COLLEGE CODE : **9628**

COLLEGE NAME: **UNIVERSITY COLLEGE OF ENGINEERING NAGERCOIL.**

DEPARTMENT :**COMPUTER SCIENCE AND ENGINEERING**

STUDENT-ID : **3791FE0EE3ABCB99BA4438D4CDA7A1E8**

REGISTER NO: **962823104078**

DATE : **15-09-2025**

COMPLETED THE PROJECT NAMED AS : -

THEME CUSTOMIZER IN WORDPRESS

SUBMITTED BY ,

NAME : **MOHAMMED RASIDH M**

MOBILE NO : **6381482861**

SOLUTION DESIGN AND ARCHITECTURE

TITLE : THEME CUSTOMIZER IN WORDPRESS

Tech Stack Solution:

Frontend:

- **Framework**: React.js with TypeScript for type safety and better development experience.
- **State Management**: Redux Toolkit for managing customizer state and theme configurations.
- **UI Components**: Material-UI or Ant Design for consistent UI elements.
- **CSS Framework**: Tailwind CSS for utility-first styling.
- **Build Tool**: Vite for fast development and optimized builds.

Backend:

- **Runtime**: Node.js with Express.js framework.
- **Database**: MySQL for storing theme configurations and user preferences.
- **ORM**: Sequelize or Prisma for database operations.
- **Authentication**: JWT tokens for secure API access.
- **File Storage**: Local file system or AWS S3 for theme assets.

WordPress Integration:

- **WordPress REST API** for communication with WordPress core.
- **Custom WordPress Plugin** to handle theme customization endpoints.
- **WordPress Hooks** for theme activation and customization events.

UI Structure / API Schema Design:

Frontend UI Structure:

```
Theme Customizer Dashboard
 ├── Header Section
 |    ├── Theme Preview Panel
 |   └── Save/Reset Controls
 ├── Sidebar Customizer Panel
 |    ├── Colors Tab
 |    ├── Typography Tab
 |    ├── Layout Tab
 |    ├── Header/Footer Tab
 |   └── Custom CSS Tab
 └── Live Preview iframe
```

API Schema Design:

```
{
"theme": {
  "id": "string",
  "name": "string",
  "version": "string",
  "customizations": {
    "colors": {
      "primary": "#color",
      "secondary": "#color",
      "accent": "#color"
    },
    "typography": {
      "headingFont": "string",
      "bodyFont": "string",
      "fontSize": "object"
    },
    "layout": {
      "containerWidth": "string",
      "sidebarPosition": "string"
    }
   }
  }
 }
```

Data Handling Approach

Data Flow Architecture:

- **Client-Side State Management**: Redux store maintains current theme customization state
- **API Communication**: RESTful API calls to save/retrieve theme configurations
- **Database Storage**: Normalized database schema for themes, customizations, and user preferences
- **WordPress Integration**: Custom hooks to apply theme changes to WordPress database
- **Real-time Preview**: WebSocket or polling mechanism for live preview updates

Data Persistence Strategy:

- **Auto-save**: Automatic saving of changes every 30 seconds
- **Version Control**: Maintain version history of theme customizations
- **Backup/Restore**: Export/import functionality for theme configurations
- **User Preferences**: Store user-specific customizer settings

Component / Module Diagram

Frontend Components:

```
App Component
 ├── ThemeCustomizer (Main Container)
│   ├── CustomizerSidebar
│   │   ├── ColorPicker
│   │   ├── FontSelector
│   │   ├── LayoutControls
│   │   └── CSSEditor
│   ├── PreviewPanel
│   │   └── ThemePreview (iframe)
│   └── ActionBar
│       ├── SaveButton
│       ├── ResetButton
│       └── ExportButton
 ├── ThemeManager
└── SettingsPanel
```

Backend Modules:

```
API Server
 ├── Routes
|    ├── /api/themes
|    ├── /api/customizations
|   └── /api/wordpress-integration
 ├── Controllers
|    ├── ThemeController
|    ├── CustomizationController
|   └── WordPressController
 ├── Services
|    ├── ThemeService
|    ├── DatabaseService
|   └── WordPressAPIService
└── Middleware
 ├── Authentication
 ├── Validation
└── ErrorHandling
```

Basic Flow Diagram:

User Interaction Flow:

- **User Access** → Load Theme Customizer Dashboard.
- **Theme Selection** → Fetch current theme configuration.
- **Customization Changes** → Update Redux state + Live preview.
- **Auto-save/Manual Save** → API call to save configuration.
- **WordPress Integration** → Update WordPress theme options.
- **Preview Generation** → Render updated theme in preview panel.

API Request Flow:

Frontend → API Gateway → Authentication → Controller → Service → Database

↓

WordPress Plugin ← WordPress REST API ← Response

Data Synchronization Flow:

- **Bidirectional sync** between customizer and WordPress database
- **Conflict resolution** for simultaneous theme modifications
- **Cache invalidation** for updated theme assets
- **Notification system** for successful/failed operations