Inspira - Technical Design Document

Table of Contents

1. **Introduction**
2. **System Overview**
3. **Architecture**
   - High-Level Architecture
   - Component Interactions
4. **Backend Design**
   - Technologies Used
   - Project Structure
   - API Design
   - Database Schema
   - Middleware
5. **Frontend Design**
   - Technologies Used
   - Project Structure
   - Routing
   - State Management
   - Key Components
6. **Template Contribution and Gallery**
7. **Security Considerations**
8. **Deployment Plan**
9. **Future Enhancements**
10. **Conclusion**

1. Introduction

Inspira is a modern web application designed to simplify and manage technically required project templates such as .gitignore, README, package.json. It provides a platform for browsing, contributing, and downloading templates with features like user authentication for admin, dynamic template pages, and a searchable gallery.

2. System Overview

Inspira is built using the MERN stack (MongoDB, Express.js, React, Node.js). The system emphasizes performance, scalability, and ease of use, ensuring seamless management of templates for developers.

3. Architecture

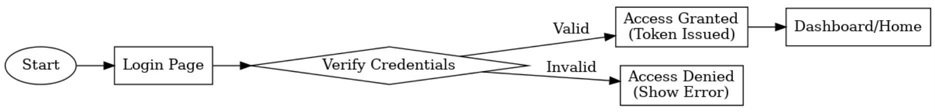
High-Level Architecture

- **Frontend**: React with Tailwind CSS for responsive user interfaces.
- **Backend**: Express.js managing APIs and middleware.
- **Database**: MongoDB storing template data and user information.

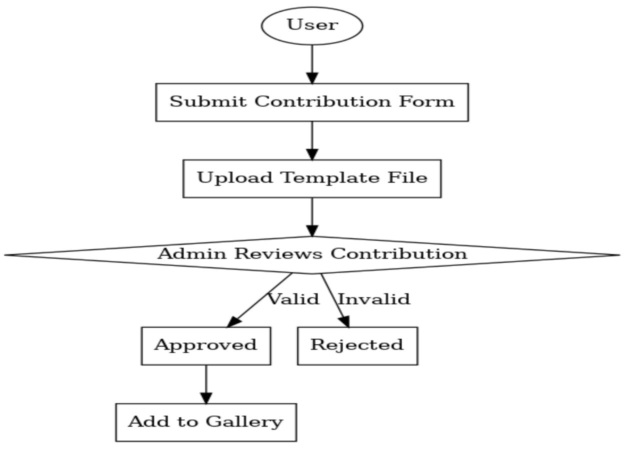
Component Interactions

- **Frontend** communicates with the backend via RESTful APIs.
- **Backend** handles business logic, user authentication, and database interactions.

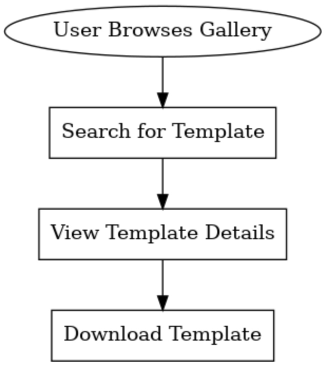
Authentication Work Flow



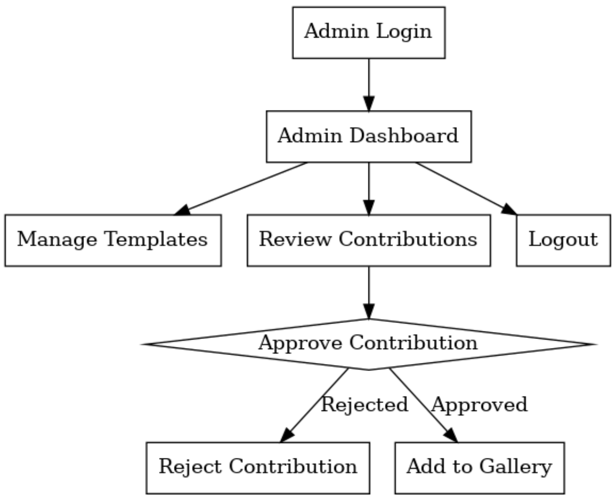
Template Contribution Work Flow



Template Gallery Work Flow



Admin Panel WorkFlow



---

4. Backend Design

Technologies Used

- **Node.js**: Runtime environment.
- **Express.js**: Web framework.
- **MongoDB**: Database for storing template and user data.
- **JWT**: Authentication.
- **Multer**: File uploads for contributions.

Project Structure

```
backend/
|
 ├── config/
|    ├── connectdb.js       # MongoDB connection logic
|   └── dotenv.config.js    # Environment variables
|
 ├── controllers/
|    ├── templateController.js  # Template-related logic
|    ├── userController.js      # User authentication logic
|    ├── contributionController.js # Contribution handling logic
|
 ├── middleware/
|    ├── authenticate.js     # JWT authentication middleware
|    ├── multer.js          # Multer configuration for file uploads (future scope)
 ├── models/
|    ├── Template.js        # Template model
|    ├── User.js           # User model
|    ├── Contribution.js     # Contribution model
 ├── routes/
|    ├── templateRoutes.js   # Routes for templates
|    ├── userRoutes.js       # Routes for authentication
|    ├── contributionRoutes.js # Routes for contributions
 ├── uploads/              # Directory to store uploaded files
 ├── server.js             # Main entry point for the backend
 ├── package.json          # Backend dependencies
└── .env                  # Environment variables
```

API Design

- **Authentication Endpoints**
    - POST /login: User login and token issuance.
    - POST /signup: User registration.
    - POST /logout: User logout.
- **Template Endpoints**
    - GET /templates: Fetch all templates.
    - GET /template/:id: Fetch a specific template.
    - POST /templates: Add a new template (Admin only).
    - DELETE /template/:id: Remove a template (Admin only).
- **Contribution Endpoints**
    - POST /contributions: Submit a new template contribution.
    - GET /contributions: Fetch all contributions for review.
    - PUT /contributions/:id/approve: Approve a contribution.

Database Schema

- **User Model**
    - email (String, required, unique)
    - password (String, required)
- **Template Model**
    - name (String, required)
    - content (String, required)
- **Contribution Model**
    - name (String, required)
    - filePath (String, required)
    - approved (Boolean, default: false)

---

5. Frontend Design

Technologies Used

- **React**: Framework.
- **Tailwind CSS**: Styling.
- **React Router DOM**: Routing.
- **Zustand**: State management.

Project Structure

```
frontend/
 ├── src/
 |   ├── api/
 |   |   ├── api.js          # Axios instance configuration
 |   |   ├── endpoints.js    # Centralized API endpoint management
 |   ├── components/
 |   |   ├── Navbar.jsx      # Main navigation bar
 |   |   ├── TemplateGallery.jsx # Displays all templates
 |   |   ├── SelfPage.jsx    # Dynamic template detail page
 |   |   ├── ContributionForm.jsx # Form for submitting contributions
 |   ├── pages/
 |   |   ├── LandingPage.jsx  # Landing page component
 |   |   ├── Login.jsx       # Login form
 |   |   ├── SignUp.jsx      # Signup form
 |   |   ├── TemplateManagementPage.jsx # Admin page for managing templates
 |   ├── stores/
 |   |   ├── authStore.js    # State management for authentication
 |   |   ├── templateStore.js  # State management for templates
 |   ├── styles/
 |   |   ├── global.css      # Global CSS styles
 |   |   ├── tailwind.css    # TailwindCSS imports and customizations
 |   ├── App.jsx             # Main app entry component
 |   ├── index.js            # React entry point
 |   ├── vite.config.js      # Vite configuration
 ├── public/
 |   ├── index.html          # Main HTML file
 ├── package.json            # Frontend dependencies
└── .env                     # Environment variables
```

Routing

- /: Landing page.
- /login: Login page.
- /signup: Signup page.
- /gallery: Template gallery.
- /template/:id: Dynamic template details.
- /admin: Template management.

Key Components

- **Navbar**: Navigation with authentication state.
- **Gallery**: Displays searchable template collection.
- **Template Page**: Dynamic page for detailed template view.
- **Contribution Form**: Form for submitting contributions.

---

6. Template Contribution and Gallery

Contribution Workflow

1. Users upload templates with optional metadata via a form.
2. Admin reviews contributions and approves valid entries.
3. Approved templates are added to the main gallery.

Gallery Features

- Search bar for filtering templates.
- Dynamic pages for individual template details.
- Download functionality for templates.

---

7. Security Considerations

- **Authentication**: Secure JWT tokens.
- **Authorization**: Role-based access controls.
- **Validation**: Middleware for sanitizing inputs.
- **Encryption**: Sensitive data encrypted with bcrypt.
- **CORS Policies**: Restrict requests to trusted origins.

---

8. Deployment Plan

- **Frontend**: Deployed on Vercel for fast, scalable hosting.
- **Backend**: Deployed on AWS EC2.
- **Database**: MongoDB Atlas with restricted IP access.
- **CI/CD**: Configured pipelines for automated deployments.

---

9. Future Enhancements

- **Template Generator**: to generate custom templates according to the requirements of the user.
- **Plagiarism Checker**: Any user that sends a contribution request, it will pass through a series of check to understand if it's already there on the website or not.
- **Template Recommendations**: AI-based suggestions.
- **Addition of New Templates**: Adding more templates options to browser through the website.

---

10. Conclusion

Inspira offers an intuitive platform for managing different type of templates required for technical projects with dynamic features and a robust backend. It provides developers a streamlined experience, with future updates aimed at enhancing usability and scalability.