- Completely separate bank states
- Independent transaction histories
- Separate user profiles
- Verified bank isolation

Branches:

- `feat/user-auth-enhancements`

4.3.6 Phase 6: Testing, Documentation & Deployment (Days 16-19, Oct 30-Nov 3)

Tasks:

- Comprehensive testing of all features
- Cross-browser testing
- Code cleanup and refactoring
- Add code comments
- Complete project document
- Update README with team info and deployment link
- Deploy to GitHub Pages
- Final testing of deployed site
- Prepare submission materials (PDF conversion)

Deliverables:

- Fully tested application
- Clean, commented code
- Complete project document (PDF)
- Updated README
- Live website on GitHub Pages
- Submission ready for Brightspace

4.4 Milestones

- **October 17**: Project setup and planning complete
- **October 20**: Landing page and routing complete
- **October 23**: Authentication system complete
- **October 28**: Credits and Debits features complete
- **October 30**: Dual bank system complete
- **November 3**: Project deployed, tested, and submitted

4.5 Risk Management

4.5.1 Identified Risks

- **State Management Complexity**: Managing separate states for two banks
- **Route Protection Logic**: Ensuring proper authentication checks
- **API Integration Issues**: Handling API errors and loading states
- **Balance Calculation Errors**: Ensuring accurate financial calculations
- **Merge Conflicts**: Multiple feature branches may conflict
- **Time Constraints**: 3 weeks is a tight timeline

4.5.2 Mitigation Strategies

- **Incremental Development**: Build and test one feature at a time
- **State Isolation**: Clear separation of bank states
- **Error Handling**: Comprehensive error handling for API calls
- **Testing**: Test balance calculations with various scenarios
- **Small Commits**: Make frequent, small commits to minimize conflicts
- **Early Start**: Begin project as soon as assigned
- **Code Reviews**: Review pull requests carefully before merging

4.6 Resource Allocation

4.6.1 Team Structure (2-person team)

- **David**: Landing page, David's bank components, styling
- **Kevin**: Kevin's bank components, authentication, routing
- **Shared**: State management, API integration, testing, documentation

4.6.2 Recommended Task Division

- **Developer 1**: Landing page, routing setup, Credits/Debits components
- **Developer 2**: Authentication, state management, User Profile, balance calculations
- **Both**: Testing, code review, documentation, deployment

4.6.3 Required Tools

- **Text Editor/IDE**: VS Code or similar
- **Web Browser**: Chrome, Firefox, or Edge with DevTools
- **Node.js & npm**: For React development
- **Git**: For version control
- **GitHub Account**: For repository hosting and Pages
- **LaTeX Editor**: For project document (Overleaf, TeXShop, etc.)

5 Implementation Notes

5.1 React Router Implementation

5.1.1 Setting Up Router

- Import `BrowserRouter as Router` from `react-router-dom`
- Wrap application in `<Router>` component
- Use `basename="/Bank-of-React"` for GitHub Pages
- Define routes using `<Route>` components
- Use `exact` prop for precise path matching

5.1.2 Navigation

- Use `<Link to="/path">` for declarative navigation
- Use `<Redirect to="/path">` for programmatic navigation
- Pass props to routes using `render` prop
- Protect routes with conditional rendering based on auth state

5.2 State Management

5.2.1 App Component State Structure

```
state = {
  // Kevin's bank state
  kevinCredits: [],
  kevinDebits: [],
  kevinBalance: 0,
  kevinAuthenticated: false,
  kevinUser: {
    userName: 'Kevin User',
    email: 'kevin@example.com',
    memberSince: '11/22/99'
  },
  // David's bank state
  davidCredits: [],
  davidDebits: [],
  davidBalance: 0,
  davidAuthenticated: false,
  davidUser: {
    userName: 'David User',
    email: 'david@example.com',
    memberSince: '01/15/20'
  }
}
```

5.2.2 Balance Calculation

- Formula: `Balance = Total Credits - Total Debits`
- Use `reduce()` to sum credits and debits
- Round to 2 decimal places using `toFixed(2)`
- Update balance after each transaction addition
- Separate calculation functions for each bank

5.3 API Integration

5.3.1 Data Fetching

- Use `async/await` in `componentDidMount()`
- Fetch from credits API: `https://johnnylaicode.github.io/api/credits.json`

- Fetch from debits API: `https://johnnylaicode.github.io/api/debits.json`
- Handle errors with try-catch blocks
- Initialize both banks with fetched data independently

5.4 Component Organization

5.4.1 Component Naming Convention

- Kevin's components: `kevin_Home.js`, `kevin_Credits.js`, etc.
- David's components: `david_Home.js`, `david_Credits.js`, etc.
- Shared logic with different styling
- Independent state management per bank

5.5 Best Practices

- **Component Structure**: Keep components focused and single-purpose
- **State Management**: Lift state up to App component when shared
- **Event Handling**: Use arrow functions for class methods
- **Form Handling**: Use controlled components with value and onChange
- **Validation**: Validate inputs before updating state
- **Error Handling**: Implement try-catch for async operations
- **Routing**: Use exact prop for precise route matching
- **Styling**: Inline styles for component-specific styling
- **Comments**: Add comments for complex logic
- **Git Workflow**: Create feature branches for each major feature

6 API Endpoints

6.1 Credits API

- **URL**: `https://johnnylaicode.github.io/api/credits.json`
- **Method**: GET
- **Response**: JSON array of credit objects
- **Credit Object Structure**:
  - `id`: Unique identifier
  - `description`: Credit description
  - `amount`: Credit amount (number)
  - `date`: Date in yyyy-mm-dd format

6.2 Debits API

- **URL**: `https://johnnylaicode.github.io/api/debits.json`
- **Method**: GET
- **Response**: JSON array of debit objects
- **Debit Object Structure**:
  - `id`: Unique identifier
  - `description`: Debit description
  - `amount`: Debit amount (number)
  - `date`: Date in yyyy-mm-dd format

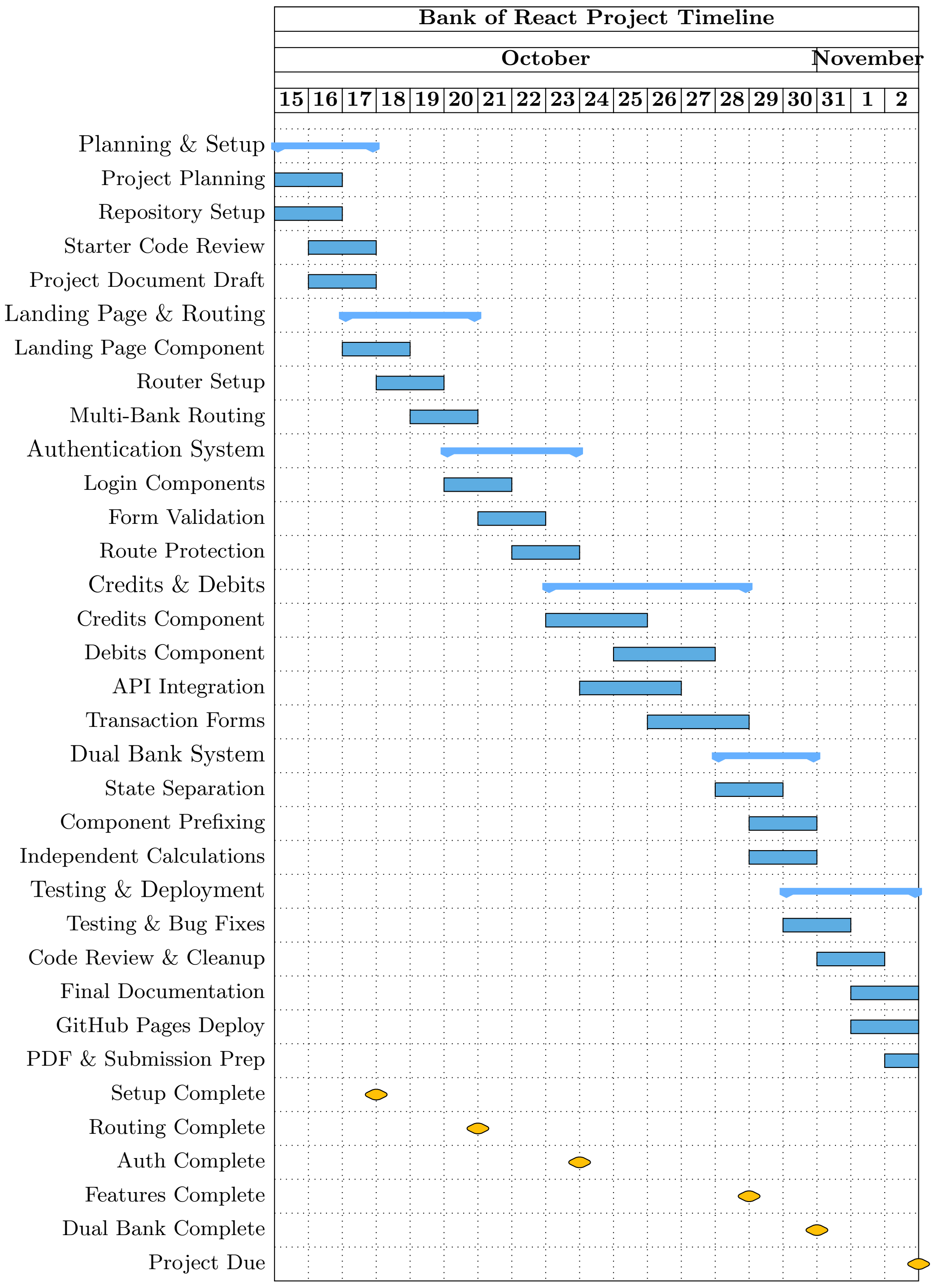

Figure 2: Project Timeline - Gantt Chart (Oct 15 - Nov 3, 2025)