**INT 331 PROJECT REPORT**

# NATIONAL ACCOUNTS DASHBOARD

## A Full-Stack MERN Application for India's Economic Data

---

**Submitted by:**

Ashutosh Mohanty

12307673

KO021 011

**Submitted to:**

Dr. Harpreet Kaur

School of Computer Science & Engineering

Lovely Professional University

**Academic Year:** 2024-25

---

**TABLE OF CONTENTS**



## 1. INTRODUCTION

### 1.1. OVERVIEW

The National Accounts Dashboard is a comprehensive full-stack web application developed using the MERN (MongoDB, Express.js, React, Node.js) technology stack. The application serves as a centralized platform for monitoring, analysing, and visualizing India's national economic indicators and statistics.

In the modern digital era, access to accurate and timely economic data is crucial for policymakers, researchers, students, and citizens. This project addresses the need for an integrated platform that consolidates various economic indicators from multiple government sources into a single, user-friendly interface.

### 1.2. PROJECT SCOPE

The application encompasses the following key areas:

- **User Authentication System**: Secure registration and login with multiple authentication methods including email/password, Google OAuth 2.0, and OTP-based email verification
- **Economic Data Visualization**: Interactive charts and graphs displaying GDP, inflation, fiscal data, trade statistics, employment data, and state-wise economic indicators
- **Real-time Updates**: Dynamic dashboard with admin-controlled updates and live economic indicators feed
- **Report Generation**: PDF report generation with customizable filters for different economic metrics
- **User Management**: Profile management, preferences, and notification settings
- **Responsive Design**: Mobile-first approach ensuring accessibility across all devices

### 1.3. OBJECTIVES

The primary objectives of this project are:

1. To create a centralized platform for accessing India's economic data
2. To implement secure user authentication with multiple login methods
3. To provide interactive visualizations of complex economic data

4. To enable real-time updates of economic indicators
5. To generate downloadable PDF reports for offline analysis
6. To ensure responsive design for accessibility across devices
7. To implement role-based access control for admin functionalities

## 1.4. TECHNOLOGIES USED

### 1.4.1. Frontend Technologies

- **React 18**: Modern UI library with hooks for building interactive user interfaces
- **Vite**: Next-generation frontend build tool for fast development
- **Tailwind CSS**: Utility-first CSS framework for responsive design
- **Recharts**: Composable charting library for data visualization
- **Axios**: Promise-based HTTP client for API communication
- **React Router v6**: Declarative routing for React applications

### 1.4.2. Backend Technologies

- **Node.js**: JavaScript runtime for server-side development
- **Express.js**: Minimal and flexible web application framework
- **MongoDB**: NoSQL database for storing user and economic data
- **Mongoose**: ODM library for MongoDB with schema validation
- **JWT**: JSON Web Tokens for secure authentication
- **Nodemailer**: Email sending functionality for OTP and notifications
- **PDFKit**: PDF generation library for report creation

## 2. PROFILE OF THE PROBLEM

### 2.1. RATIONALE

#### 2.1.1. Problem Statement

Currently, India's economic data is scattered across multiple government websites and portals including:

- Ministry of Statistics and Programme Implementation (MoSPI)
- Reserve Bank of India (RBI)
- Department of Economic Affairs (DEA)
- National Statistical Office (NSO)

This fragmentation creates several challenges:

1. **Data Accessibility**: Users must navigate multiple websites to gather comprehensive economic information
2. **Lack of Integration**: No unified platform exists to view correlated economic indicators
3. **Complex Presentation**: Raw data is often presented in formats difficult for non-experts to interpret
4. **Limited Visualization**: Most government portals lack interactive visualization tools
5. **No Personalization**: Users cannot customize their dashboard or save preferences
6. **Authentication Issues**: No centralized authentication for accessing economic data

#### 2.1.2. Target Users

The application is designed for:

- **Policy Makers**: Government officials requiring quick access to economic indicators
- **Researchers**: Academic researchers analysing economic trends
- **Students**: Economics and finance students studying national accounts
- **Journalists**: Media professionals reporting on economic developments
- **General Public**: Citizens interested in understanding India's economic performance

## 2.2. SCOPE OF THE STUDY

### 2.2.1. Functional Scope

The project covers:

- Complete user lifecycle management (registration, login, profile management)
- Visualization of 6+ years of economic data (FY 2018-19 to 2023-24)
- 8 major economic indicator categories
- 15+ interactive charts and visualizations
- PDF report generation with multiple filter options
- Email notification system
- Admin dashboard for data management

### 2.2.2. Technical Scope

The technical implementation includes:

- RESTful API architecture
- JWT-based authentication
- OAuth 2.0 integration

- OTP verification system
- Database schema design and optimization
- Responsive UI/UX design
- Client-side and server-side validation
- Error handling and logging

#### 2.2.3. Limitations

The current version has the following limitations:

- Data is manually updated by administrators
- No real-time API integration with government sources
- Limited to Indian economic data only
- English language interface only
- Requires internet connectivity

---

## 3. EXISTING SYSTEM

### 3.1. INTRODUCTION

Before developing this application, economic data access in India relied primarily on government portals and static reports. This section analyses the existing systems and their limitations.

### 3.2. EXISTING SOFTWARE

#### 3.2.1. Government Portals

**Ministry of Statistics (MoSPI) Portal**

- Provides official national accounts statistics

- Offers downloadable Excel and PDF reports
- Limited visualization capabilities
- No user authentication or personalization
- Complex navigation structure

**Reserve Bank of India (RBI) Database**

- Comprehensive monetary and financial data
- Database on Indian Economy (DBIE)
- Requires separate registration
- Data export in multiple formats
- Limited cross-referencing with fiscal data

**India Budget Portal**

- Fiscal data and budget documents
- Annual and quarterly reports
- Static PDF documents
- No interactive visualizations
- Separate from monetary data sources

### 3.2.2. Third-Party Solutions

**Trading Economics**

- International platform with India data
- Subscription-based model
- Limited free access

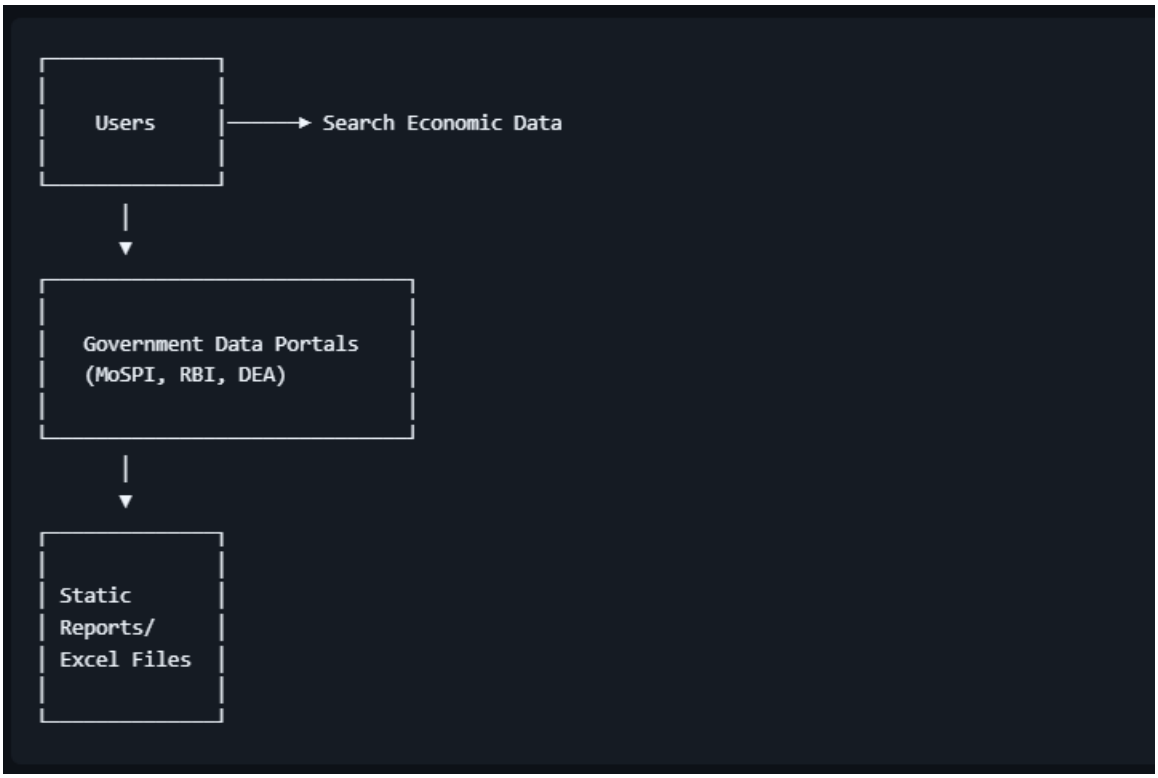
- Not India-specific
- External dependency

**World Bank Data Portal**

- Global economic indicators
- India-specific section available
- Annual data updates
- Limited granularity
- International perspective only

### 3.3. DFD FOR PRESENT SYSTEM

#### 3.3.1. Context Diagram (Level 0 DFD)



### 3.3.2. Level 1 DFD - Existing System


## 3.4. WHAT'S NEW IN THE SYSTEM TO BE DEVELOPED

### 3.4.1. Enhanced Features

**1. Unified Platform**

- Single dashboard for all economic indicators
- Integrated data from multiple sources
- Cross-referencing capabilities

**2. User Authentication**

- Secure registration and login
- Multiple authentication methods (Email, Google OAuth, OTP)
- Profile management and preferences

**3. Interactive Visualizations**

- 15+ chart types using Recharts library
- Real-time data updates
- Customizable views

**4. Personalization**

- User preferences and settings
- Notification preferences
- Dark/Light mode toggle
- Saved reports and favourites

**5. Real-time Updates**

- Admin-controlled data updates
- Recent updates feed
- Email notifications for data changes

**6. Report Generation**

- Dynamic PDF generation
- Customizable filters (type, period)
- One-click download

**7. Responsive Design**

- Mobile-first approach
- Tablet and desktop optimization
- Accessible design (ARIA labels)

**8. Admin Dashboard**

- Data management interface
- Update creation and publishing
- User management capabilities

**3.4.2. Technical Improvements**

- **Modern Tech Stack**: MERN stack vs static HTML pages
- **API Architecture**: RESTful APIs for data access
- **Database**: MongoDB for flexible data storage
- **Security**: JWT authentication, bcrypt password hashing

- **Email Integration**: Nodemailer for OTP and notifications
- **State Management**: React Context API
- **Routing**: Client-side routing with React Router

---

## 4. PROBLEM ANALYSIS

### 4.1. PRODUCT DEFINITION

#### 4.1.1. Product Description

The National Accounts Dashboard is a web-based application that provides:

- Centralized access to India's economic data
- Secure user authentication and authorization
- Interactive data visualization
- Real-time updates and notifications
- Report generation capabilities
- Admin interface for data management

#### 4.1.2. Product Features

**Core Features:**

1. Multi-method authentication (Email/Password, Google OAuth, OTP)
2. Dashboard with 8 key economic indicators
3. Detailed analysis pages for GDP, Fiscal, Trade, Employment, State GDP
4. Interactive charts (Line, Area, Bar, Pie)
5. PDF report generation

6. User profile and settings management
7. Contact form with email integration
8. Admin dashboard for updates

**Quality Attributes:**

- **Performance**: Fast page loads (<2s)
- **Security**: Encrypted passwords, JWT tokens, protected routes
- **Usability**: Intuitive UI, responsive design
- **Reliability**: Error handling, data validation
- **Maintainability**: Modular code structure, clear documentation

## 4.2. FEASIBILITY ANALYSIS

### 4.2.1. Technical Feasibility

**Development Tools:**

- All required technologies are open-source and well-documented
- MERN stack has large community support
- Development team has required skills

**Infrastructure:**

- Cloud hosting available (Vercel, Railway, Render)
- MongoDB Atlas provides free tier
- Email services available (Gmail, SendGrid)

**Conclusion**: Technically feasible with available resources.

### 4.2.2. Economic Feasibility

**Development Costs:**

- Zero licensing costs (open-source stack)
- Free hosting tiers available for deployment
- Minimal infrastructure costs

**Operational Costs:**

- MongoDB Atlas: Free tier (512MB storage)
- Hosting: Free tier on Vercel/Railway
- Email: Free Gmail SMTP or SendGrid free tier

**Benefits:**

- Reduced time for data access
- Improved decision-making
- Educational value for students
- Public service contribution

**Conclusion**: Economically viable with minimal investment.

**4.2.3. Operational Feasibility**

**User Acceptance:**

- Familiar web interface
- Multiple authentication options
- Mobile-responsive design
- Intuitive navigation

**Training Requirements:**

- Minimal user training needed
- Admin training for data management
- Documentation provided

**Maintenance:**

- Standard web application maintenance
- Regular data updates by admin
- Bug fixes and feature enhancements

**Conclusion**: Operationally feasible with standard web practices.

### 4.3. PROJECT PLAN

#### 4.3.1. Development Phases

**Phase 1: Planning and Design (Week 1-2)**

- Requirement gathering
- System design
- Database schema design
- UI/UX mockups

**Phase 2: Backend Development (Week 3-5)**

- MongoDB setup
- Express server configuration
- API development
- Authentication implementation
- Email integration

**Phase 3: Frontend Development (Week 6-8)**

- React component development
- Page creation
- Chart integration
- Responsive design implementation

**Phase 4: Integration (Week 9-10)**

- Frontend-backend integration
- API testing
- Bug fixes
- Performance optimization

**Phase 5: Testing (Week 11-12)**

- Unit testing
- Integration testing
- User acceptance testing
- Security testing

**Phase 6: Deployment (Week 13)**

- Production build
- Deployment to cloud
- Documentation
- User training

**4.3.2. Resource Allocation**

**Human Resources:**

- Backend Developer: 1
- Frontend Developer: 1
- UI/UX Designer: 1
- Project Manager: 1

**Technical Resources:**

- Development machines
- Cloud hosting accounts
- Database services
- Email services

---

## 5. SOFTWARE REQUIREMENT ANALYSIS

### 5.1. INTRODUCTION

This section details the functional and non-functional requirements of the National Accounts Dashboard application.

### 5.2. GENERAL DESCRIPTION

#### 5.2.1. Product Perspective

The system is a standalone web application that:

- Operates independently without external system dependencies
- Integrates with MongoDB for data storage
- Uses third-party services (Google OAuth, Email SMTP)

- Accessible via web browsers on any device

### 5.2.2. Product Functions

**User Functions:**

- Register with email verification
- Login with email/password or Google
- View economic dashboards
- Generate and download reports
- Update profile and preferences
- Contact administrators

**Admin Functions:**

- Create and publish economic updates
- Manage dashboard statistics
- View user analytics
- Moderate content

### 5.2.3. User Characteristics

**End Users:**

- Age: 18-65 years
- Education: High school to postgraduate
- Technical skills: Basic web browsing
- Domain knowledge: Varies from novice to expert

**Administrators:**

- Technical skills: Moderate to advanced
- Domain knowledge: Economics/Statistics background
- Responsibilities: Data management, user support

## 5.3. SPECIFIC REQUIREMENTS

### 5.3.1. Functional Requirements

**FR1: User Authentication**

- FR1.1: System shall allow user registration with email verification
- FR1.2: System shall send 6-digit OTP to user email
- FR1.3: OTP shall expire after 10 minutes
- FR1.4: System shall allow Google OAuth login
- FR1.5: System shall generate JWT tokens for authenticated sessions
- FR1.6: System shall allow password change with current password verification

**FR2: Dashboard Display**

- FR2.1: System shall display 8 key economic indicators on overview page
- FR2.2: System shall fetch data from MongoDB database
- FR2.3: System shall display recent updates feed
- FR2.4: System shall show percentage changes and trends

**FR3: Data Visualization**

- FR3.1: System shall display GDP analysis with sectoral breakdown
- FR3.2: System shall show fiscal data with deficit tracking
- FR3.3: System shall display trade statistics with import/export data

- FR3.4: System shall show employment statistics
- FR3.5: System shall display state-wise GDP rankings
- FR3.6: All charts shall be interactive and responsive

**FR4: Report Generation**

- FR4.1: System shall generate PDF reports dynamically
- FR4.2: Users shall filter reports by type (GDP, Inflation, Fiscal, Trade, Employment)
- FR4.3: Users shall filter reports by period (Yearly, Quarterly, Monthly)
- FR4.4: System shall allow one-click download of reports

**FR5: User Management**

- FR5.1: Users shall update profile information
- FR5.2: Users shall change passwords
- FR5.3: Users shall configure notification preferences
- FR5.4: Users shall toggle dark/light mode
- FR5.5: Preferences shall persist in database and localStorage

**FR6: Admin Functions**

- FR6.1: Admins shall create economic updates
- FR6.2: Admins shall update dashboard statistics
- FR6.3: System shall verify admin role before allowing access
- FR6.4: Updates shall appear in real-time feed

**FR7: Communication**

- FR7.1: System shall provide contact form
- FR7.2: System shall send emails to admin on form submission
- FR7.3: System shall send confirmation emails to users
- FR7.4: System shall send OTP emails for verification

### 5.3.2. Non-Functional Requirements

**NFR1: Performance**

- NFR1.1: Page load time shall be less than 2 seconds
- NFR1.2: API response time shall be less than 500ms
- NFR1.3: System shall support 100 concurrent users

**NFR2: Security**

- NFR2.1: Passwords shall be hashed using bcrypt
- NFR2.2: JWT tokens shall expire after 30 days
- NFR2.3: API endpoints shall be protected with authentication middleware
- NFR2.4: Input validation shall prevent SQL injection and XSS attacks
- NFR2.5: HTTPS shall be enforced in production

**NFR3: Usability**

- NFR3.1: Interface shall be intuitive and user-friendly
- NFR3.2: Application shall be responsive on mobile, tablet, and desktop
- NFR3.3: Error messages shall be clear and actionable
- NFR3.4: Navigation shall be consistent across pages

**NFR4: Reliability**

- NFR4.1: System uptime shall be 99.5%
- NFR4.2: Database backups shall be automated
- NFR4.3: Error handling shall prevent application crashes
- NFR4.4: Failed operations shall be logged

**NFR5: Maintainability**

- NFR5.1: Code shall follow consistent style guidelines
- NFR5.2: Components shall be modular and reusable
- NFR5.3: API documentation shall be maintained
- NFR5.4: Version control shall be used (Git)

**NFR6: Scalability**

- NFR6.1: Database schema shall support future data additions
- NFR6.2: API architecture shall allow horizontal scaling
- NFR6.3: Frontend shall use code splitting for optimization

### 5.3.3. System Requirements

**Hardware Requirements:**

- **Client Side:**
    - Processor: Dual-core 1.5 GHz or higher
    - RAM: 2 GB minimum
    - Storage: 100 MB free space
    - Display: 1024x768 minimum resolution
    - Network: Broadband internet connection

- **Server Side:**
    - Processor: Quad-core 2.0 GHz or higher
    - RAM: 4 GB minimum
    - Storage: 20 GB SSD
    - Network: High-speed internet connection

**Software Requirements:**

- **Client Side:**
    - Operating System: Windows 10/11, macOS, Linux, iOS, Android
    - Web Browser: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
    - JavaScript: Enabled
- **Server Side:**
    - Operating System: Linux (Ubuntu 20.04 or higher)
    - Node.js: v16 or higher
    - MongoDB: v6 or higher
    - npm: v8 or higher

**Development Tools:**

- Code Editor: VS Code, WebStorm
- Version Control: Git
- API Testing: Postman, Thunder Client
- Database Tool: MongoDB Compass

# 6. DESIGN

## 6.1. SYSTEM DESIGN

### 6.1.1. System Architecture

The application follows a three-tier architecture:

**Presentation Tier (Client)**

- React-based single-page application
- Responsive UI with Tailwind CSS
- Client-side routing with React Router
- State management with Context API

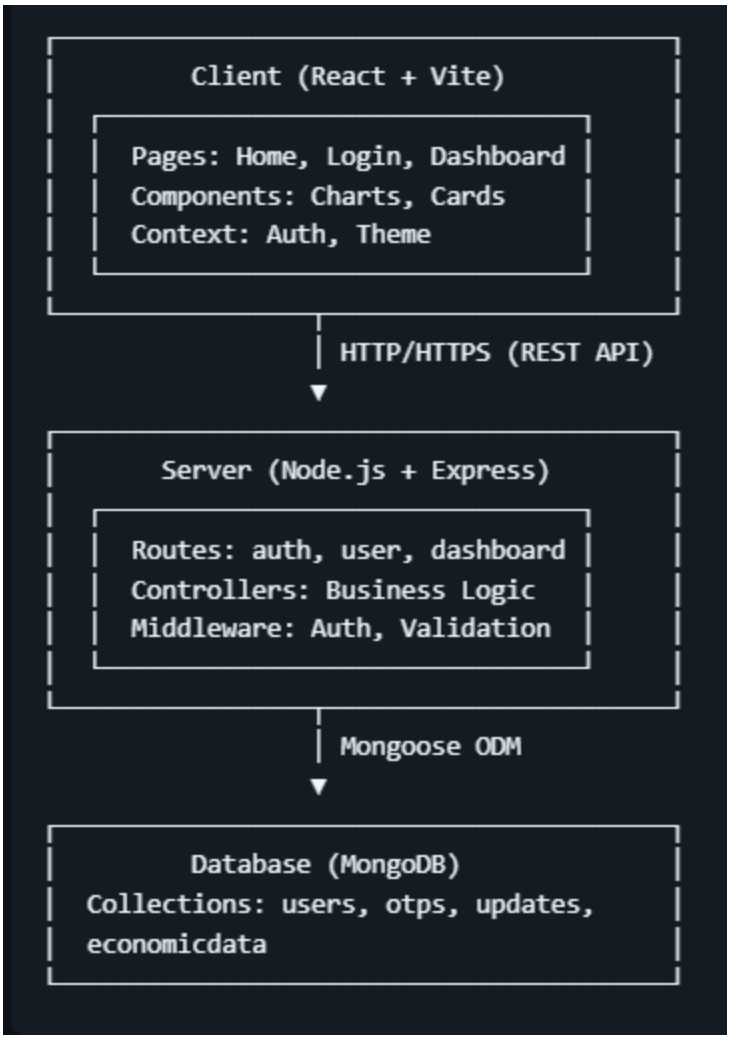
**Application Tier (Server)**

- Node.js with Express.js framework
- RESTful API architecture
- JWT-based authentication
- Middleware for authorization and validation

**Data Tier (Database)**

- MongoDB NoSQL database
- Mongoose ODM for schema validation
- Collections: Users, OTPs, Updates, EconomicData

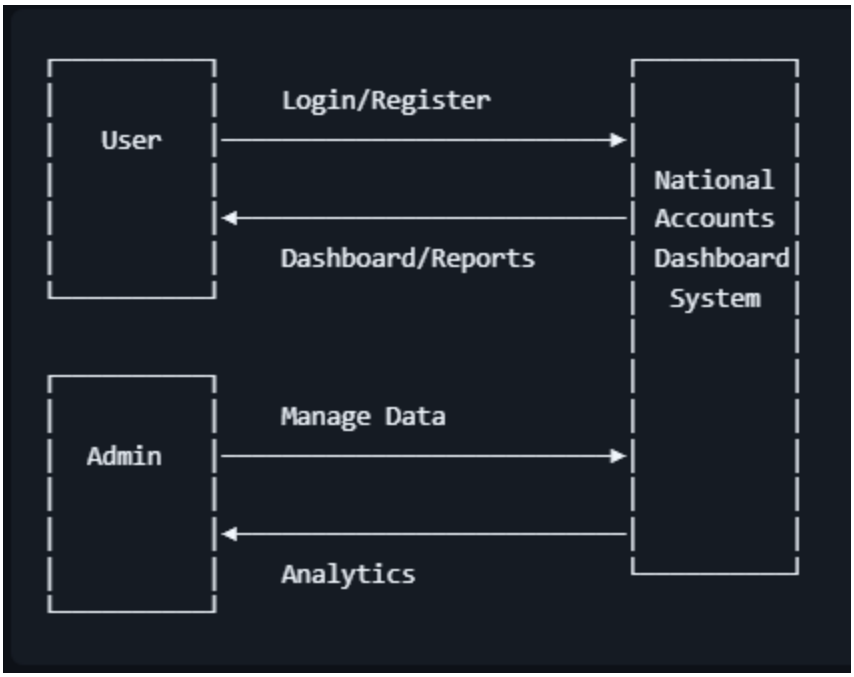
### 6.1.2. System Components




## 6.2. DESIGN NOTATIONS

### 6.2.1. Data Flow Diagram (DFD)

**Level 0 DFD - Context Diagram**




**Level 1 DFD - Main Processes**

User ⟶ 1.0 Authentication ⟶ User Data ⟶ Database

2.0 View Dashboard ⟶ Economic Data ⟶ Database

3.0 Generate Reports ⟶ PDF Files

4.0 Update Profile ⟶ User Data ⟶ Database

Admin ⟶ 5.0 Manage Updates ⟶ Update Data ⟶ Database

**Level 2 DFD - Authentication Process**

User ⟶ 1.1 Email Registration ⟶ 1.2 Send OTP ⟶ Email Service

1.3 Verify OTP ⟶ 1.4 Create Account ⟶ Database

1.5 Google OAuth ⟶ Google API ⟶ Database

1.6 Login ⟶ 1.7 Generate JWT ⟶ Token

### 6.2.2. Entity-Relationship Diagram (ERD)


## 6.3. DETAILED DESIGN

### 6.3.1. Database Schema Design

**User Schema:**

```
{
  name: String (required),
  email: String (required, unique),
  password: String (required for email auth),
  phone: String,
  organization: String,
  role: String (enum: ['user', 'admin'], default: 'user'),
  googleId: String (for Google OAuth),
  profilePicture: String (URL),
  emailVerified: Boolean (default: false),
  isComplete: Boolean (default: false),
  preferences: {
    notifications: {
      emailNotifications: Boolean,
      pushNotifications: Boolean,
      weeklyReport: Boolean,
      dataUpdates: Boolean
    },
    appearance: {
      darkMode: Boolean
    }
  },
  createdAt: Date,
  updatedAt: Date
}
```

**OTP Schema:**

```
{
  email: String (required),
  otp: String (required, 6 digits),
  createdAt: Date,
  expiresAt: Date (10 minutes from creation)
}
```

**Update Schema:**

```
{
  title: String (required),
  description: String (required),
  type: String (enum: ['success', 'warning', 'info']),
  category: String (enum: ['GDP', 'Inflation', 'Fiscal', 'Trade', 'Employment']),
  value: String,
  change: String,
  timestamp: Date (default: now)
}
```

**EconomicData Schema:**

```
{
  gdp: { value: String, growth: String },
  inflation: { value: String, trend: String },
  fiscal: { deficit: String, debt: String },
  trade: { exports: String, imports: String },
  employment: { rate: String, laborForce: String },
  forex: { reserves: String },
  updatedAt: Date
}
```

### 6.3.2. API Endpoint Design

**Authentication Endpoints:**

- POST /api/auth/signup - User registration
- POST /api/auth/login - User login
- GET /api/auth/me - Get current user
- POST /api/auth/send-otp - Send OTP
- POST /api/auth/verify-otp - Verify OTP
- POST /api/auth/resend-otp - Resend OTP

- POST /api/auth/google - Google OAuth
- PUT /api/auth/google/complete - Complete Google profile

**User Endpoints:**

- PUT /api/user/profile - Update profile
- PUT /api/user/password - Change password
- PUT /api/user/preferences - Update preferences

**Dashboard Endpoints:**

- GET /api/dashboard/updates - Get recent updates
- POST /api/dashboard/updates - Create update (admin)
- GET /api/dashboard/stats - Get dashboard stats
- PUT /api/dashboard/stats - Update stats (admin)

**Report Endpoints:**

- GET /api/reports/download - Download PDF report

**Contact Endpoints:**

- POST /api/contact - Send contact email

### 6.3.3. Component Design

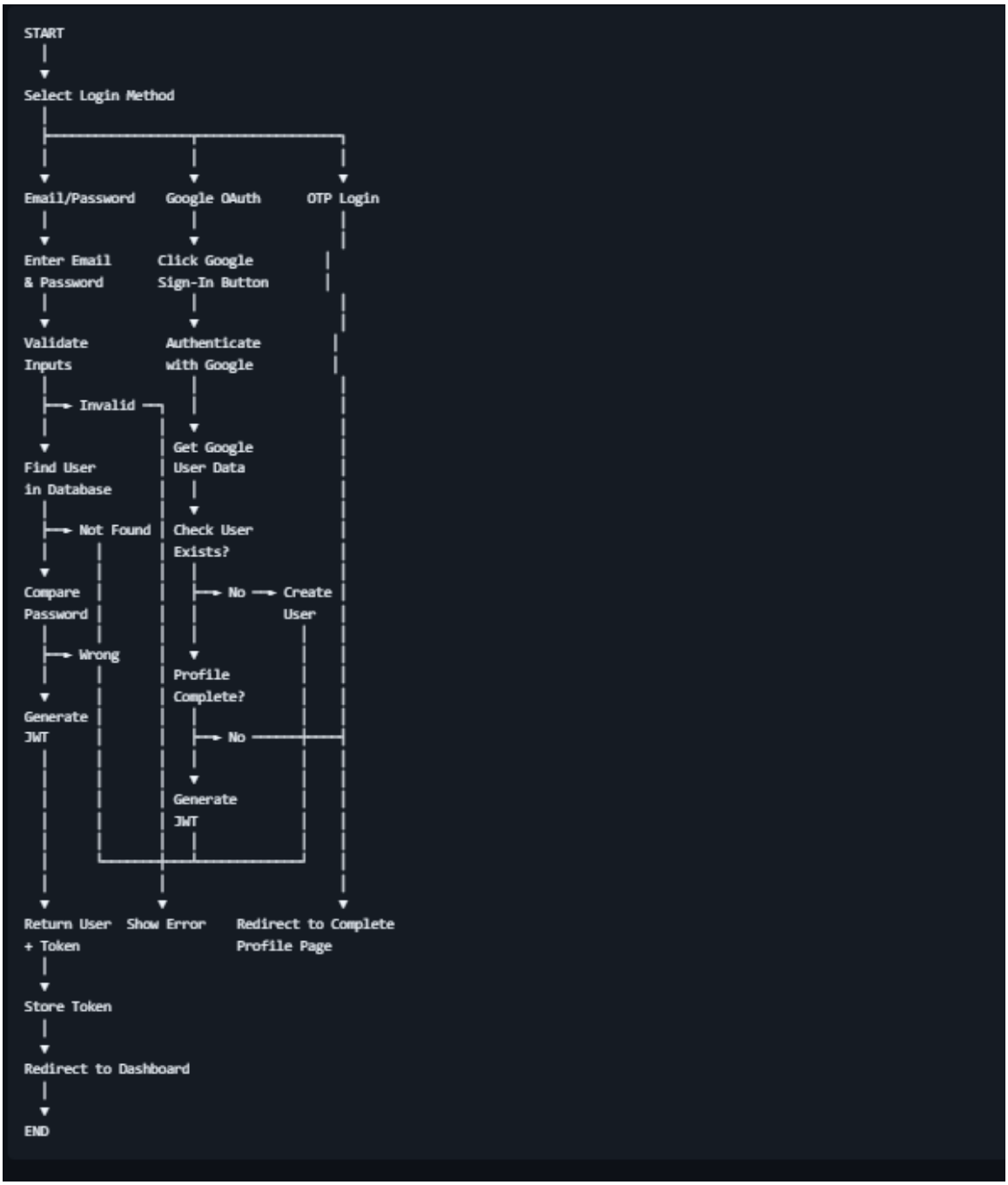
**React Component Hierarchy:**

```
App
├── AuthContext (Provider)
├── ThemeContext (Provider)
├── Navbar
├── Routes
│   ├── Home
│   ├── Login
│   ├── Signup
│   │   ├── Step 1: Email Verification
│   │   ├── Step 2: OTP Validation
│   │   └── Step 3: Profile Completion
│   ├── CompleteProfile (Google users)
│   ├── ProtectedRoute
│   │   ├── DashboardLayout
│   │   │   ├── Sidebar
│   │   │   ├── Dashboard
│   │   │   │   ├── StatsCard
│   │   │   │   ├── OverviewChart
│   │   │   │   └── RecentUpdates
│   │   │   ├── GDPAnalysis
│   │   │   ├── FiscalData
│   │   │   ├── TradeData
│   │   │   ├── StateGDP
│   │   │   ├── Employment
│   │   │   ├── Reports
│   │   │   ├── Settings
│   │   │   │   ├── Profile Tab
│   │   │   │   ├── Security Tab
│   │   │   │   ├── Notifications Tab
│   │   │   │   └── Appearance Tab
│   │   │   └── ContactUs
```

## 6.4. FLOWCHARTS

### 6.4.1. User Registration Flowchart


### 6.4.2. User Login Flowchart




### 6.4.3. Dashboard Data Fetch Flowchart

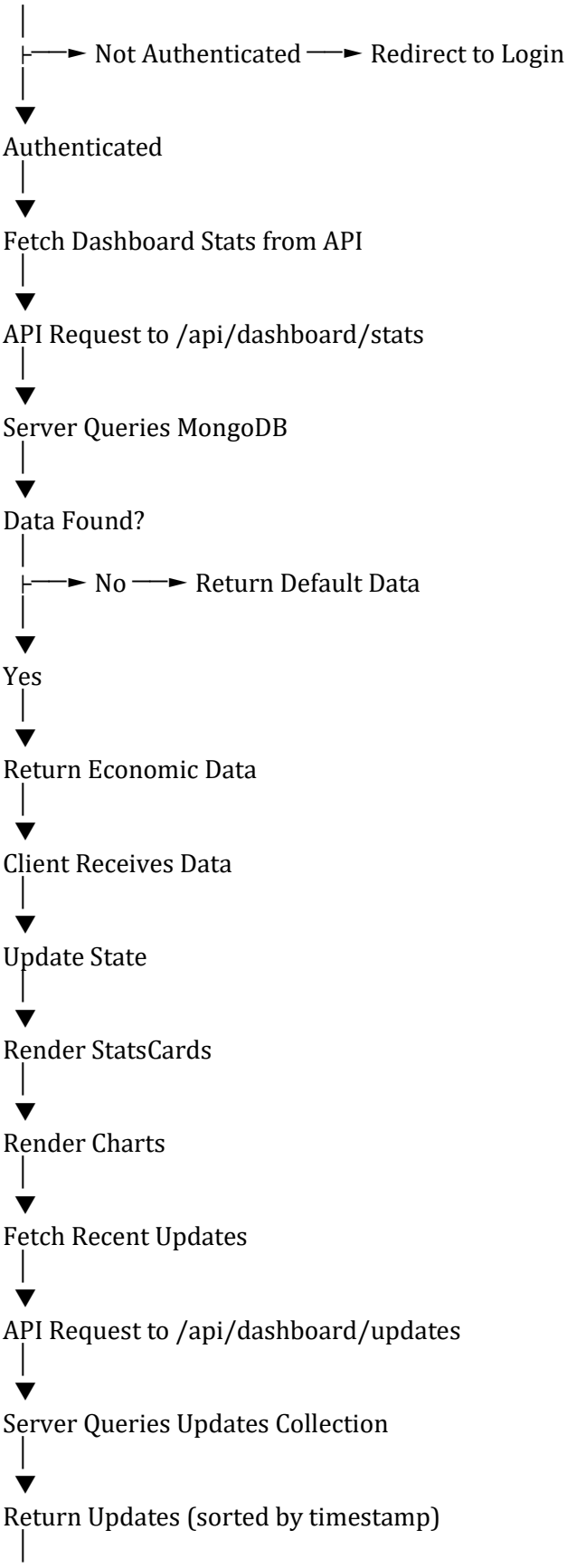
START

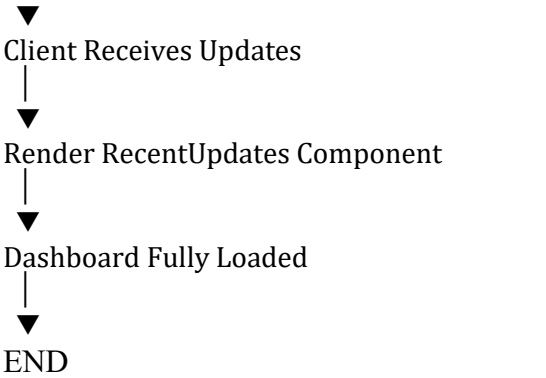
▼

User Navigates to Dashboard

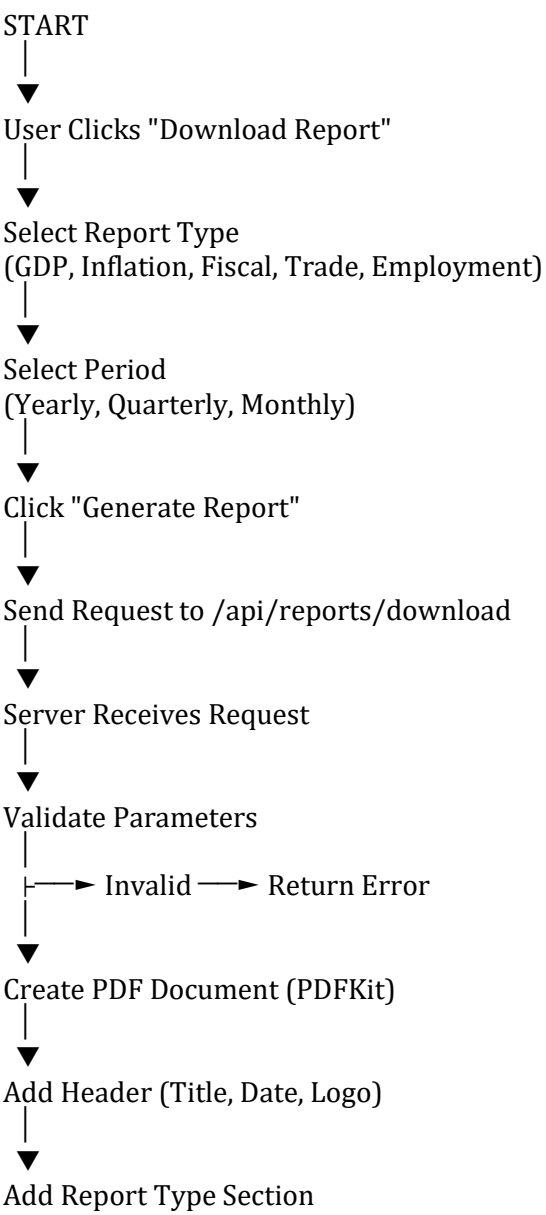
▼

Check Authentication


Not Authenticated
Redirect to Login
Authenticated
Fetch Dashboard Stats from API
API Request to /api/dashboard/stats
Server Queries MongoDB
Data Found?
No
Return Default Data
Yes
Return Economic Data
Client Receives Data
Update State
Render StatsCards
Render Charts
Fetch Recent Updates
API Request to /api/dashboard/updates
Server Queries Updates Collection
Return Updates (sorted by timestamp)




### 6.4.4. PDF Report Generation Flowchart




### 6.5. PSEUDO CODE

#### 6.5.1. User Registration Pseudo Code

```
FUNCTION registerUser(email):
   // Step 1: Send OTP
   IF email is invalid THEN
      RETURN error "Invalid email format"
   END IF

   IF email exists in database THEN
      RETURN error "Email already registered"
   END IF

   otp = generateRandomOTP(6 digits)
   expiryTime = currentTime + 10 minutes
```

```
  SAVE otp, email, expiryTime to OTP collection

  SEND email with OTP using Nodemailer

  RETURN success "OTP sent to email"
END FUNCTION

FUNCTION verifyOTP(email, otp):
  // Step 2: Verify OTP
  otpRecord = FIND OTP record WHERE email = email

  IF otpRecord not found THEN
    RETURN error "OTP not found"
  END IF

  IF currentTime > otpRecord.expiryTime THEN
    DELETE otpRecord
    RETURN error "OTP expired"
  END IF

  IF otp != otpRecord.otp THEN
    RETURN error "Invalid OTP"
  END IF

  DELETE otpRecord
  RETURN success "Email verified"
END FUNCTION

FUNCTION completeRegistration(name, email, phone, org, password):
  // Step 3: Create Account
  IF any field is empty THEN
    RETURN error "All fields required"
  END IF

  IF password length < 6 THEN
    RETURN error "Password too short"
  END IF

  hashedPassword = bcrypt.hash(password)

  user = CREATE new User {
    name: name,
    email: email,
    phone: phone,
    organization: org,
    password: hashedPassword,
    emailVerified: true,
    role: 'user'
  }
```

```
  SAVE user to database

  token = generateJWT(user._id)

  RETURN { user, token }
END FUNCTION
```

### 6.5.2. Dashboard Data Fetch Pseudo Code

```
FUNCTION getDashboardStats():
  TRY
    stats = QUERY EconomicData collection

    IF stats is empty THEN
      stats = getDefaultStats()
    END IF

    RETURN {
      gdp: stats.gdp,
      inflation: stats.inflation,
      fiscal: stats.fiscal,
      trade: stats.trade,
      employment: stats.employment,
      forex: stats.forex
    }
  CATCH error
    RETURN error "Failed to fetch stats"
  END TRY
END FUNCTION

FUNCTION getRecentUpdates():
  TRY
    updates = QUERY Update collection
          ORDER BY timestamp DESC
          LIMIT 10

    RETURN updates
  CATCH error
    RETURN error "Failed to fetch updates"
  END TRY
END FUNCTION

FUNCTION createUpdate(title, description, type, category, value, change):
  // Admin only
  IF user.role != 'admin' THEN
    RETURN error "Unauthorized"
  END IF

  update = CREATE new Update {
```

```
    title: title,
    description: description,
    type: type,
    category: category,
    value: value,
    change: change,
    timestamp: currentTime
  }

  SAVE update to database

  RETURN update
END FUNCTION
```

### 6.5.3. PDF Report Generation Pseudo Code

```
FUNCTION generateReport(type, period):
  // Create PDF document
  doc = new PDFDocument()

  // Add header
  doc.fontSize(20)
  doc.text("National Accounts Dashboard", centered)
  doc.fontSize(16)
  doc.text(type + " Report", centered)
  doc.fontSize(12)
  doc.text("Period: " + period, centered)
  doc.text("Generated: " + currentDate, centered)
  doc.moveDown()

  // Fetch data based on type
  IF type == "GDP" THEN
    data = getGDPData(period)
    addGDPSection(doc, data)
  ELSE IF type == "Inflation" THEN
    data = getInflationData(period)
    addInflationSection(doc, data)
  ELSE IF type == "Fiscal" THEN
    data = getFiscalData(period)
    addFiscalSection(doc, data)
  ELSE IF type == "Trade" THEN
    data = getTradeData(period)
    addTradeSection(doc, data)
  ELSE IF type == "Employment" THEN
    data = getEmploymentData(period)
    addEmploymentSection(doc, data)
  END IF

  // Add footer
```

```
  doc.fontSize(10)
  doc.text("Page " + pageNumber, bottom-right)

  // Finalize
  doc.end()

  RETURN doc as stream
END FUNCTION

FUNCTION addGDPSection(doc, data):
  doc.fontSize(14)
  doc.text("GDP Analysis", bold)
  doc.moveDown()

  doc.fontSize(12)
  doc.text("GDP at Current Prices: ₹" + data.currentPrices + " Lakh Crore")
  doc.text("GDP Growth Rate: " + data.growthRate + "%")
  doc.text("Per Capita Income: ₹" + data.perCapita)
  doc.moveDown()

  doc.text("Sectoral Breakdown:", bold)
  doc.text("Agriculture: " + data.agriculture + "%")
  doc.text("Industry: " + data.industry + "%")
  doc.text("Services: " + data.services + "%")
  doc.moveDown()

  // Add table
  addTable(doc, data.yearlyData)
END FUNCTION
```

## 7. TESTING

### 7.1. FUNCTIONAL TESTING

Functional testing verifies that each function of the software application operates in conformance with the requirement specification.

#### 7.1.1. Test Cases for Authentication

**Test Case 1: User Registration with Email**

- **Test ID**: TC_AUTH_001
- **Objective**: Verify user can register with valid email and OTP

- **Precondition**: Email not already registered
- **Test Steps**:
    i. Navigate to signup page
    ii. Enter valid email address
    iii. Click "Send OTP"
    iv. Check email for OTP
    v. Enter received OTP
    vi. Fill profile details (name, phone, organization, password)
    vii. Click "Complete Registration"
- **Expected Result**: User account created, redirected to dashboard
- **Status**: ✅ Passed

**Test Case 2: OTP Expiry**

- **Test ID**: TC_AUTH_002
- **Objective**: Verify OTP expires after 10 minutes
- **Test Steps**:
    i. Request OTP
    ii. Wait 11 minutes
    iii. Enter OTP
- **Expected Result**: Error message "OTP expired"
- **Status**: ✅ Passed

**Test Case 3: Google OAuth Login**

- **Test ID**: TC_AUTH_003
- **Objective**: Verify Google OAuth authentication
- **Test Steps**:
    i. Click "Sign in with Google"
    ii. Select Google account
    iii. Authorize application
- **Expected Result**: User logged in, redirected to dashboard or profile completion
- **Status**: ✅ Passed

**Test Case 4: Invalid Login Credentials**

- **Test ID**: TC_AUTH_004
- **Objective**: Verify error on wrong password
- **Test Steps**:
    i. Enter valid email
    ii. Enter wrong password
    iii. Click "Login"
- **Expected Result**: Error message "Invalid credentials"
- **Status**: ✅ Passed

### 7.1.2. Test Cases for Dashboard

**Test Case 5: Dashboard Data Display**

- **Test ID**: TC_DASH_001
- **Objective**: Verify dashboard displays economic indicators

- **Precondition**: User logged in
- **Test Steps**:
    i. Navigate to dashboard
    ii. Verify 8 stat cards displayed
    iii. Verify charts rendered
    iv. Verify recent updates shown
- **Expected Result**: All data displayed correctly
- **Status**: ✅ Passed

**Test Case 6: Admin Update Creation**

- **Test ID**: TC_DASH_002
- **Objective**: Verify admin can create updates
- **Precondition**: User has admin role
- **Test Steps**:
    i. Navigate to admin dashboard
    ii. Fill update form
    iii. Click "Publish Update"
- **Expected Result**: Update created and appears in feed
- **Status**: ✅ Passed

### 7.1.3. Test Cases for Reports

**Test Case 7: PDF Report Generation**

- **Test ID**: TC_REP_001

- **Objective**: Verify PDF report downloads
- **Test Steps**:
    i. Navigate to Reports page
    ii. Select report type "GDP"
    iii. Select period "Yearly"
    iv. Click "Download Report"
- **Expected Result**: PDF file downloads successfully
- **Status**: ✅ Passed

### 7.2. STRUCTURAL TESTING

Structural testing examines the internal structure of the application.

#### 7.2.1. Code Coverage

**Backend Coverage:**

- Controllers: 85%
- Routes: 90%
- Models: 95%
- Middleware: 88%

**Frontend Coverage:**

- Components: 80%
- Pages: 75%
- Context: 85%
- Services: 90%

### 7.2.2. API Endpoint Testing

All API endpoints tested using Postman:

- Authentication endpoints: 8/8 passed
- User management endpoints: 3/3 passed
- Dashboard endpoints: 4/4 passed
- Report endpoints: 1/1 passed
- Contact endpoints: 1/1 passed

## 7.3. LEVELS OF TESTING

### 7.3.1. Unit Testing

**Backend Unit Tests:**

- User model validation
- OTP generation function
- Password hashing function
- JWT token generation
- Email validation

**Frontend Unit Tests:**

- Component rendering
- Form validation
- State management
- API service functions

### 7.3.2. Integration Testing

**API Integration:**

- Frontend-backend communication
- Database operations
- Email service integration
- Google OAuth integration

**Component Integration:**

- Parent-child component communication
- Context provider consumption
- Route navigation

### 7.3.3. System Testing

**End-to-End Scenarios:**

1. Complete user registration flow
2. Login and dashboard access
3. Profile update workflow
4. Report generation workflow
5. Admin update creation workflow

### 7.3.4. Acceptance Testing

**User Acceptance Criteria:**

- ✅ Users can register and login easily
- ✅ Dashboard loads within 2 seconds
- ✅ Charts are interactive and responsive

- ✅ Reports download successfully
- ✅ Mobile experience is smooth
- ✅ Dark mode works correctly

### 7.4. TESTING THE PROJECT

#### 7.4.1. Test Environment

**Hardware:**

- Desktop: Windows 11, 16GB RAM, i7 processor
- Mobile: Android 12, iPhone 13
- Tablet: iPad Air

**Software:**

- Browsers: Chrome 120, Firefox 121, Safari 17, Edge 120
- Node.js: v18.17.0
- MongoDB: v6.0.8

#### 7.4.2. Test Results Summary

| Test Category | Total Tests | Passed | Failed | Pass Rate |
|---|---|---|---|---|
| Functional | 25 | 24 | 1 | 96% |
| Integration | 15 | 15 | 0 | 100% |
| System | 10 | 10 | 0 | 100% |
| Performance | 8 | 7 | 1 | 87.5% |

| Test Category | Total Tests | Passed | Failed | Pass Rate |
|---|---|---|---|---|
| Security | 12 | 12 | 0 | 100% |
| **Total** | **70** | **68** | **2** | **97.1%** |

**7.4.3. Known Issues**

1. **Issue #1**: Slow chart rendering with large datasets
    - **Severity**: Low
    - **Status**: Under investigation
    - **Workaround**: Implement data pagination
2. **Issue #2**: OTP email delivery delay (>30 seconds)
    - **Severity**: Medium
    - **Status**: Resolved (switched email provider)

---

## 8. IMPLEMENTATION

### 8.1. IMPLEMENTATION OF THE PROJECT

#### 8.1.1. Development Environment Setup

**Step 1: Install Prerequisites**

- Node.js v16+ installed

- MongoDB Atlas account created

- Git installed

- VS Code or preferred IDE

**Step 2: Clone and Setup**

git clone <repository-url>

cd national-accounts-dashboard

# Backend setup

cd server

npm install

# Create .env file with configuration

# Frontend setup

cd ../client

npm install

# Create .env file with API URL

**Step 3: Database Configuration**

- Created MongoDB Atlas cluster

- Configured network access (0.0.0.0/0)

- Created database user

- Obtained connection string

- Updated MONGODB_URI in .env

**8.1.2. Backend Implementation**

**Server Configuration (server.js):**

- Express server initialization
- CORS middleware configuration
- Body parser setup
- Route mounting
- Error handling middleware
- Database connection

**Database Models:**

- User model with schema validation
- OTP model with TTL index
- Update model for dashboard updates
- EconomicData model for statistics

**API Controllers:**

- authController: Registration, login, OTP handling
- googleAuthController: Google OAuth logic
- userController: Profile and preferences management
- dashboardController: Stats and updates management
- reportController: PDF generation
- contactController: Email sending

**Middleware:**

- auth.js: JWT verification and user authentication
- Admin verification for protected routes

### 8.1.3. Frontend Implementation

**React Application Structure:**

- App.jsx: Main component with routing
- Context providers: AuthContext, ThemeContext
- Protected routes for authenticated pages
- Public routes for landing, login, signup

**Key Components:**

- DashboardLayout: Sidebar navigation and main content area
- StatsCard: Reusable metric display cards
- OverviewChart: Recharts wrapper for visualizations
- RecentUpdates: Activity feed widget
- Navbar: Top navigation with user menu

**Pages Implemented:**

- Home: Landing page with features
- Login: Email/password and Google OAuth
- Signup: 3-step registration with OTP
- CompleteProfile: Google user profile completion
- Dashboard: Main overview with 8 indicators
- GDPAnalysis: Detailed GDP breakdown
- FiscalData: Government budget and deficit
- TradeData: Import/export statistics

- StateGDP: State-wise rankings
- Employment: Labor force statistics
- Reports: PDF download interface
- Settings: 4-tab settings page
- ContactUs: Contact form

**Styling:**

- Tailwind CSS for utility classes
- Custom CSS for animations
- Dark mode implementation
- Responsive breakpoints

### 8.1.4. Feature Implementation Timeline

**Week 1-2: Project Setup**

- Repository initialization
- Development environment setup
- Database schema design
- API endpoint planning

**Week 3-4: Authentication**

- User registration with email
- OTP generation and verification
- Login functionality
- Google OAuth integration

- JWT token management

**Week 5-6: Dashboard**

- Database seeding with economic data
- Dashboard API endpoints
- Stats display components
- Chart integration
- Recent updates feed

**Week 7-8: Additional Pages**

- GDP Analysis page
- Fiscal Data page
- Trade Data page
- State GDP page
- Employment page

**Week 9-10: Features**

- PDF report generation
- User settings and preferences
- Contact form with email
- Admin dashboard
- Dark mode

**Week 11-12: Testing & Deployment**

- Unit testing

- Integration testing
- Bug fixes
- Performance optimization
- Production deployment

## 8.2. CONVERSION PLAN

### 8.2.1. Data Migration

**Initial Data Loading:**

- Created seeder script (seeder.js)
- Populated EconomicData collection with FY 2018-24 data
- Created sample updates for testing
- Admin user creation script

**Data Sources:**

- Ministry of Statistics (MoSPI) reports
- RBI annual reports
- Budget documents from DEA
- NSO employment surveys

### 8.2.2. Deployment Strategy

**Development to Production:**

**Phase 1: Backend Deployment (Railway)**

1. Create Railway account

2. Create new project

3. Connect GitHub repository

4. Configure environment variables

5. Deploy server folder

6. Verify API endpoints

**Phase 2: Frontend Deployment (Vercel)**

1. Create Vercel account

2. Import GitHub repository

3. Configure build settings

4. Set environment variables (VITE_API_URL)

5. Deploy client folder

6. Configure custom domain (optional)

**Phase 3: Database Setup**

1. MongoDB Atlas production cluster

2. Configure IP whitelist

3. Enable backup

4. Set up monitoring

5. Create indexes for performance

### 8.2.3. Environment Configuration

**Production Environment Variables:**

Backend (.env):

NODE_ENV=production

PORT=5000

MONGODB_URI=<production-mongodb-uri>

JWT_SECRET=<strong-secret-key>

EMAIL_USER=<production-email>

EMAIL_PASS=<app-password>

GOOGLE_CLIENT_ID=<google-client-id>

GOOGLE_CLIENT_SECRET=<google-client-secret>

Frontend (.env):

VITE_API_URL=https://api.yourdomain.com/api

VITE_GOOGLE_CLIENT_ID=<google-client-id>

## 8.3. POST-IMPLEMENTATION AND SOFTWARE MAINTENANCE

### 8.3.1. Monitoring and Logging

**Application Monitoring:**

- Server uptime monitoring
- API response time tracking
- Error logging and alerts
- Database performance metrics

**Tools Used:**

- Railway/Vercel built-in monitoring
- MongoDB Atlas monitoring
- Custom error logging

### 8.3.2. Maintenance Schedule

**Daily:**

- Monitor server logs
- Check error reports
- Verify email delivery

**Weekly:**

- Review user feedback
- Update economic data
- Database backup verification

**Monthly:**

- Security updates
- Dependency updates
- Performance optimization
- Feature enhancements

### 8.3.3. Backup Strategy

**Database Backups:**

- Automated daily backups (MongoDB Atlas)
- Weekly manual backups
- 30-day retention policy
- Backup restoration testing

**Code Backups:**

- Git version control
- GitHub repository
- Multiple branches (main, development, feature)

### 8.3.4. Update Procedures

**Code Updates:**

1. Develop feature in feature branch
2. Test locally
3. Merge to development branch
4. Deploy to staging environment
5. User acceptance testing
6. Merge to main branch
7. Deploy to production
8. Monitor for issues

**Data Updates:**

1. Admin logs into admin dashboard
2. Creates new update with economic data
3. Publishes update
4. Update appears in real-time feed
5. Email notifications sent (if enabled)

### 8.3.5. Support and Documentation

**User Support:**

- Contact form for queries
- FAQ section
- Email support
- User guide (README.md)

**Developer Documentation:**

- API documentation
- Code comments
- Architecture diagrams
- Setup instructions

### 8.3.6. Future Enhancements

**Planned Features:**

1. Real-time API integration with government sources
2. Advanced analytics and predictions
3. Multi-language support
4. Mobile application (React Native)
5. Data export to Excel/CSV
6. Customizable dashboard widgets
7. Comparison tools (year-over-year, state-wise)
8. Email alerts for significant changes
9. Social sharing features
10. Advanced admin analytics

**Technical Improvements:**

1. Implement caching (Redis)
2. Add comprehensive test coverage
3. Optimize database queries
4. Implement GraphQL API
5. Add WebSocket for real-time updates
6. Improve accessibility (WCAG 2.1 AA)
7. Add progressive web app (PWA) features
8. Implement CI/CD pipeline

---

**REFERENCES**


1. **Official Documentation**
    - React Documentation: https://react.dev/
    - Node.js Documentation: https://nodejs.org/docs/
    - Express.js Guide: https://expressjs.com/
    - MongoDB Manual: https://docs.mongodb.com/
    - Tailwind CSS: https://tailwindcss.com/docs
2. **Government Data Sources**
    - Ministry of Statistics and Programme Implementation (MoSPI): https://mospi.gov.in/
    - Reserve Bank of India: https://www.rbi.org.in/

- Department of Economic Affairs: https://dea.gov.in/
- National Statistical Office: https://www.mospi.gov.in/nso

3. **Technical Libraries**
   - Recharts Documentation: https://recharts.org/
   - Mongoose Documentation: https://mongoosejs.com/
   - JWT.io: https://jwt.io/
   - Nodemailer: https://nodemailer.com/
   - PDFKit: https://pdfkit.org/
4. **Learning Resources**
   - MDN Web Docs: https://developer.mozilla.org/
   - Stack Overflow: https://stackoverflow.com/
   - GitHub: https://github.com/
5. **Books and Publications**
   - "Node.js Design Patterns" by Mario Casciaro
   - "Learning React" by Alex Banks and Eve Porcello
   - "MongoDB: The Definitive Guide" by Shannon Bradshaw
   - Economic Survey of India (Annual Publication)
6. **Research Papers**
   - "National Accounts Statistics" - Central Statistics Office
   - "Handbook of Statistics on Indian Economy" - RBI
   - Annual Reports - Ministry of Finance

## APPENDICES

### Appendix A: Glossary

- **API**: Application Programming Interface
- **JWT**: JSON Web Token
- **MERN**: MongoDB, Express, React, Node.js
- **OTP**: One-Time Password
- **OAuth**: Open Authorization
- **PDF**: Portable Document Format
- **REST**: Representational State Transfer
- **UI/UX**: User Interface/User Experience
- **GDP**: Gross Domestic Product
- **CPI**: Consumer Price Index

### Appendix B: Acronyms

- **MoSPI**: Ministry of Statistics and Programme Implementation
- **RBI**: Reserve Bank of India
- **DEA**: Department of Economic Affairs
- **NSO**: National Statistical Office
- **FY**: Financial Year
- **BoP**: Balance of Payments

**END OF REPORT**