Mobile Application

Overview

Thank you for purchasing **eLMS**. We appreciate your support. In this documentation, you will find instructions on how to set up the mobile application for eLMS. This document also provides features of eLMS.

Contact Information

By: WRTeam
Email: wrteam.anish@gmail.com

About This Documentation

This documentation provides comprehensive instructions for setting up and customizing the eLMS mobile application. You'll find detailed guides for:

- Setting up Flutter development environment
- Customizing app appearance and branding
- Integrating with Firebase
- Configuring language and localization
- Generating release versions
- And much more

If you have any questions, feel free to reach out. Thank you for choosing eLMS!

Setup Flutter

Setup flutter in your system

Visit flutter official website: https://docs.flutter.dev/get-started/install for full install guide.

If you prefer video tutorials, we recommend this playlist for the full installation process: https://www.youtube.com/playlist?list=PLSzsOkUDsvdtl3Pw48-R8lcK2oYkk40cm

Change App Name

This guide explains how to change the display name of the ELMS Flutter application on both Android and iOS devices.

Current App Name

The app is currently named **"eLMS"** in the following locations:

- Android: `android/app/src/main/AndroidManifest.xml` (line 7)
- iOS: `ios/Runner/Info.plist` (lines 10 and 18)
- In-App Display: `lib/core/configs/app_settings.dart` (line 4)

Method 1: Using rename Package (Recommended)

This is the easiest and safest method to change your app name across both platforms.

Step 1: Install rename Package

```
dart pub global activate rename
```

Step 2: Change App Name

```
dart pub global run rename setAppName --targets ios,android --value "Your App Name"
```

Replace `"Your App Name"` with your desired app name.

Example:

```
dart pub global run rename setAppName --targets ios,android --value "My Learning App"
```

Step 3: Update In-App Display Name

The `rename` package updates the platform-specific names, but you also need to manually update the in-app display name:

1. Open `lib/core/configs/app_settings.dart`
2. Change line 4:

```
static const String appName = 'Your App Name';
```

Step 4: Test

```
flutter clean
flutter pub get
flutter run
```

Method 2: Manual Change

If you prefer to change the app name manually:

Android

1. Open `android/app/src/main/AndroidManifest.xml`
2. Find line 7 with `android:label="eLMS"`
3. Change `eLMS` to your desired app name:

```
<application
    android:label="Your App Name"
```

```
    android:name="${applicationName}"
    android:icon="@mipmap/ic_launcher"
```

Example:

```
android:label="My Learning App"
```

iOS

1. Open `ios/Runner/Info.plist`
2. Find the `CFBundleDisplayName` key (line 9-10):

```
<key>CFBundleDisplayName</key>
<string>eLMS</string>
```

3. Change `eLMS` to your desired app name:

```
<key>CFBundleDisplayName</key>
<string>Your App Name</string>
```

4. Also update `CFBundleName` (line 17-18):

```
<key>CFBundleName</key>
<string>Your App Name</string>
```

Note: `CFBundleDisplayName` is what appears on the home screen.

In-App Display Name

1. Open `lib/core/configs/app_settings.dart`
2. Find line 4 with `static const String appName = 'eLMS';`
3. Change `eLMS` to your desired app name:

```
static const String appName = 'Your App Name';
```

Example:

```
static const String appName = 'My Learning App';
```

Note: This changes the app name displayed inside the app (UI components, headers, etc.).

Test

```
flutter clean
flutter pub get
flutter run
```

Troubleshooting

App name not changing on device:

- Uninstall the app completely
- Run `flutter clean`
- Rebuild and reinstall
- For iOS, try cleaning build folder in Xcode (Product → Clean Build Folder)

Quick Reference

Location	File	Line	What to Change
Android	`AndroidManifest.xml`	7	`android:label="eLMS"`

Location	File	Line	What to Change
iOS	`Info.plist`	10	`<string>eLMS</string>` (CFBundleDisplayName)
iOS	`Info.plist`	18	`<string>eLMS</string>` (CFBundleName)
In-App	`app_settings.dart`	4	`appName = 'eLMS'`

Checklist

- [ ] Choose your method (rename package or manual)
- [ ] Update app name on both Android and iOS
- [ ] Run `flutter clean && flutter pub get`
- [ ] Test on Android device/emulator
- [ ] Test on iOS device/simulator
- [ ] Verify new name appears on home screen

Change Package Name

1. Unzip the downloaded code. After unzipping you will have eLMS - Flutter Code zip folder. Unzip that folder and open it in Android Studio or Visual Studio Code.

2. Open IDE terminal, go to your project path and execute command:

```
flutter pub get
```

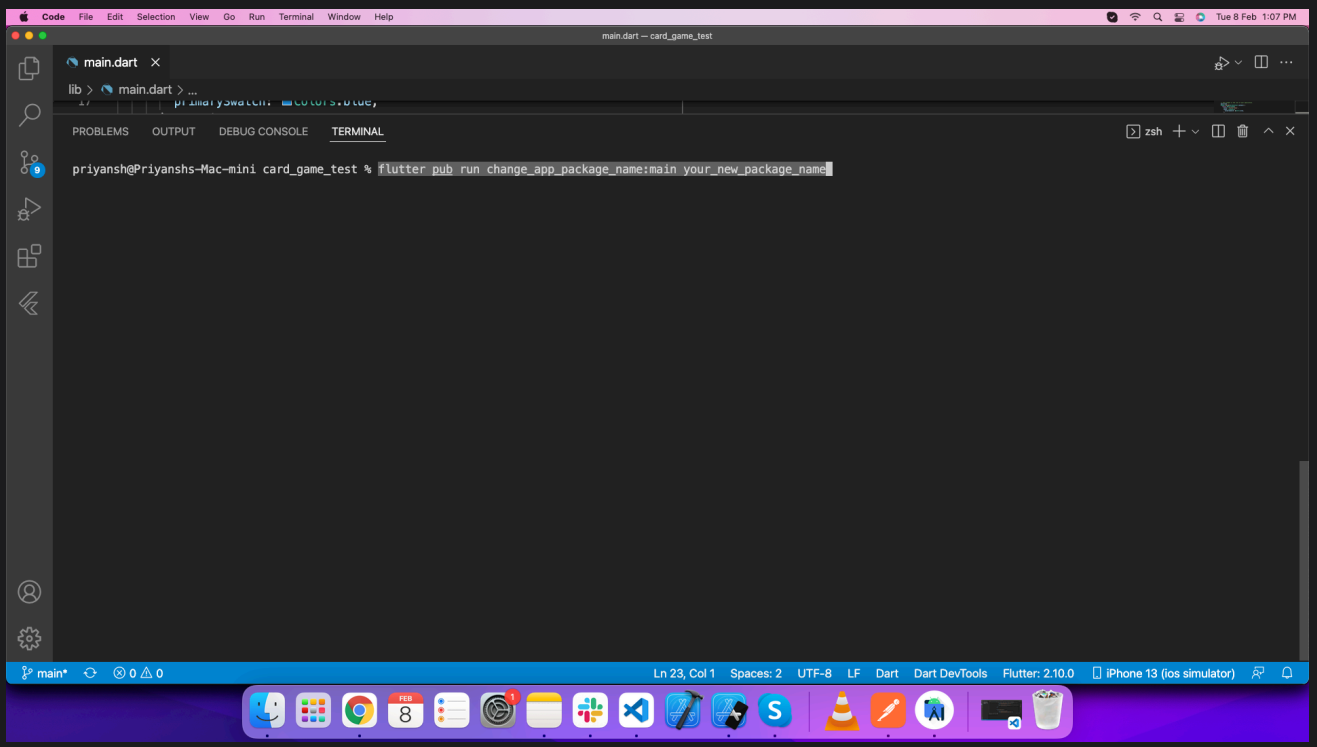
3. If you are running this app for iOS then run these following commands in terminal:

```
cd ios
pod install
cd ..
```

4. Change package name of Android app:

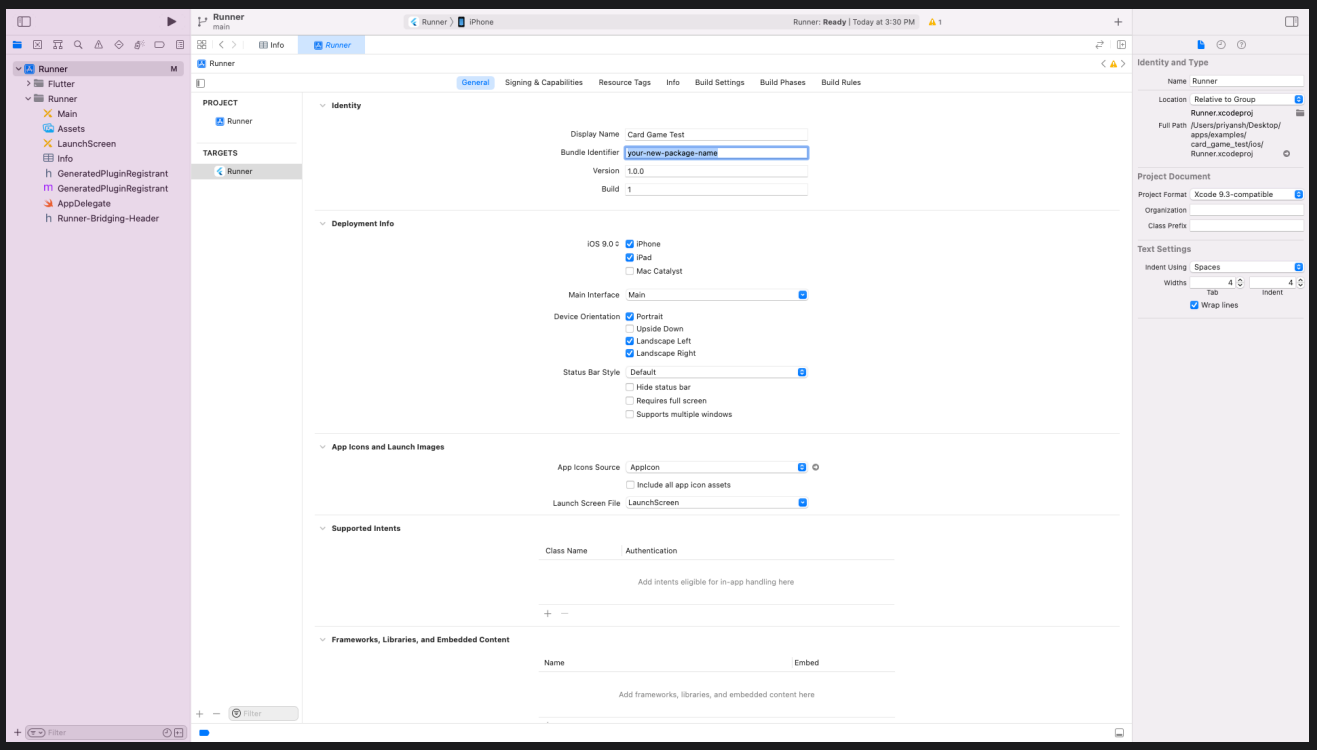
Execute this command in your terminal:

```
flutter pub run change_app_package_name:main your_new_package_name
```



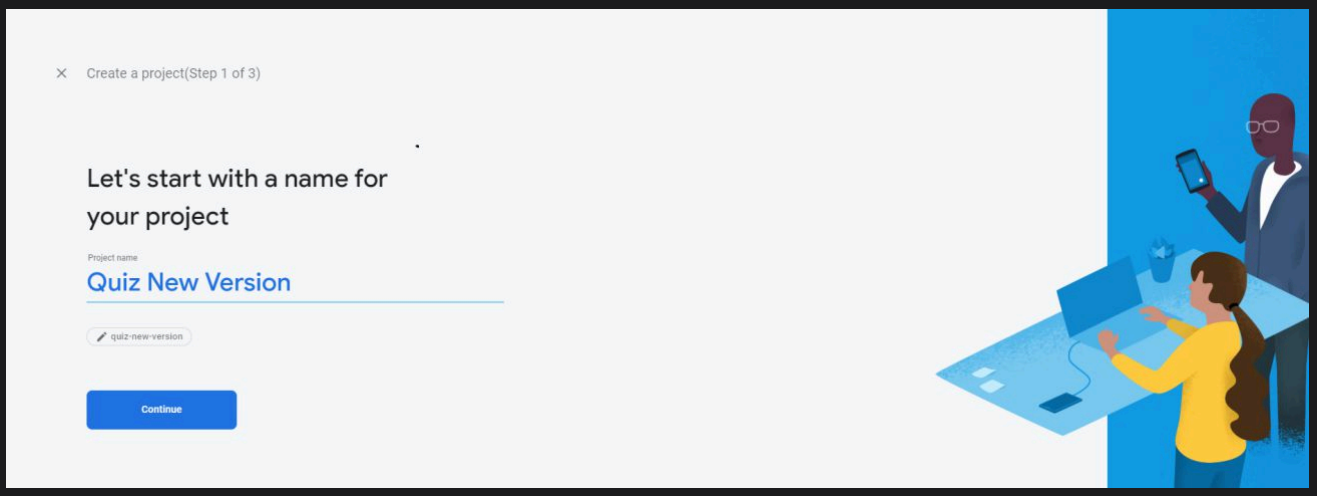
5. Change package name of iOS app:

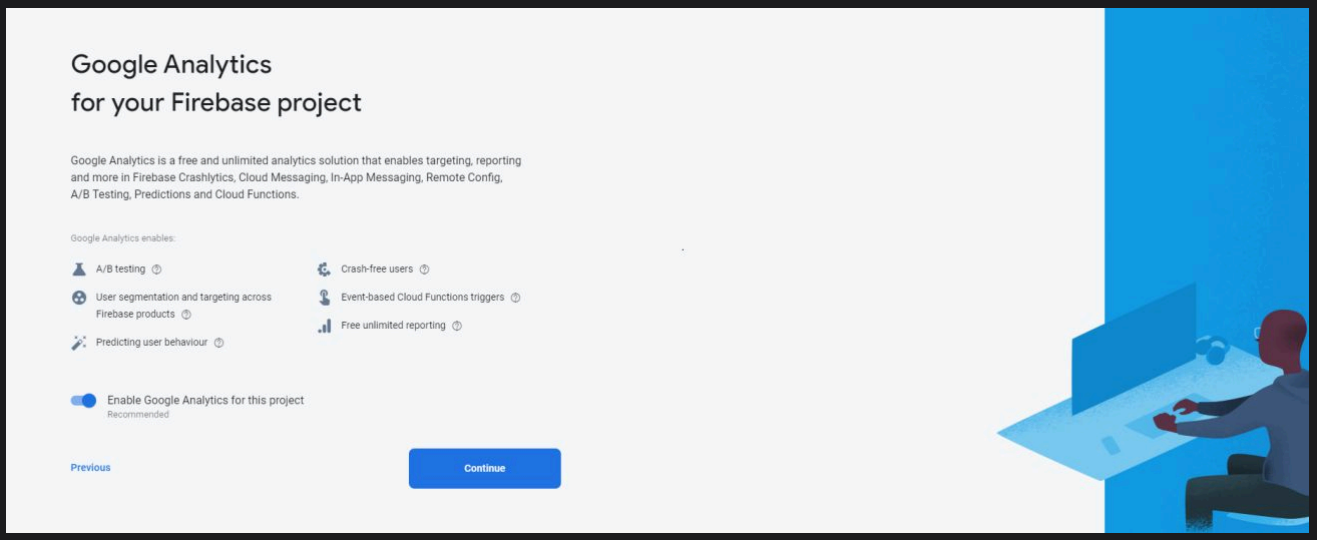
Open iOS folder of this project in Xcode. Go Select Runner->Targets->General->Identity and enter new package name in Build Identifier.

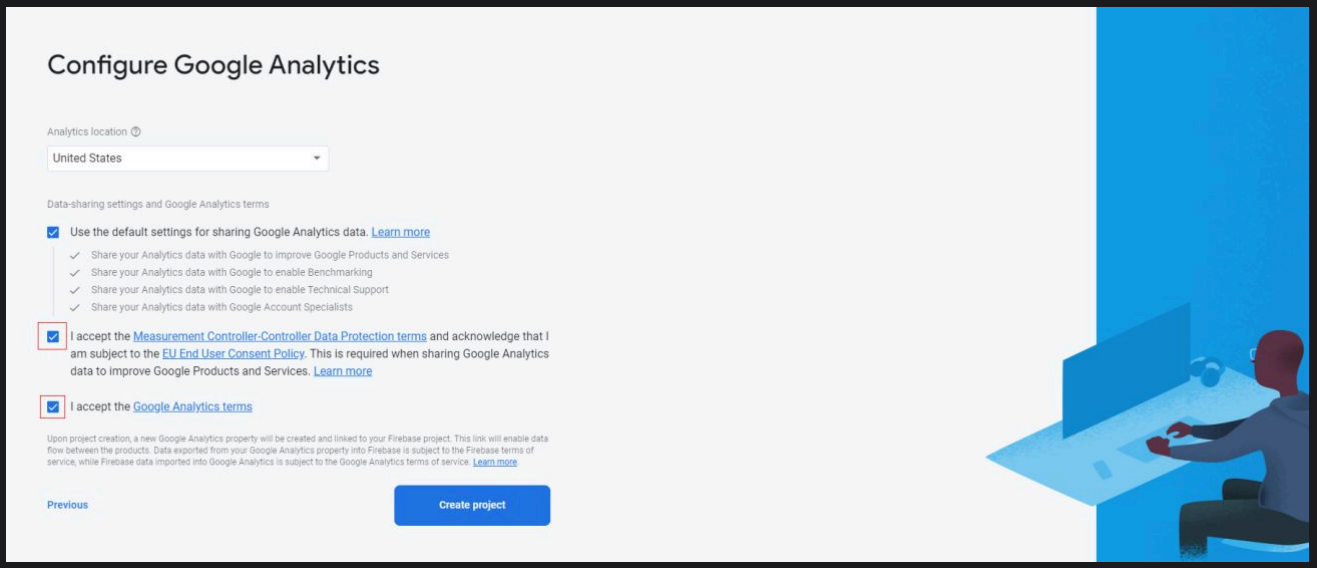


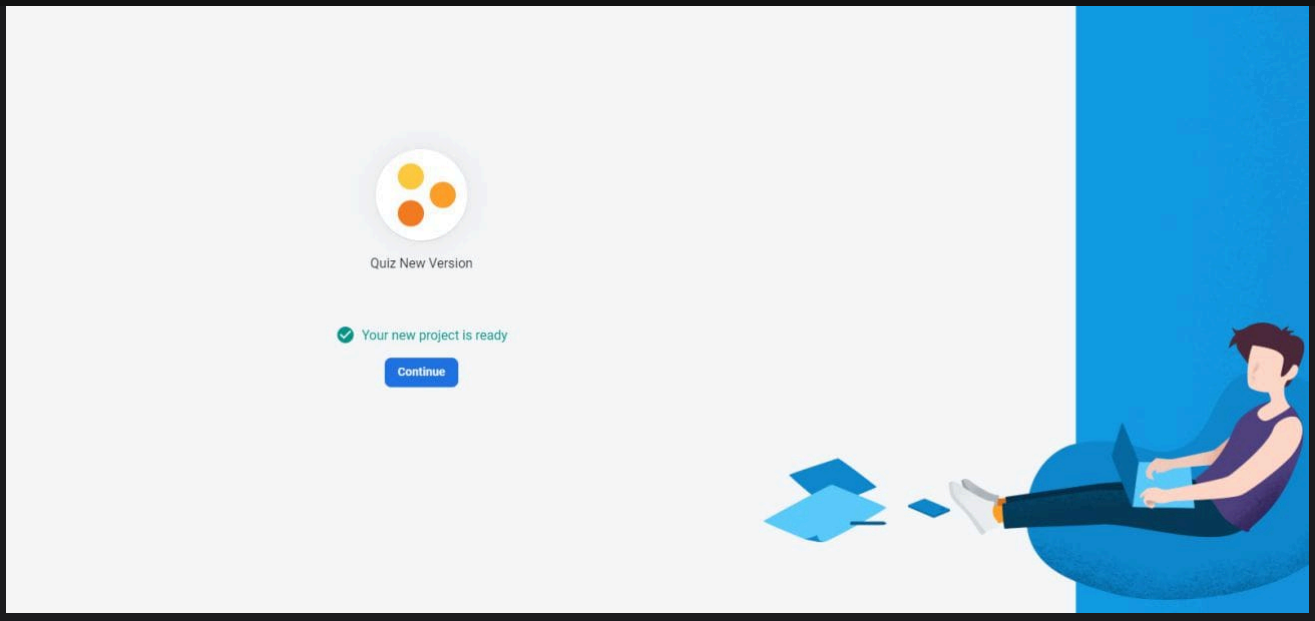
Integrate with Firebase

1. Create Firebase project in your account

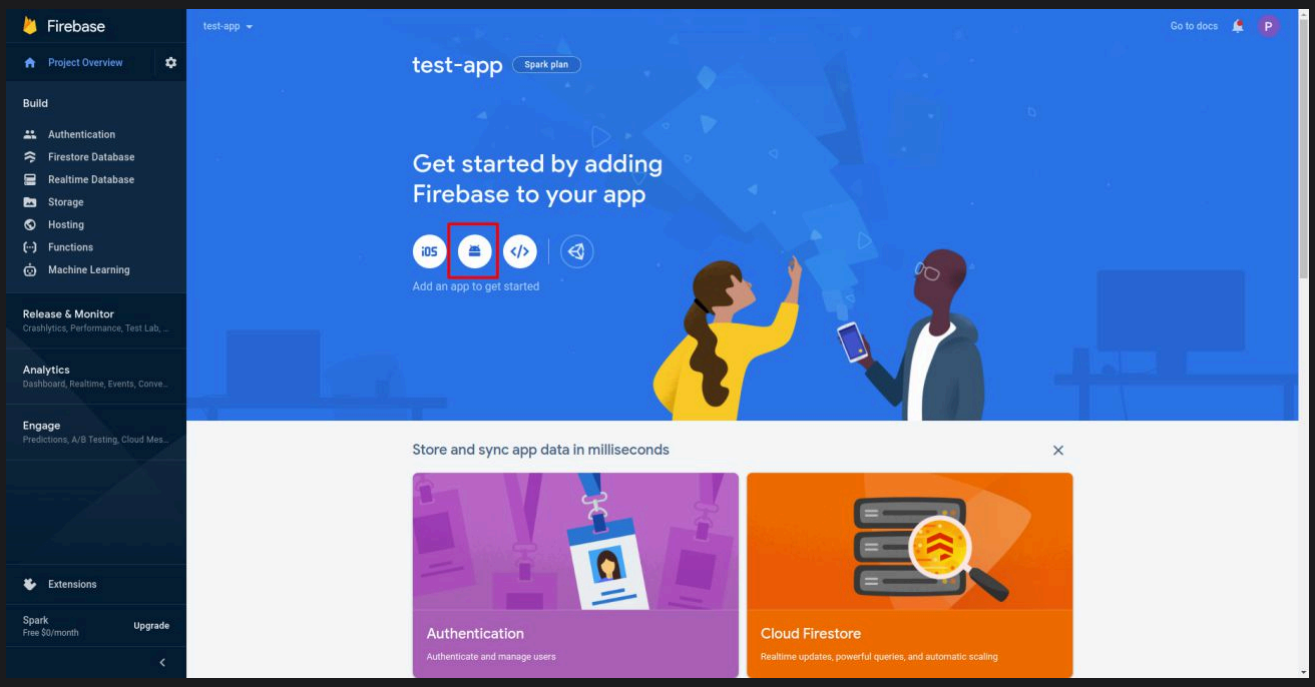








2. Add android application to your Firebase project



Download the google-service.json file and add in this folder android/app/

3. Add ios application to your Firebase project

4. Download GoogleService-Info.plist and add in this folder ios/Runner/

5. Please configure this settings in-order to send ios notifications.

https://firebase.flutter.dev/docs/messaging/apple-integration

6. You have configured Firebase in your project successfully

Additional Resources: For detailed Firebase setup and configuration: Firebase Setup Guide

Firebase Auth

This guide mirrors the structure of our other mobile setup docs. It walks through the Firebase Console work plus the exact locations for the generated configuration files inside the Android and iOS projects. Follow it whenever you onboard a new Firebase project or rotate credentials.

Prerequisites

- Access to the Firebase Console project used by ELMS.
- Local copies of the Flutter source code (Android and iOS folders included).
- Android Studio / Xcode installed for editing native files.
- Apple Developer access for the bundle ID that ships with the app.

Firebase Console Configuration

Step 1: Enable Authentication Providers

1. Navigate to **Firebase Console → Build → Authentication** and click **Get started** if prompted.
2. Under **Sign-in method**:
    - **Email/Password** → toggle **Enable** → **Save**.
    - **Phone** → toggle **Enable** → **Save**.
    - **Google** → toggle **Enable**, pick a **Project support email**, then **Save**.
    - **Apple** (iOS only) → toggle **Enable** → **Save**. Ensure your Apple Developer account already owns the bundle ID with “Sign in with Apple” capability.

Step 2: Gather Identifiers & Fingerprints

Platform	Required value	Where to find it
Android	Package name (`applicationId`)	`android/app/build.gradle`
Android	SHA-1 & SHA-256 fingerprints (debug + release)	`keytool -list -v -keystore ...`
iOS	Bundle ID	`ios/Runner.xcodeproj` → **Runner** target → **General**
iOS	Apple Team ID	developer.apple.com/account

Add these items under **Project settings → Your apps** before downloading any configuration files.

Step 3: Download Config Files

1. Still in **Project settings → Your apps**, select the Android app → click **Download `google-services.json`**.
2. Select the iOS app → click **Download `GoogleService-Info.plist`**.
3. Repeat this download any time you change SHA fingerprints (Android) or bundle identifiers (iOS).

Android Native Configuration

Step 4: Place `google-services.json`

1. Copy the downloaded file into `android/app/`.
2. Confirm the Gradle project is wired correctly:
   - `android/build.gradle` must include `classpath 'com.google.gms:google-services:X.X.X'`.
   - `android/app/build.gradle` must apply the plugin at the bottom:

```
apply plugin: "com.google.gms.google-services"
```

3. Rebuild the Android project so Gradle ingests the updated JSON.

---

iOS Native Configuration

Step 5: Add `GoogleService-Info.plist` to Runner

1. Copy the plist into `ios/Runner/`.
2. Open `ios/Runner.xcworkspace` in Xcode.
3. Drag the plist into the **Runner** target (check **Copy items if needed**).
4. Verify it is listed under **Build Phases → Copy Bundle Resources** to ensure the file ships with the app.

Step 6: Register the Reversed Client ID (Google Sign-In)

1. Open `GoogleService-Info.plist` and copy the `REVERSED_CLIENT_ID` value (for example `com.googleusercontent.apps.12345-abcdef`).
2. In Xcode → select the **Runner** target → **Info** tab → expand **URL Types**.
3. Click **+** and fill in:
   - **Identifier**: `com.google`
   - **URL Schemes**: paste the `REVERSED_CLIENT_ID`.
4. Save to commit the change to `ios/Runner/Info.plist`. The relevant block should look like:

```
<key>CFBundleURLTypes</key>
<array>
  <dict>
    <key>CFBundleTypeRole</key>
    <string>Editor</string>
    <key>CFBundleURLSchemes</key>
    <array>
      <string>com.googleusercontent.apps.12345-abcdef</string>
      <!-- Other schemes such as deep links can follow -->
    </array>
```

```
  </dict>
</array>
```

Without this entry, Google Sign-In cannot hand control back to your app after authentication.

Step 7: Enable Apple Sign-In Capability

1. Sign in to developer.apple.com with the Team that matches your bundle ID and ensure the App ID has **Sign in with Apple** enabled. Download refreshed provisioning profiles if Apple prompts you.
2. In Xcode → **Runner** target → **Signing & Capabilities**, click **+ Capability**, search for **Sign in with Apple**, and add it.
3. Open `ios/Runner/Runner.entitlements` to confirm it contains:

   ```
   <key>com.apple.developer.applesignin</key>
   <array>
     <string>Default</string>
   </array>
   ```

   Keep this file under source control so teammates inherit the capability.
4. Build the app on a physical iOS device (iOS 13+) to verify the Apple Sign-In sheet appears and stays open.

Change App Theme

This guide explains how to customize the color scheme of the ELMS (E-Learning Management System) Flutter application for both light and dark themes.

Overview

The ELMS app uses a centralized color management system that supports both light and dark themes. All theme colors are defined in a single file, making it easy to customize the entire app's appearance.

File Location

Main Color Configuration File:

```
lib/core/constants/app_colors.dart
```

Color Structure

The app defines colors in three categories:

1. Light Mode Colors

These colors are used when the app is in light theme mode:

Color Name	Default Value	Purpose
`primaryColor`	`#5A5BB5` (Purple)	Main theme color used for primary actions, buttons, and branding
`secondaryColor`	`#FFFFFF` (White)	Card backgrounds and form field backgrounds

Color Name	Default Value	Purpose
`backgroundColor`	`#F2F5F7` (Light Gray)	Screen background color
`borderColor`	`#D8E0E6` (Gray)	Border color for inputs, cards, and dividers
`errorColor`	`#DB3D26` (Red)	Error messages and validation states

2. Dark Mode Colors

These colors are used when the app is in dark theme mode:

Color Name	Default Value	Purpose
`darkPrimaryColor`	`#7273D3` (Light Purple)	Main theme color for dark mode
`darkSecondaryColor`	`#787878` (Gray)	Card and field backgrounds in dark mode
`darkBackgroundColor`	`#101010` (Almost Black)	Screen background in dark mode
`darkBorderColor`	`#343536` (Dark Gray)	Borders in dark mode
`darkErrorColor`	`#DB3D26` (Red)	Error messages in dark mode

3. Custom Constant Colors

These colors remain the same in both light and dark modes:

Color Name	Default Value	Purpose
`infoColor`	`#0186D8` (Blue)	Information messages and notifications
`warningColor`	`#E29512` (Orange)	Warning messages
`successColor`	`#34A853` (Green)	Success messages
`darkColor`	`#000000` (Black)	Text and icons (adapts in dark mode)

How to Change Colors

Step 1: Open the Color Configuration File

Navigate to and open:

```
lib/core/constants/app_colors.dart
```

Step 2: Modify the Desired Colors

The file contains static color constants. To change a color, simply update its hex value:

```
// Example: Changing primary color from purple to blue
static const Color primaryColor = Color(0xff5A5BB5);  // Old purple
static const Color primaryColor = Color(0xff2196F3);  // New blue
```

Step 3: Color Format

Colors are defined using hexadecimal values with the `Color()` constructor:

Format Options:

1. **Hex with alpha (most common):**

```
Color(0xffRRGGBB)
// ff = alpha (opacity), RR = red, GG = green, BB = blue
```

2. **ARGB format:**

```
Color.fromARGB(255, red, green, blue)
// 255 = fully opaque, then RGB values from 0-255
```

3. **RGBA format:**

```
Color.fromRGBO(red, green, blue, opacity)
// RGB values from 0-255, opacity from 0.0 to 1.0
```

Example: Complete Theme Color Change

Let's say you want to change the app to use a green theme:

Light Mode Changes

```
// Primary color to green
static const Color primaryColor = Color(0xff4CAF50);

// Keep secondary (white) as is
static const Color secondaryColor = Color(0xffffffff);

// Light green background
static const Color backgroundColor = Color(0xffE8F5E9);

// Green-tinted border
static const Color borderColor = Color(0xffC8E6C9);

// Keep error red
static const Color errorColor = Color(0xffDB3D26);
```

Dark Mode Changes

```
// Dark mode primary - lighter green
static const Color darkPrimaryColor = Color(0xff81C784);

// Keep dark secondary as is
static const Color darkSecondaryColor = Color.fromRGBO(120, 120,
120, 1);

// Dark green background
static const Color darkBackgroundColor = Color(0xff1B5E20);

// Dark green border
static const Color darkBorderColor = Color(0xff2E7D32);

// Keep error red
static const Color darkErrorColor = Color(0xffDB3D26);
```

Theme Mapping

The colors from `AppColors` are mapped to Flutter's `ColorScheme` in `lib/core/theme/app_theme.dart`:

Light Theme Mapping

- `primary` → `primaryColor`
- `surface` → `secondaryColor` (cards, sheets)
- `outline` → `borderColor`
- `error` → `errorColor`
- `scaffoldBackgroundColor` → `backgroundColor`

Dark Theme Mapping

- `primary` → `darkPrimaryColor`
- `surface` → `darkBackgroundColor` (slightly brightened)
- `outline` → `darkBorderColor`
- `error` → `darkErrorColor`
- `scaffoldBackgroundColor` → `darkBackgroundColor`

Testing Your Changes

After changing colors:

1. **Hot Reload:** Press `r` in your terminal or use your IDE's hot reload feature
2. **Test Both Themes:**
   - Switch between light and dark mode in the app
   - Verify all screens look correct
3. **Check Contrast:** Ensure text is readable on all backgrounds
4. **Test Components:**
   - Buttons
   - Text fields
   - Cards
   - Dialogs
   - Navigation bars

Best Practices

1. **Maintain Contrast:** Ensure sufficient contrast between text and backgrounds for accessibility
2. **Consistent Dark Mode:** Dark mode colors should be lighter versions of light mode colors
3. **Brand Consistency:** Use colors that align with your brand identity
4. **Test Thoroughly:** Always test both light and dark themes after changes
5. **Use Color Tools:** Use tools like Coolors.co or Adobe Color to generate harmonious color palettes

Color Accessibility

Follow WCAG 2.1 guidelines for color contrast:

- **Normal text:** Minimum contrast ratio of 4.5:1
- **Large text:** Minimum contrast ratio of 3:1
- **UI components:** Minimum contrast ratio of 3:1

Test your colors using tools like WebAIM Contrast Checker.

Getting Color Codes

From Design Tools

- **Figma:** Select color → Copy as CSS → Extract hex
- **Adobe XD:** Use color picker → Copy hex value
- **Sketch:** Color picker → Hex value

From Images

- Use online color pickers like ImageColorPicker.com
- Upload your logo/brand image to extract exact colors

Converting Colors to Flutter Format

If you have a hex color like `#5A5BB5`:

```
// Add 'ff' prefix for full opacity and '0x' for Flutter
Color(0xff5A5BB5)
```

Troubleshooting

Q: My changes aren't showing up

- Try hot restart instead of hot reload (press `R` in terminal)
- Rebuild the entire app

Q: Colors look different in dark mode

- Make sure you updated both light AND dark color variants
- Check `app_theme.dart` for any color transformations

Q: Text is not readable

- Increase contrast between text and background colors
- Check if you need to adjust text colors separately

Q: Some parts of the app didn't change color

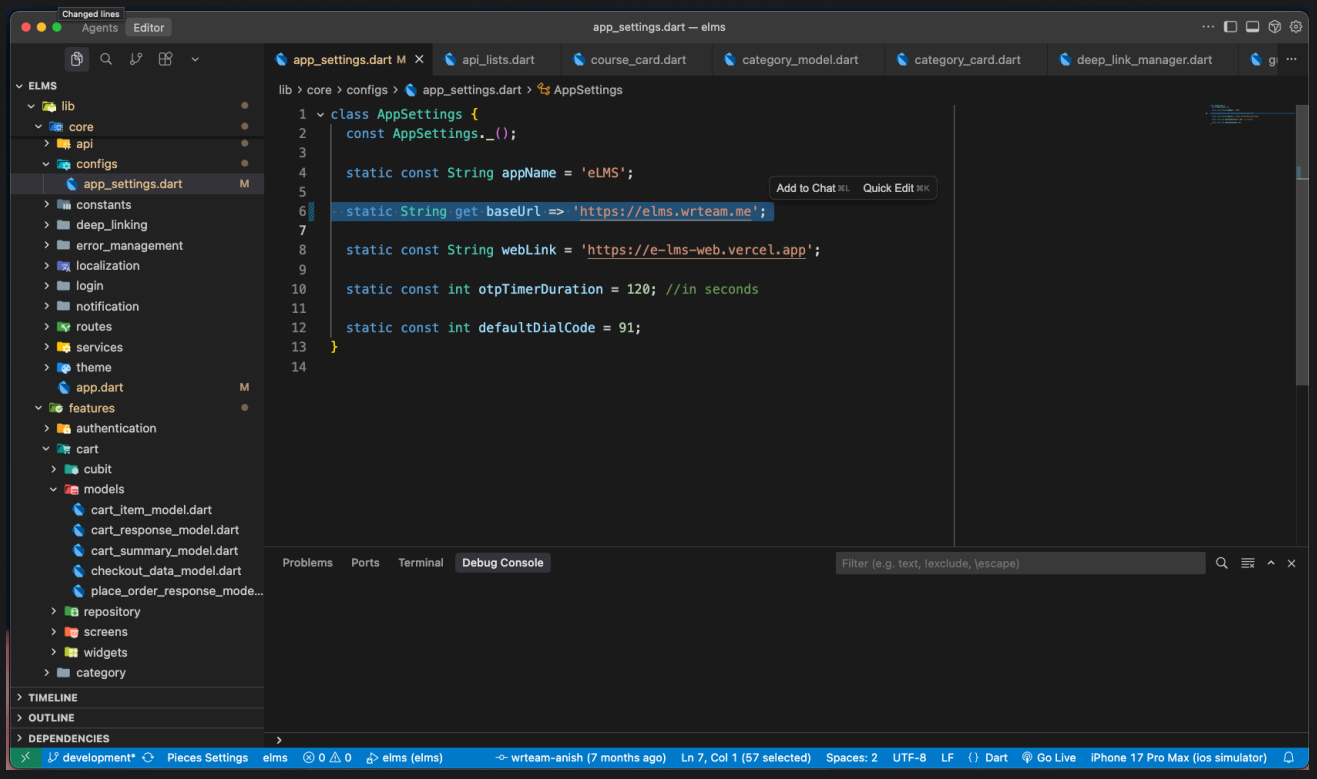
- Some widgets might use hardcoded colors (not recommended)
- Search for `Color(0x` in the codebase to find hardcoded colors

Additional Resources

- **Color Palette Generators:**
  - Coolors.co
  - Material Design Color Tool
  - Adobe Color
- **Accessibility Testing:**
  - WebAIM Contrast Checker
  - Colorable
- **Flutter Documentation:**
  - Material Design Colors
  - ColorScheme Class

App Settings

Configure your ELMS app settings by modifying the values in `lib/core/configs/app_settings.dart`.



Configuration File Location

```
lib/core/configs/app_settings.dart
```

Available Settings

1. App Name

```
static const String appName = 'eLMS';
```

Description: The display name of your application.

Usage:

- Used throughout the app for branding
- Displayed in app title bars and navigation
- Referenced in `lib/core/app.dart` for the MaterialApp title

How to Change:

```
static const String appName = 'Your App Name';
```

Example:

```
static const String appName = 'My Learning Platform';
```

---

2. Base URL (Backend API)

```
static String get baseUrl => 'https://elms.wrteam.me';
```

Description: The base URL of your backend server/admin panel. This is the primary server endpoint that the app connects to for all API calls.

Usage:

- All API requests are made to this URL
- Used by the API client in `lib/core/api/api_client.dart`
- Critical for app functionality - the app cannot work without a valid backend URL

How to Change:

```
static String get baseUrl => 'https://your-domain.com';
```

Examples:

```
// Production server
static String get baseUrl => 'https://api.mylearningapp.com';
```

```
// Staging server
static String get baseUrl => 'https://staging-
api.mylearningapp.com';

// Local development server
static String get baseUrl => 'http://192.168.1.100:8000';

// Local emulator (Android)
static String get baseUrl => 'http://10.0.2.2:8000';
```

Important Notes:

- Must be a valid HTTP/HTTPS URL
- Should NOT end with a trailing slash
- Ensure your server has CORS configured for mobile apps
- Use HTTPS in production for security
- For local development, use your computer's IP address (not localhost)

---

3. Web Link

```
static const String webLink = 'https://e-lms-web.vercel.app';
```

Description: The URL of your web application or website. This is used when you need to redirect users to the web version of your platform.

Usage:

- Deep linking and web redirects
- Sharing links that work on both mobile and web
- Fallback for features not available in mobile app
- Terms of service, privacy policy, or help pages

How to Change:

```
static const String webLink = 'https://your-website.com';
```

Examples:

```
static const String webLink = 'https://www.mylearningapp.com';
static const String webLink = 'https://learn.mycompany.com';
```

---

4. OTP Timer Duration

```
static const int otpTimerDuration = 120; // in seconds
```

Description: The duration (in seconds) for OTP (One-Time Password) validity and resend timer.

Usage:

- Controls how long the user has to enter an OTP before it expires
- Sets the countdown timer before "Resend OTP" button becomes active
- Used in authentication flows (phone/email verification)

How to Change:

```
static const int otpTimerDuration = 60; // 1 minute
static const int otpTimerDuration = 180; // 3 minutes
static const int otpTimerDuration = 300; // 5 minutes
```

Recommended Values:

- **60 seconds** - Quick verification, better security
- **120 seconds** (default) - Balanced approach
- **180-300 seconds** - More user-friendly for slower networks

Important Notes:

- Value is in seconds
- Longer durations are more user-friendly but less secure
- Should match or be shorter than backend OTP expiry time
- Consider your target audience and typical network conditions

---

5. Default Dial Code

```
static const int defaultDialCode = 91;
```

Description: The default country dial code for phone number input fields.

Usage:

- Pre-fills the country code selector in phone number fields
- Used during phone authentication/registration
- Referenced in country code picker components

How to Change:

```
static const int defaultDialCode = 1;  // United States/Canada
static const int defaultDialCode = 44; // United Kingdom
static const int defaultDialCode = 971; // UAE
```

How to Choose:

- Set to the primary country of your target audience
- Consider where most of your users are located
- Users can still change it manually during registration

Complete Configuration Example

Here's an example configuration for a custom learning platform:

```
class AppSettings {
  const AppSettings._();

  // Your app's display name
  static const String appName = 'ExampleHub Pro';

  // Your production API server
```

```
  static String get baseUrl => 'https://api.examplehubpro.com';

  // Your website URL
  static const String webLink = 'https://www.examplehubpro.com';

  // OTP valid for 3 minutes
  static const int otpTimerDuration = 180;

  // Default to United States
  static const int defaultDialCode = 1;
}
```

Change App Logo & Assets

This guide explains how to update branding assets (logos and icons) for the ELMS Flutter application.

Current Asset Structure

The ELMS app has the following asset organization:

- **Icons and logos**: `assets/icons/`
- **General images**: `assets/images/`
- **Illustrators**: `assets/images/illustrators/`

Current Logo Files

File	Location	
`splash_logo.svg`	`assets/icons/`	Ap spl ref Ap
Launcher icons	`android/app/src/main/res/mipmap-*/` and `ios/Runner/Assets.xcassets/AppIcon.appiconset/`	De icc

Current Illustrator Assets

These files can be customized for visual branding:

- `assets/images/illustrators/error.svg`
- `assets/images/illustrators/no_data.svg`
- `assets/images/illustrators/no_internet.svg`

Onboarding Assets

- `assets/icons/onboarding_1.svg`
- `assets/icons/onboarding_2.svg`
- `assets/icons/onboarding_3.svg`
- `assets/icons/onboarding_bg.svg`

Method 1: Quick Logo Update (In-App Logo Only)

This method updates the logo shown inside the app (splash screen, etc.) without changing launcher icons.

Step 1: Replace the Logo File

Simply replace the existing file with your new logo:

```
# Backup current logo (optional)
cp assets/icons/splash_logo.svg assets/icons/splash_logo_backup.svg

# Replace with your logo (keep the same filename)
# Copy your logo file to: assets/icons/splash_logo.svg
```

Important: Keep the filename as `splash_logo.svg` - this is referenced in `lib/core/constants/app_icons.dart` as `AppIcons.appLogo`

Step 2: Test

```
flutter clean
flutter pub get
flutter run
```

Method 2: Update Launcher Icons (Auto-Generate)

This method updates the app icon shown on device home screens using an automated tool.

Step 1: Install flutter_launcher_icons

Add to `pubspec.yaml` under `dev_dependencies`:

```
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^5.0.0
  flutter_launcher_icons: ^0.14.2  # Add this line
```

Step 2: Configure Icon Generation

Add this configuration at the end of `pubspec.yaml`:

```
flutter_launcher_icons:
  android: true
  ios: true
  image_path: "assets/icons/ic_launcher.png"
  adaptive_icon_background: "#FFFFFF"  # Change to your brand color
  adaptive_icon_foreground:
"assets/icons/ic_launcher_transparent.png"
  remove_alpha_ios: true
```

Step 3: Prepare Icon Files

Create two icon files in `assets/icons/`:

1. **`ic_launcher.png`** - Square logo (1024x1024px recommended, solid background)
2. **`ic_launcher_transparent.png`** - Logo with transparent background (for Android adaptive icons)

Step 4: Generate Icons

```
flutter pub get
dart run flutter_launcher_icons
```

This automatically generates all required icon sizes for Android and iOS.

Step 5: Test

```
flutter clean
flutter pub get
flutter run
```

Method 3: Manual Launcher Icon Update

If you prefer not to use the automated tool:

Android

1. Generate icons in various sizes:
    - `mipmap-ldpi/` - 36x36
    - `mipmap-mdpi/` - 48x48
    - `mipmap-hdpi/` - 72x72
    - `mipmap-xhdpi/` - 96x96
    - `mipmap-xxhdpi/` - 144x144
    - `mipmap-xxxhdpi/` - 192x192
2. Place files in: `android/app/src/main/res/mipmap-*/ic_launcher.png`

iOS

1. Generate all required sizes:
    - 20x20, 29x29, 40x40, 60x60, 76x76, 83.5x83.5, 1024x1024

2. Replace icons in: `ios/Runner/Assets.xcassets/AppIcon.appiconset/`

3. Update filenames in `Contents.json` if needed

Rebuild

```
flutter clean
flutter pub get
flutter run
```

Updating Other Assets

Onboarding Screens

Replace these files to customize onboarding:

- `assets/icons/onboarding_1.svg`
- `assets/icons/onboarding_2.svg`
- `assets/icons/onboarding_3.svg`
- `assets/icons/onboarding_bg.svg`

Error/Empty State Illustrations

Replace these files in `assets/images/illustrators/`:

- `error.svg` - Shown on error screens
- `no_data.svg` - Shown when no data available
- `no_internet.svg` - Shown when offline

Icon Reference System

The app uses a centralized icon management system in `lib/core/constants/app_icons.dart`.

How Icons are Referenced

```
// In app_icons.dart
static final String appLogo = _getSvg('splash_logo');

// Usage in code
CustomImage(imagePath: AppIcons.appLogo)
```

Adding New Icons

If you add a new icon to `assets/icons/`, register it in `app_icons.dart`:

```
static final String myNewIcon = _getSvg('my_new_icon');
```

Notification Icons (Android)

The app uses `awesome_notifications` package. If you need to update notification icons, check the notification initialization code for icon references.

Troubleshooting

Icons not updating on device:

- Uninstall the app completely and reinstall
- Run `flutter clean` before rebuilding
- For iOS, clean Xcode build folder (Product → Clean Build Folder)

SVG not displaying:

- Verify SVG file is valid (the app uses `flutter_svg` package which is already installed)
- Check file path is correct in `app_icons.dart`
- Ensure asset is listed in `pubspec.yaml`

Launcher icon not changing:

- Make sure you uninstall the old app before installing new version
- Clear device cache if needed
- For Android, check if adaptive icon files are properly generated

---

Quick Reference Checklist

For In-App Logo Only:

- [ ] Replace `assets/icons/splash_logo.svg` with your logo
- [ ] Run `flutter clean && flutter pub get && flutter run`
- [ ] Test splash screen shows new logo

For Launcher Icons:

- [ ] Create `assets/icons/ic_launcher.png` (1024x1024)
- [ ] Create `assets/icons/ic_launcher_transparent.png`
- [ ] Add `flutter_launcher_icons` to `pubspec.yaml`
- [ ] Configure `flutter_launcher_icons` settings
- [ ] Run `flutter pub get`
- [ ] Run `dart run flutter_launcher_icons`
- [ ] Run `flutter clean && flutter pub get && flutter run`
- [ ] Uninstall old app and install fresh to verify icons

File Locations Reference

Asset Type	Current Location
App logo (in-app)	`assets/icons/splash_logo.svg`

Asset Type	Current Location
App icons reference	`lib/core/constants/app_icons.dart`
Android launcher icons	`android/app/src/main/res/mipmap-*/`
iOS launcher icons	`ios/Runner/Assets.xcassets/AppIcon.appiconset/`
Illustrators	`assets/images/illustrators/`
Onboarding images	`assets/icons/onboarding_*.svg`

Change font

This guide explains how to customize the font family used throughout the ELMS (E-Learning Management System) Flutter application.

Overview

The ELMS app uses a centralized font management system where the font family is defined once and applied globally throughout the entire application. The current app uses the **Geist** font family.

Current Font Configuration

Current Font: Geist **Font Weights Available:** 100, 200, 300, 400, 500, 600, 700, 800, 900

File Locations

1. Theme Configuration File

```
lib/core/theme/app_theme.dart
```

This file contains the `fontFamily` constant that sets the global font for the entire app.

2. Font Assets Directory

```
assets/fonts/
```

This directory contains all the `.ttf` (TrueType Font) files for your custom fonts.

3. Font Registration File

`pubspec.yaml`

This file registers your fonts with Flutter so they can be used in the app.

Step-by-Step Guide to Change Fonts

Step 1: Obtain Your Font Files

Before you begin, you need to have your font files ready:

Supported Font Formats:

- `.ttf` (TrueType Font) - Most common and recommended
- `.otf` (OpenType Font) - Also supported

What You Need:

- Font files for different weights (Regular, Bold, Light, etc.)
- At minimum: Regular (400) and Bold (700) weights
- Optionally: Multiple weights for better typography

Where to Get Fonts:

1. **Google Fonts** (Free): https://fonts.google.com
    - Download any font family
    - Click "Download family" to get all weights
2. **Font Squirrel** (Free): https://www.fontsquirrel.com
    - High-quality free fonts for commercial use
3. **Adobe Fonts** (Paid): https://fonts.adobe.com
    - Premium fonts with Adobe subscription
4. **Custom Fonts**: Use your own brand fonts
    - Get from your design team

- Ensure you have proper licensing

Step 2: Add Font Files to Assets

1. **Navigate to the fonts directory:**

```
assets/fonts/
```

2. **Add your font files:**

   - Copy all your `.ttf` or `.otf` files to this directory
   - Use clear naming convention: `FontName-Weight.ttf`

   Example:

```
assets/fonts/
├── Roboto-Regular.ttf
├── Roboto-Bold.ttf
├── Roboto-Light.ttf
├── Roboto-Medium.ttf
└── Roboto-Black.ttf
```

3. **Naming Best Practices:**

   - Keep filenames consistent
   - Use hyphen or underscore between name and weight
   - Examples: `Roboto-Bold.ttf`, `OpenSans-Regular.ttf`, `Poppins-SemiBold.ttf`

Step 3: Register Fonts in pubspec.yaml

Open `pubspec.yaml` and update the fonts section:

Current Configuration (Geist font):

```
fonts:
  - family: Geist
    fonts:
```

```
      - asset: assets/fonts/Geist-Black.ttf
        weight: 900
      - asset: assets/fonts/Geist-Bold.ttf
        weight: 700
      - asset: assets/fonts/Geist-ExtraBold.ttf
        weight: 800
      - asset: assets/fonts/Geist-ExtraLight.ttf
        weight: 200
      - asset: assets/fonts/Geist-Light.ttf
        weight: 300
      - asset: assets/fonts/Geist-Medium.ttf
        weight: 500
      - asset: assets/fonts/Geist-Regular.ttf
        weight: 400
      - asset: assets/fonts/Geist-SemiBold.ttf
        weight: 600
      - asset: assets/fonts/Geist-Thin.ttf
        weight: 100
```

Example: Changing to Roboto font:

```
fonts:
  - family: Roboto
    fonts:
      - asset: assets/fonts/Roboto-Thin.ttf
        weight: 100
      - asset: assets/fonts/Roboto-Light.ttf
        weight: 300
      - asset: assets/fonts/Roboto-Regular.ttf
        weight: 400
      - asset: assets/fonts/Roboto-Medium.ttf
        weight: 500
      - asset: assets/fonts/Roboto-Bold.ttf
        weight: 700
      - asset: assets/fonts/Roboto-Black.ttf
        weight: 900
```

Font Weight Reference:

Weight	Common Name	Numeric Value
Thin	Thin/Hairline	100
Extra Light	Ultra Light	200
Light	Light	300
Regular	Normal/Book	400
Medium	Medium	500
Semi Bold	Demi Bold	600
Bold	Bold	700
Extra Bold	Ultra Bold	800
Black	Heavy/Black	900

Important Notes:

- The `family` name is what you'll use in your code
- Match weight numbers to the actual font weight
- Indentation matters in YAML - use 2 spaces
- Regular (400) weight is required for proper fallback

Step 4: Update Theme Configuration

Open `lib/core/theme/app_theme.dart` and update the `fontFamily` constant:

Current Code:

```
class AppTheme {
  ThemeData theme;
  AppTheme(this.theme);
  bool isDarkMode = false;
  static const String fontFamily = 'Geist';  // Current font
```

```
  // ... rest of the code
}
```

Update to your new font:

```
class AppTheme {
  ThemeData theme;
  AppTheme(this.theme);
  bool isDarkMode = false;
  static const String fontFamily = 'Roboto';  // Your new font name

  // ... rest of the code
}
```

Important: The `fontFamily` string must **exactly match** the `family` name you used in `pubspec.yaml`.

Step 5: Clean and Rebuild

After making changes, you need to rebuild your app:

1. **Stop the running app** (if any)
2. **Clean the build:**

   ```
   flutter clean
   ```

3. **Get dependencies:**

   ```
   flutter pub get
   ```

4. **Rebuild and run:**

   ```
   flutter run
   ```

Why Clean Build is Required:

- Font assets are compiled into the app during build
- Changes to `pubspec.yaml` require a fresh build
- Hot reload/restart won't apply font changes

Complete Example: Changing to Poppins Font

Let's walk through a complete example of changing from Geist to Poppins:

1. Download Poppins from Google Fonts

- Go to https://fonts.google.com/specimen/Poppins
- Click "Download family"
- Extract the ZIP file

2. Copy Font Files

```
# Copy the font files you need
cp ~/Downloads/Poppins/Poppins-Regular.ttf assets/fonts/
cp ~/Downloads/Poppins/Poppins-Bold.ttf assets/fonts/
cp ~/Downloads/Poppins/Poppins-Light.ttf assets/fonts/
cp ~/Downloads/Poppins/Poppins-Medium.ttf assets/fonts/
cp ~/Downloads/Poppins/Poppins-SemiBold.ttf assets/fonts/
```

3. Update pubspec.yaml

```
fonts:
  - family: Poppins
    fonts:
      - asset: assets/fonts/Poppins-Light.ttf
        weight: 300
      - asset: assets/fonts/Poppins-Regular.ttf
        weight: 400
      - asset: assets/fonts/Poppins-Medium.ttf
        weight: 500
      - asset: assets/fonts/Poppins-SemiBold.ttf
        weight: 600
```

```
      - asset: assets/fonts/Poppins-Bold.ttf
        weight: 700
```

4. Update app_theme.dart

```
static const String fontFamily = 'Poppins';
```

5. Rebuild

```
flutter clean
flutter pub get
flutter run
```

Using Multiple Font Families

You can register multiple font families in your app and use them selectively:

In pubspec.yaml:

```
fonts:
  - family: Poppins
    fonts:
      - asset: assets/fonts/Poppins-Regular.ttf
        weight: 400
      - asset: assets/fonts/Poppins-Bold.ttf
        weight: 700

  - family: RobotoMono
    fonts:
      - asset: assets/fonts/RobotoMono-Regular.ttf
        weight: 400
      - asset: assets/fonts/RobotoMono-Bold.ttf
        weight: 700
```

Using in Code:

```
// Main app font (defined in AppTheme.fontFamily)
Text('Regular text')  // Uses default Poppins

// Use specific font for code blocks
Text(
  'Code snippet',
  style: TextStyle(fontFamily: 'RobotoMono'),
)
```

Font Weight Usage in Code

Once your fonts are configured, you can use different weights:

```
// Regular (400)
Text('Regular text')

// Bold (700)
Text('Bold text', style: TextStyle(fontWeight: FontWeight.bold))

// Specific weights
Text('Light text', style: TextStyle(fontWeight: FontWeight.w300))
Text('Medium text', style: TextStyle(fontWeight: FontWeight.w500))
Text('SemiBold text', style: TextStyle(fontWeight:
FontWeight.w600))

// Available FontWeight constants
FontWeight.w100  // Thin
FontWeight.w200  // Extra Light
FontWeight.w300  // Light
FontWeight.w400  // Regular/Normal
FontWeight.w500  // Medium
FontWeight.w600  // Semi Bold
FontWeight.w700  // Bold
FontWeight.w800  // Extra Bold
FontWeight.w900  // Black
```

Troubleshooting

Problem: Font not showing after changes

Solution 1: Clean build required

```
flutter clean
flutter pub get
flutter run
```

Solution 2: Check pubspec.yaml indentation

- YAML is indent-sensitive
- Use 2 spaces for indentation
- No tabs allowed

Solution 3: Verify font family name

- Must match exactly between `pubspec.yaml` and `app_theme.dart`
- Case-sensitive (e.g., "Roboto" ≠ "roboto")

Problem: Some text still shows old font

Possible causes:

- Some widgets might have hardcoded font family
- Need to do a clean rebuild
- Cache not cleared

Solution:

```
# On iOS
flutter clean
cd ios
rm -rf Pods Podfile.lock
pod install
cd ..
flutter run

# On Android
flutter clean
cd android
```

```
./gradlew clean
cd ..
flutter run
```

Problem: Font looks different than expected

Check:

1. Correct weight files are added
2. Weight numbers match the actual font weight
3. Font files are not corrupted (re-download if needed)

Problem: App crashes after font change

Common issues:

- Missing Regular (400) weight font file
- Incorrect file path in pubspec.yaml
- Corrupted font file

Solution:

- Ensure Regular weight is always included
- Verify all paths are correct
- Re-download font files

Problem: YAML parsing error

Common mistakes:

```
# ❌ Wrong - using tabs
fonts:
        - family: Roboto

# ❌ Wrong - incorrect indentation
fonts:
- family: Roboto
  fonts:
    - asset: assets/fonts/Roboto-Regular.ttf
```

```
# ✅ Correct - 2 spaces, consistent indentation
fonts:
  - family: Roboto
    fonts:
      - asset: assets/fonts/Roboto-Regular.ttf
        weight: 400
```

Best Practices

1. Font Selection

- Choose readable fonts for body text
- Ensure good support for numbers and special characters
- Test on both iOS and Android devices
- Consider accessibility for users with dyslexia (fonts like OpenDyslexic)

2. Font Weights

- Always include Regular (400) and Bold (700) at minimum
- Add additional weights for better typography
- Don't add unnecessary weights to reduce app size

3. Font File Size

- Multiple font files increase app size
- Each weight file can be 50-200KB
- Consider app size when adding many weights
- Remove unused weights after testing

4. Testing

- Test on multiple screen sizes
- Test both light and dark themes
- Verify special characters display correctly
- Check RTL (Right-to-Left) language support if applicable

5. Performance

- Fonts are loaded at app startup
- More font files = slightly longer startup time
- Generally negligible impact for 5-10 font files

Font Licensing

Important: Always verify you have the legal right to use a font in your app.

License Types:

1. **Open Source (OFL - Open Font License)**
    - Free for commercial use
    - Example: Google Fonts, Font Squirrel
    - No attribution required (but appreciated)
2. **Commercial License**
    - Purchase required for commercial apps
    - Check license terms for mobile app usage
    - May require per-app or per-developer license
3. **Personal Use Only**
    - Cannot be used in commercial/published apps
    - Only for personal projects and learning

Checking License:

```
# Most fonts include a LICENSE.txt or OFL.txt file
ls assets/fonts/
# Look for: LICENSE.txt, OFL.txt, or README.txt
```

Recommended Free Fonts

Here are some popular, free, high-quality fonts for mobile apps:

Sans-Serif (Modern, Clean)

- **Roboto** - Google's Android system font
- **Open Sans** - Highly readable, versatile
- **Lato** - Professional, friendly
- **Poppins** - Modern, geometric
- **Montserrat** - Elegant, urban
- **Inter** - Optimized for screens
- **Nunito** - Rounded, friendly

Serif (Traditional, Formal)

- **Merriweather** - Readable, elegant
- **Lora** - Modern serif, readable
- **Playfair Display** - Elegant headlines

Monospace (Code, Technical)

- **Roboto Mono** - Clean, professional
- **Fira Code** - Developer favorite
- **JetBrains Mono** - Excellent for code

Download These Fonts:

- **Google Fonts**: https://fonts.google.com
- **Font Squirrel**: https://www.fontsquirrel.com

Advanced: Using Google Fonts Package

Instead of manually adding fonts, you can use the `google_fonts` package:

1. Add dependency to pubspec.yaml:

```
dependencies:
  google_fonts: ^6.2.0
```

2. Use in code:

```
import 'package:google_fonts/google_fonts.dart';

// In app_theme.dart
static TextStyle get textTheme => GoogleFonts.robotoTextTheme();

// Or in individual widgets
Text(
  'Hello World',
  style: GoogleFonts.poppins(
    fontSize: 16,
    fontWeight: FontWeight.bold,
  ),
)
```

Pros:

- No need to download font files
- Easy to switch fonts
- Automatic caching

Cons:

- Requires internet on first load
- Slightly larger app size
- Less control over exact weights

Additional Resources

Font Resources:

- **Google Fonts**: https://fonts.google.com (Free, open source)

- **Font Squirrel**: https://www.fontsquirrel.com (Free, commercial use)
- **DaFont**: https://www.dafont.com (Free and paid)
- **Adobe Fonts**: https://fonts.adobe.com (Requires subscription)
- **1001 Fonts**: https://www.1001fonts.com

Font Tools:

- **Font Pair**: https://fontpair.co (Font combination suggestions)
- **Type Scale**: https://typescale.com (Typography scale generator)
- **FontDrop**: https://fontdrop.info (Online font file inspector)

Flutter Documentation:

- **Using Custom Fonts**: https://docs.flutter.dev/cookbook/design/fonts
- **Google Fonts Package**: https://pub.dev/packages/google_fonts
- **Material Design Typography**: https://material.io/design/typography

Testing Fonts:

- **Fonts Arena**: https://fontsarena.com (Preview fonts before downloading)
- **Font Flipper**: https://fontflipper.com (Quick font comparison)

Quick Reference Checklist

Use this checklist when changing fonts:

- [ ] Downloaded font files (.ttf or .otf)
- [ ] Verified font license allows commercial use
- [ ] Copied font files to `assets/fonts/` directory
- [ ] Updated `pubspec.yaml` with font family and weights
- [ ] Updated `fontFamily` in `lib/core/theme/app_theme.dart`
- [ ] Verified family name matches exactly (case-sensitive)
- [ ] Included at least Regular (400) weight
- [ ] Ran `flutter clean`
- [ ] Ran `flutter pub get`

- [ ] Rebuilt and ran the app
- [ ] Tested on both light and dark themes
- [ ] Verified text is readable on different screens
- [ ] Checked all font weights display correctly

Note: Font changes require a complete rebuild (not just hot reload) to take effect. Always test thoroughly on actual devices after changing fonts, as fonts may render differently than on simulators/emulators.

Deep links

This guide explains how to configure deep linking for the ELMS (E-Learning Management System) Flutter application on iOS and Android platforms.

Prerequisites

- Basic understanding of iOS and Android configuration

Native Platform Configuration

Step 1: Choose Your URL Scheme

First, decide on your custom URL scheme. The default is `elms`, but you should change it to something unique for your app.

Naming Guidelines:

- Use lowercase letters only
- Keep it short and memorable
- Make it unique to avoid conflicts
- Examples: `mylearning`, `eduapp`, `learnhub`

Step 2: Configure Android

2.1 Open Android Manifest

Navigate to:

```
android/app/src/main/AndroidManifest.xml
```

2.2 Update Custom Scheme

Find this section and replace `elms` with your chosen scheme:

```
<!-- Deep Link Intent Filter -->
<intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>

    <!-- Custom scheme: yourscheme://... -->
    <data android:scheme="elms"/>  <!-- Change 'elms' to your
scheme -->
</intent-filter>
```

Example: If your scheme is `mylearning`:

```
<data android:scheme="mylearning"/>
```

Complete Example (AndroidManifest.xml):

```
<activity
    android:name=".MainActivity"
    android:exported="true"
    android:launchMode="singleTop"
    android:theme="@style/LaunchTheme"

android:configChanges="orientation|keyboardHidden|keyboard|screenSize
    android:hardwareAccelerated="true"
    android:windowSoftInputMode="adjustResize">

    <!-- Standard launcher intent -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>

    <!-- Custom scheme deep links -->
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="mylearning"/>
    </intent-filter>
</activity>
```

Step 3: Configure iOS

3.1 Open Info.plist

Navigate to:

```
ios/Runner/Info.plist
```

3.2 Update Custom Scheme

Find the `CFBundleURLTypes` section and update your custom scheme:

```
<key>CFBundleURLTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
        <string>Editor</string>
        <key>CFBundleURLSchemes</key>
        <array>
            <!-- Your custom scheme here -->
            <string>elms</string>  <!-- Change to your scheme -->
            <!-- Other schemes (Google Sign-In, etc.) -->

<string>com.googleusercontent.apps.YOUR_CLIENT_ID</string>
        </array>
    </dict>
</array>
```

Example: If your scheme is `mylearning`:

```
<array>
    <string>mylearning</string>
    <string>com.googleusercontent.apps.YOUR_CLIENT_ID</string>
</array>
```

3.3 Enable Flutter Deep Linking

Ensure this key is present and set to `true`:

```
<key>FlutterDeepLinkingEnabled</key>
<true/>
```

Step 4: Testing Your Configuration

4.1 Testing Custom Scheme Links

On Android (ADB):

```
adb shell am start -W -a android.intent.action.VIEW -d
"elms://course-details/123"
```

On iOS (Simulator):

```
xcrun simctl openurl booted "elms://course-details/123"
```

4.2 Testing on Real Devices

- Send the deep link via Messages, WhatsApp, or email
- Tap the link to verify the app opens correctly
- Test from Safari (iOS) or Chrome (Android) browser

Troubleshooting

Issue: Deep links not working on Android

Solutions:

1. **Verify intent filter is correct:**
    - Check `AndroidManifest.xml` for typos
    - Verify scheme matches exactly
2. **Clear app data and reinstall:**

```
```

```
adb shell pm clear com.yourcompany.yourapp
adb uninstall com.yourcompany.yourapp
flutter run
```

3. **Check ADB logs:**

```
adb logcat | grep -i "intent"
```

Issue: Deep links not working on iOS

Solutions:

1. **Verify Info.plist configuration:**
   - Check scheme is in `CFBundleURLSchemes`
   - Ensure `FlutterDeepLinkingEnabled` is `true`
2. **Test with different methods:**
   - Long-press link in Notes app
   - Use Safari instead of in-app browser
   - Try from Messages app
3. **Check iOS logs:**
   - Open Console app on Mac
   - Filter by your device
   - Look for deep link related errors

Issue: App opens but doesn't navigate to correct screen

Solutions:

1. **Check Deep Link Manager implementation:**
   - Verify route name matches in switch statement
   - Check if screen route exists in `AppRoutes`
   - Ensure navigation context is available

2. **Add logging:**

```
void _handleDeepLink(String link) {
  print('Handling deep link: $link');  // Add this
  try {
    // ... rest of code
  } catch (e) {
    print('Error handling deep link: $e');  // Add this
    return;
  }
}
```

3. **Verify app initialization:**
   - Ensure `DeepLinkManager.instance.initialize()` is called
   - Check it's called after navigation context is available

Additional Resources

Documentation:

- **app_links package**: https://pub.dev/packages/app_links
- **Android Deep Links**: https://developer.android.com/training/app-links/deep-linking
- **iOS URL Schemes**: https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app

Testing Tools:

- **Deep Link Tester (Android)**: https://play.google.com/store/apps/details?id=com.tetracode.deeplinktester

Debugging Tools:

- **ADB (Android Debug Bridge)**: For testing Android deep links
- **Xcode Console**: For debugging iOS deep links

Configuration Checklist

Android Native Configuration

- [ ] Updated `AndroidManifest.xml` with custom scheme
- [ ] Verified scheme matches exactly (no typos)
- [ ] Tested deep links with ADB commands
- [ ] Tested on real devices

iOS Native Configuration

- [ ] Updated `Info.plist` with custom scheme in `CFBundleURLSchemes`
- [ ] Set `FlutterDeepLinkingEnabled` to `true` in `Info.plist`
- [ ] Tested deep links with Simulator
- [ ] Tested deep links on real device

Note: Deep linking configuration requires careful setup on both native platforms. Test thoroughly on both iOS and Android before releasing to production.

Run App

Follow these steps to run the ELMS Flutter application on your device or emulator.

Prerequisites

Before running the app, ensure you have:

- Flutter SDK installed (check with `flutter --version`)
- Android Studio or Xcode installed (for Android or iOS development)
- An emulator/simulator running or a physical device connected

Steps to Run

1. Navigate to Project Directory

Open your terminal and navigate to the project root:

```
cd /path/to/elms
```

2. Install Dependencies

If you haven't already, get all the required packages:

```
flutter pub get
```

3. Check Available Devices

List all available devices (emulators, simulators, or connected physical devices):

```
flutter devices
```

4. Run the App

Option A: Run on Default Device

```
flutter run
```

Option B: Run on Specific Device

```
flutter run -d <device-id>
```

For example:

- iOS Simulator: `flutter run -d iphone`
- Android Emulator: `flutter run -d emulator-5554`
- Chrome (web): `flutter run -d chrome`

Option C: Run with Specific Flavor (if configured)

```
flutter run --flavor development
flutter run --flavor production
```

5. Development Mode Features

Once the app is running, you can use these hot reload features:

- Press `r` to hot reload
- Press `R` to hot restart
- Press `h` to display help
- Press `q` to quit

Common Issues & Solutions

Issue: "No devices found"

Solution: Start an emulator or connect a physical device, then run `flutter devices` to verify.

Issue: Build errors or dependency conflicts

Solution: Clean and rebuild:

```
flutter clean
flutter pub get
flutter run
```

Issue: iOS build fails

Solution: Update CocoaPods dependencies:

```
cd ios
pod install
cd ..
flutter run
```

Issue: Android build fails

Solution:

- Ensure Android SDK is properly configured
- Check `android/local.properties` has correct SDK path
- Run `flutter doctor` to diagnose issues

Quick Command Reference

Command	Description
`flutter run`	Run app on default device
`flutter run --release`	Run in release mode

Command	Description
`flutter run --profile`	Run in profile mode (for performance testing)
`flutter devices`	List available devices
`flutter clean`	Clean build artifacts
`flutter pub get`	Get dependencies
`flutter doctor`	Check Flutter environment setup

Additional Notes

- For detailed troubleshooting and project-specific commands, refer to the README.md file
- The app supports both light and dark themes
- All user-facing strings are localized (see `assets/languages/`)
- API endpoints are configured in `lib/core/api/`

For more information about the project architecture and development guidelines, see CLAUDE.md.

Generate Release Version

Android Release

Create Key Store File

1. Generate a keystore file using the following command:

   ```
   keytool -genkey -v -keystore your-keystore-file.jks -keyalg RSA
   -keysize 2048 -validity 10000 -alias your-alias
   ```

   - Replace "your-keystore-file.jks" with your preferred filename (keep .jks extension)
   - Replace "your-alias" with your preferred alias
   - Set a password when prompted (characters won't be visible)
   - Press Enter to skip optional fields

2. Create `key.properties` in the `android` folder with:

   ```
   storePassword=[your-password-from-previous-step]
   keyPassword=[your-password-from-previous-step]
   keyAlias=[your-alias-from-previous-step]
   storeFile=[your-keystore-file-location]
   ```

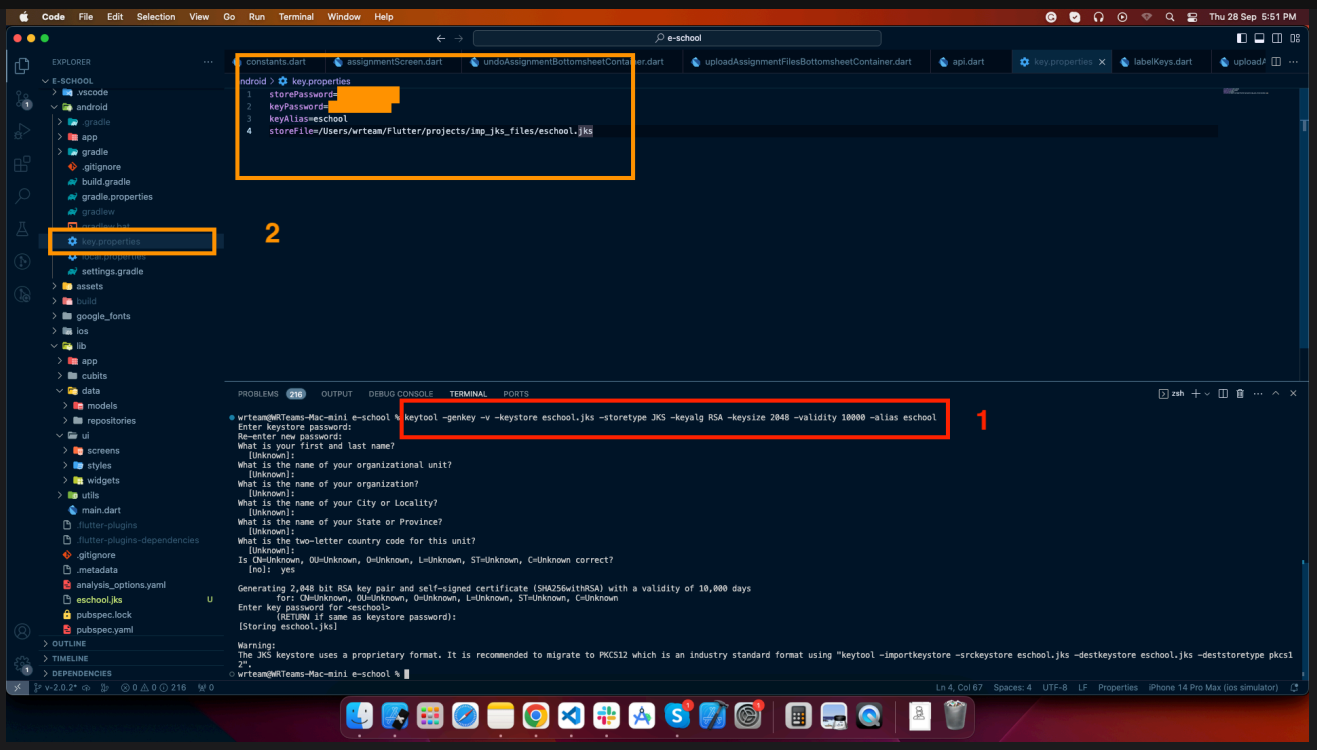
Generate Release Build

For APK:

```
flutter build apk
```

For App Bundle (Play Store):

```
flutter build appbundle
```



iOS Release

Follow the official Flutter documentation for iOS deployment: Flutter iOS Deployment Guide

Additional Resources

For more detailed information:

- Android deployment: Flutter Android Deployment
- iOS deployment: Flutter iOS Deployment

Admin Panel

Thank you for purchasing **eLMS**. We appreciate your support. In this documentation, you will find instructions on how to set up the admin panel for eLMS. This document also provides features of eLMS.

Contact Information

By: WRTeam
Email: wrteam.vijya@gmail.com

About This Documentation

This documentation provides comprehensive instructions for setting up and configuring the eLMS admin panel. You'll find detailed guides for:

- Setting up the admin panel
- Configuring system settings
- Managing notifications
- Setting up payment gateways
- And much more

If you have any questions, feel free to reach out. Thank you for choosing eLMS!

Prerequisite

eLMS Admin panel is built using **Laravel Framework 12** so you need **PHP version upto 8.3** version installed on your server.

Upload Admin Panel code on server

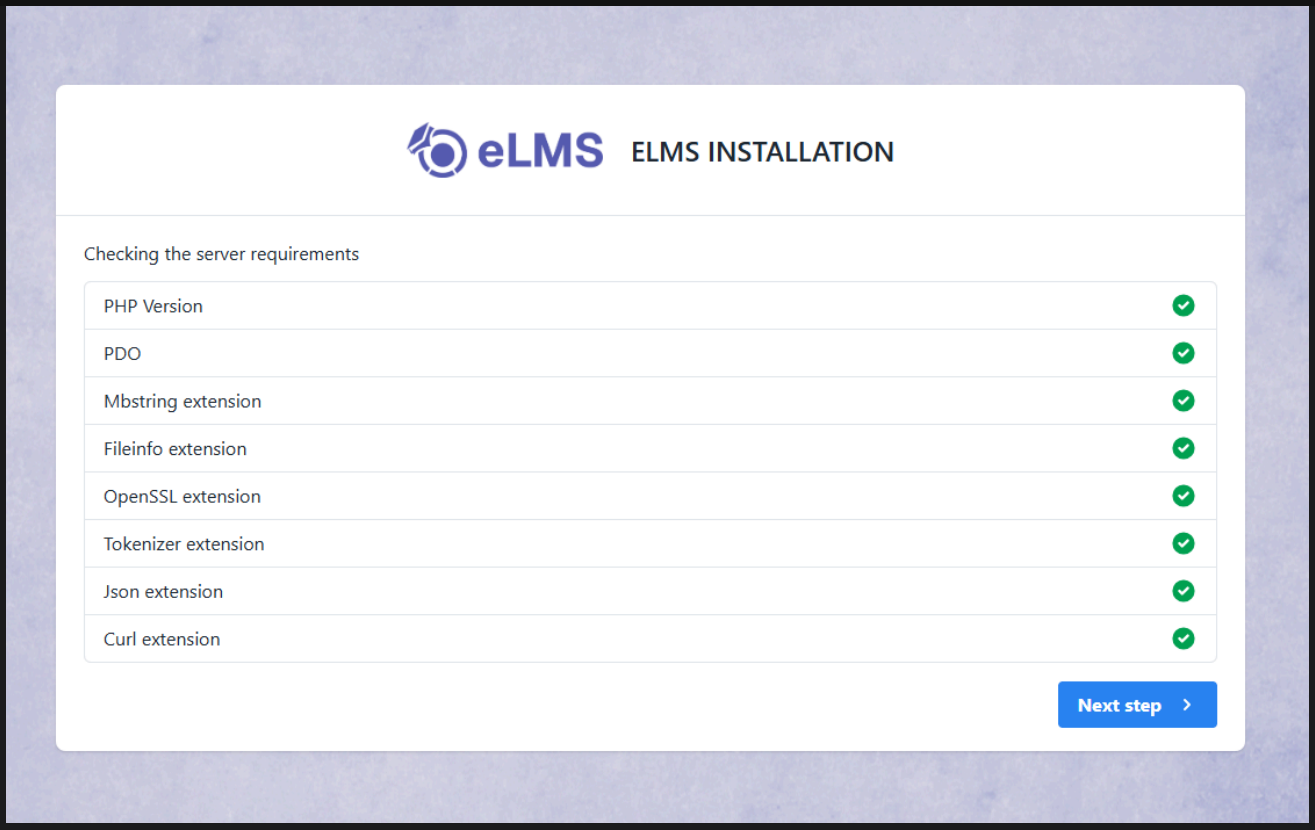
1. Unzip the code you will see the **Admin Panel** inside the zip.
2. Upload this folder content on your server in **public_html** folder.
3. After uploading visit your domain url. You will see the installation wizard. Follow the instructions.

Installation Steps

1. Installation Screen

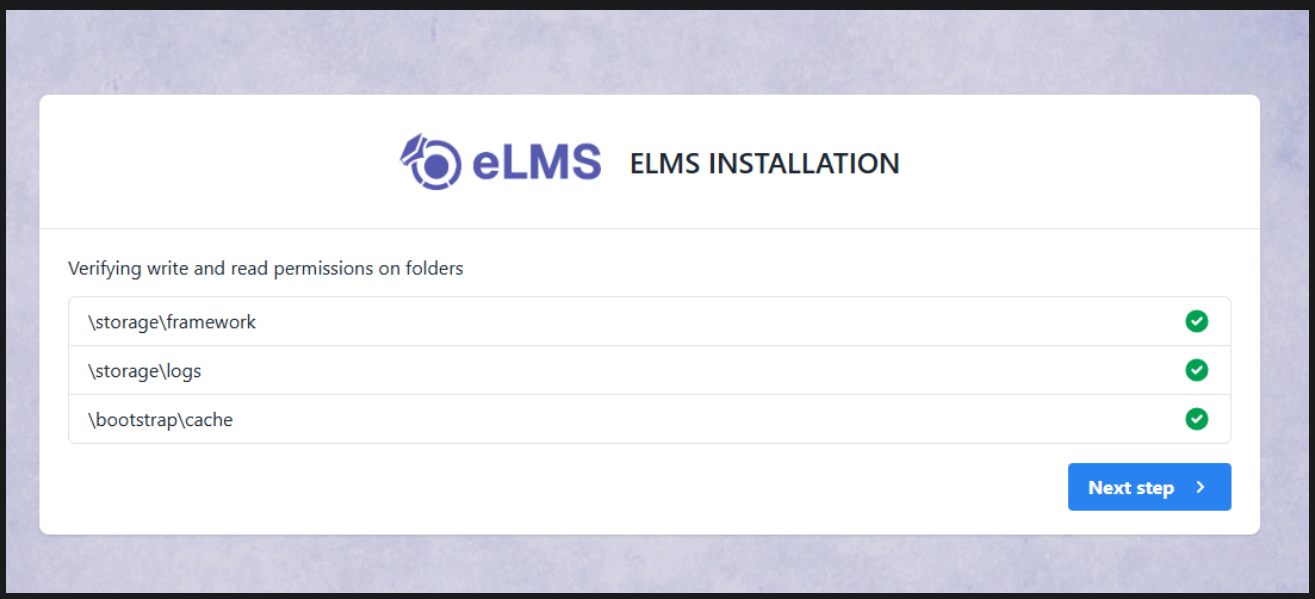
2. Server Requirements & Extensions

If PHP version is lower than 8.3 or any extension is not installed then it will be highlighted using Red color. So you need to make sure that your server meets all requirements.



3. Permission Screen

Make sure this folder have read & write permissions. If not then assign this folders read & write permissions.



4. Purchase Code Validation

Here you'll have to insert the purchase code obtained from CodeCanyon to authenticate your purchase and authorize your server to access the services.

To find your Purchase code you can visit this link: Where is My Purchase code?

5. Database Configuration Screen

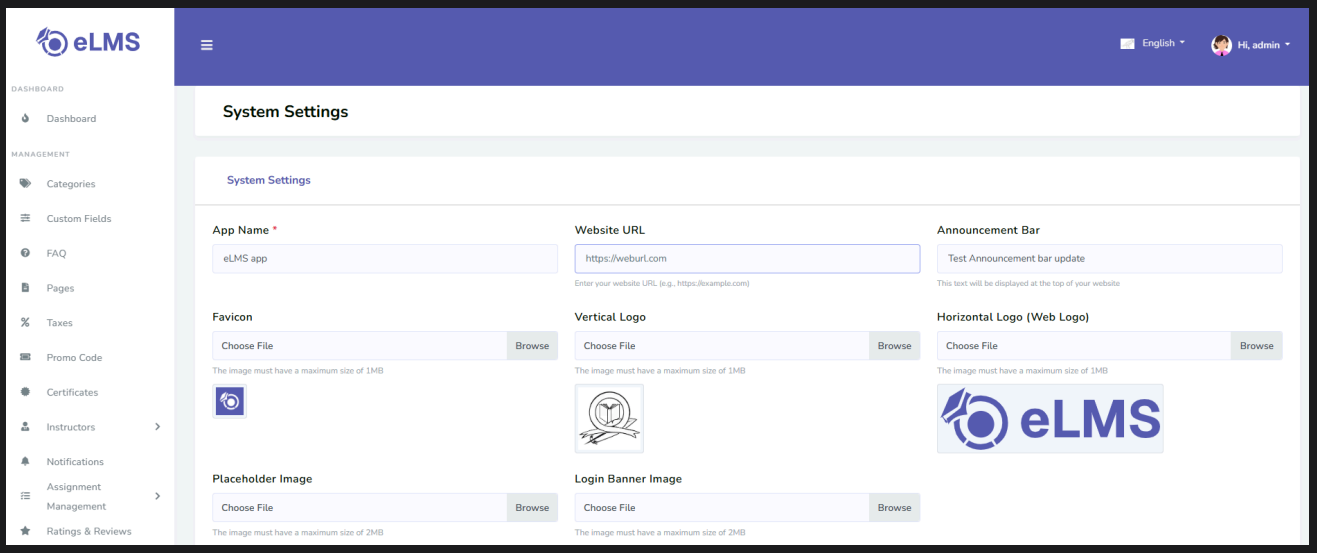
6. Database Connection Status

7. Final Installation Screen

If everything is configured successfully then your Admin panel has been installed successfully. You can login as Super Admin using the credentials provided here.

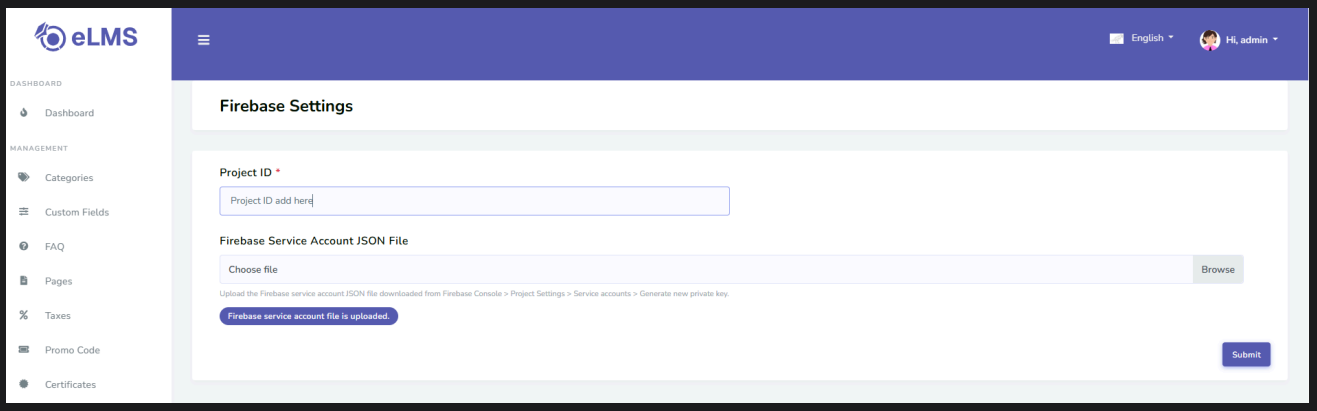
Syetem Settings

Set your System Settings here.

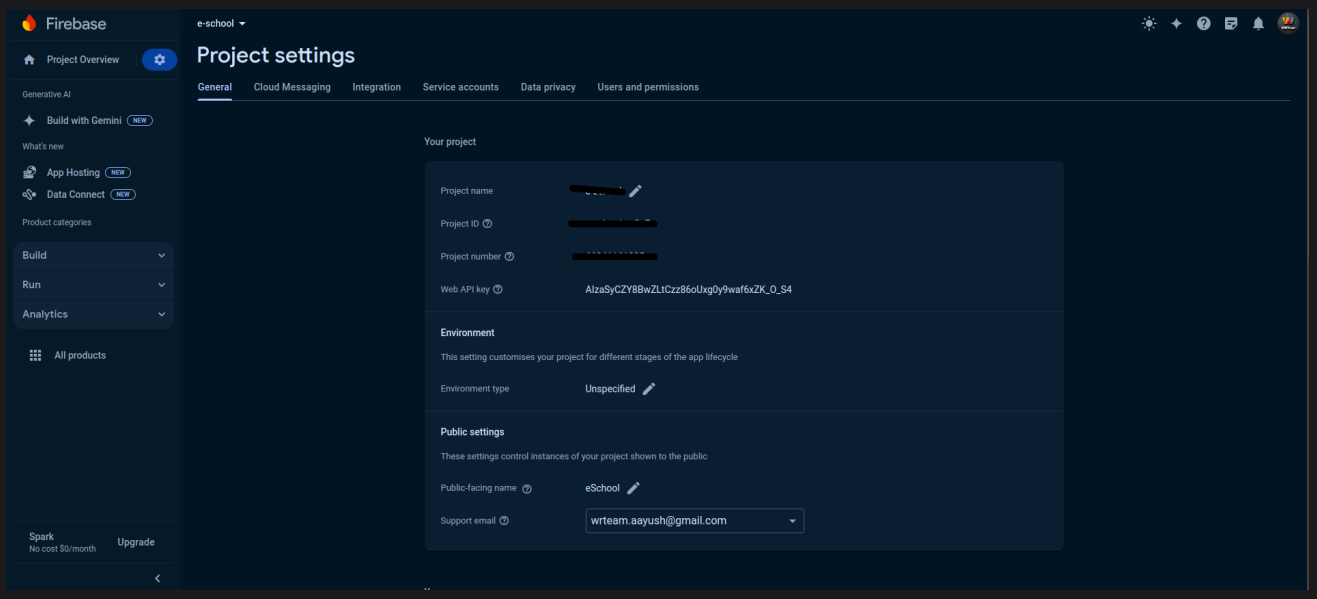


Firebase Settings

These settings will be used by your Flutter APP / Next JS web.

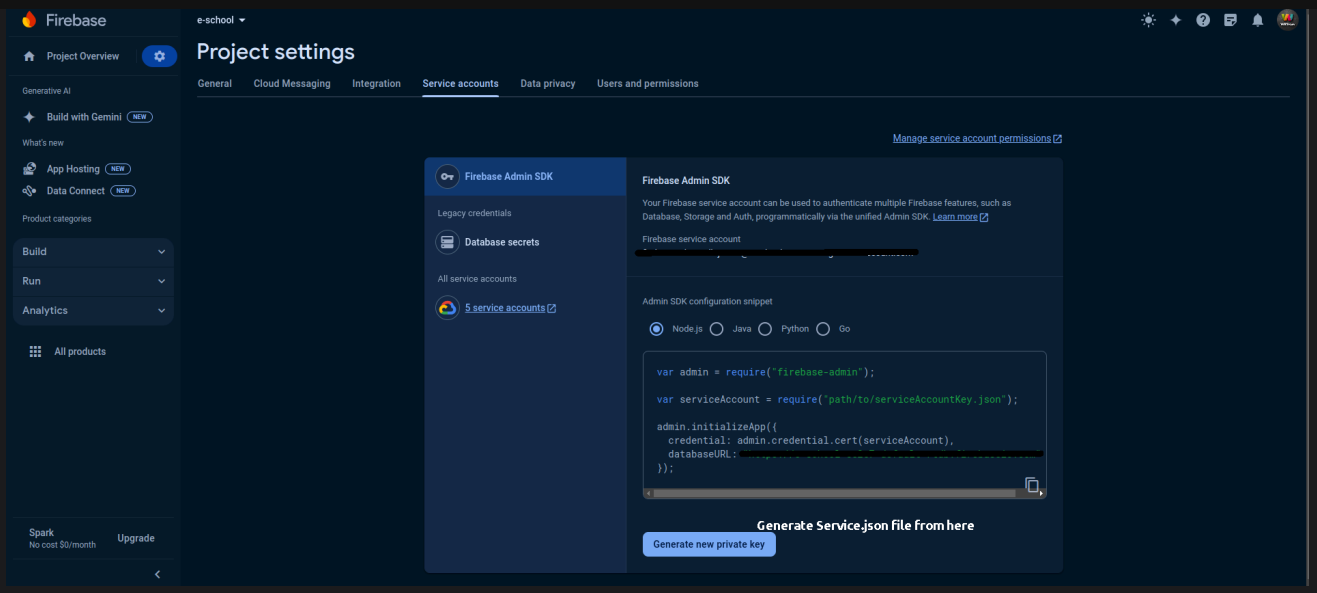


Firebase Project Id



1. In the Firebase console, go to Project Settings
2. Go to > General where Project Id is given add that to Admin Panel in Settings > Firebase Setting.

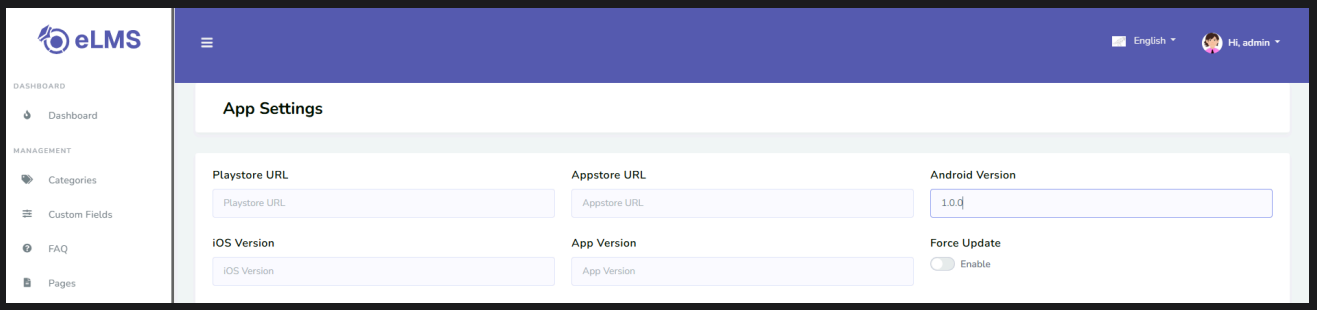
Service Json File



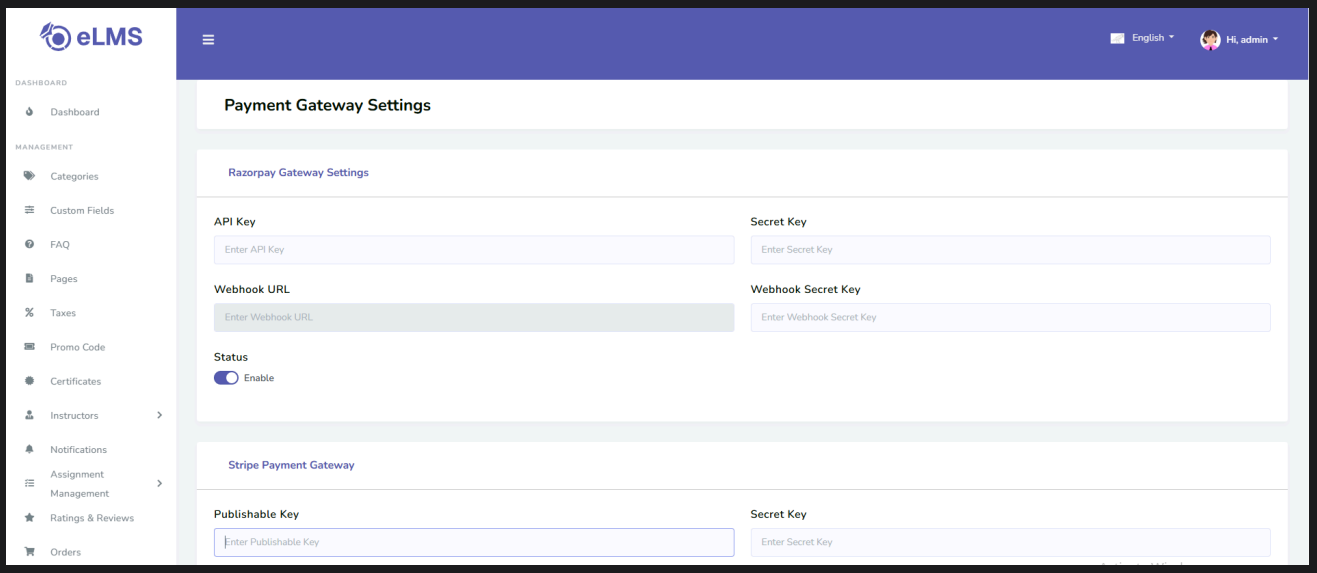
1. In the Firebase console, open Settings > Service Accounts.
2. Click Generate New Private Key, then confirm by clicking Generate Key.
3. Securely store the JSON file containing the key and upload in Admin Panel in Settings > Firebase Setting.

App Settings

These settings will be used by your Flutter APP.

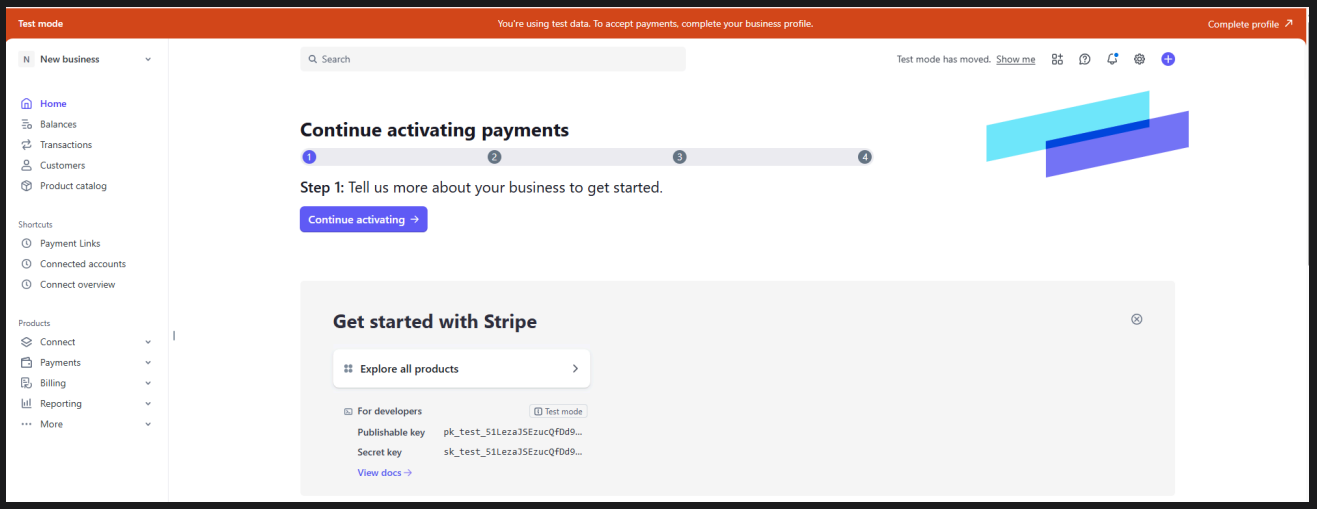


Payment Gateway Settings



Stripe

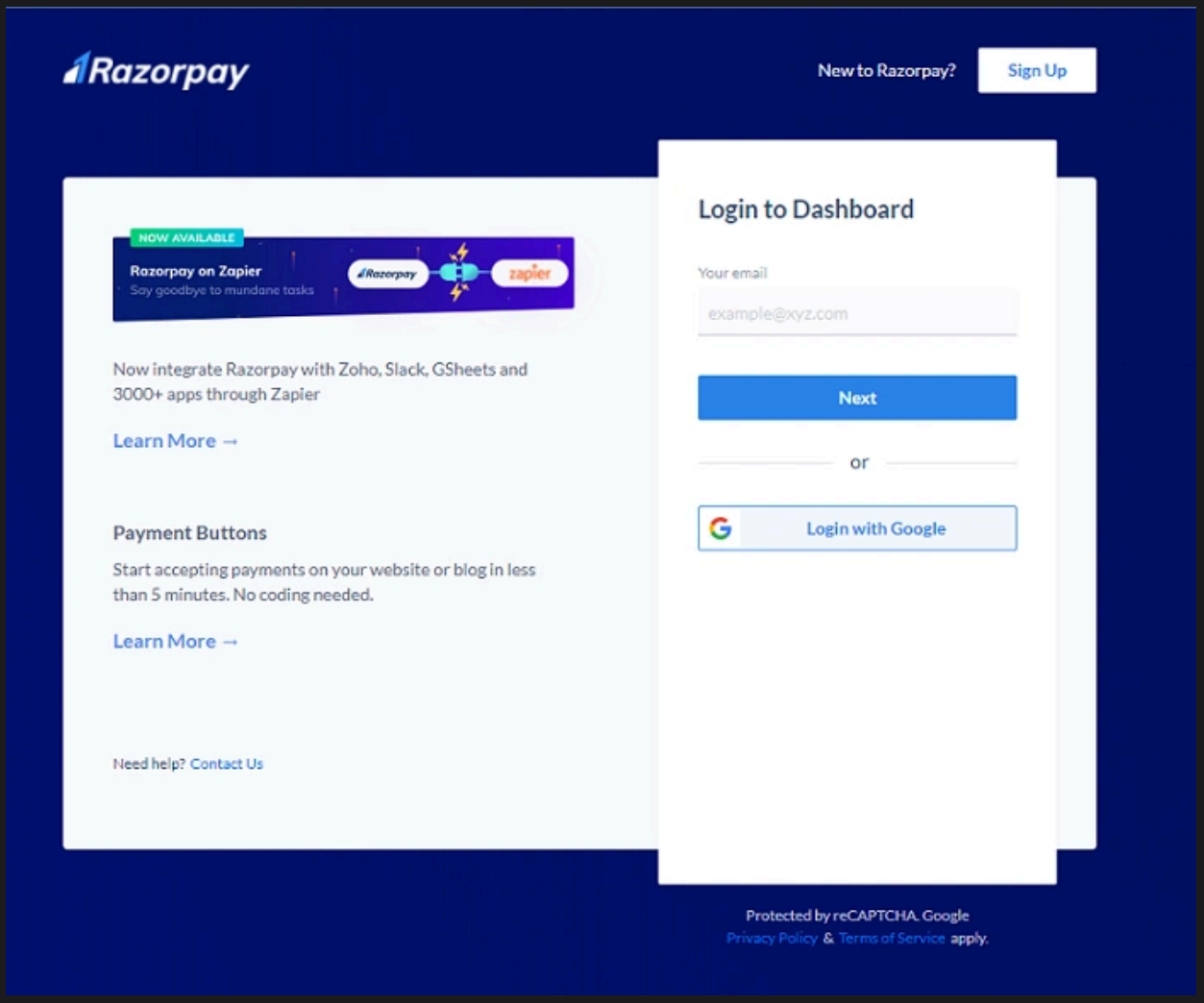
1. Log in to the Stripe dashboard — https://dashboard.stripe.com/
2. You will find your Publishable key and Secret key in the dashboard.



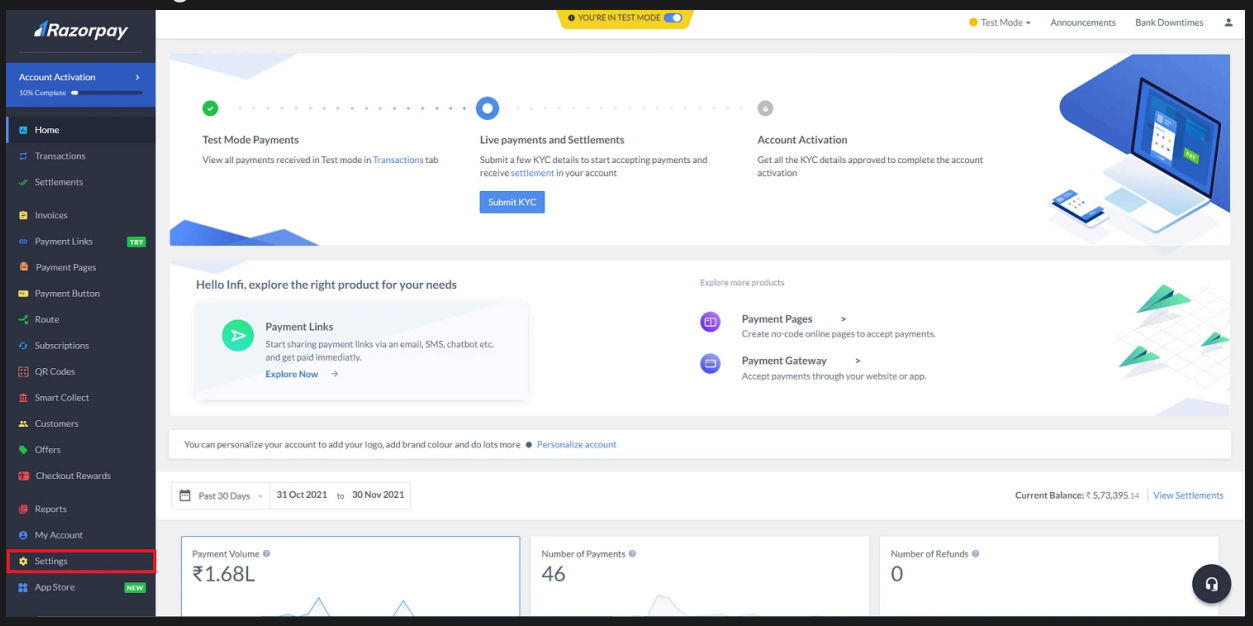
3. Docs - https://docs.stripe.com/payments?payments=popular

Razorpay

1. Go to https://dashboard.razorpay.com/ and sign in with your Razorpay account



2. Click Settings



3. Click Api keys

4. Genetare new key or regenerate key and copy it.

Flutterwave

1. Log in to the Flutterwave dashboard — https://app.flutterwave.com/login
2. In Settings → Developers → API Keys, you can find your keys.

3. Docs - https://developer.flutterwave.com/docs/getting-started?_gl=1*1xkvvzx*_gcl_au*MTQxNjg3NzkzOC4xNzY0MjU2Nzg1*_ga*MTg3MTEwMDMzNy4xNzY0MjMwNzI5*_ga_KQ9NSEMFCF*czE3NjQyNTY3NDgkbzEkZzEkdDE3NjQyNTY3ODUkajIzJGwwJGgw

Introduction

This guide will walk you through deploying the Web application.

Prerequisites

> ⚠ **WARNING**
>
> If you possess a VPS server along with some familiarity with Node.js, npm, and pm2, you're well-equipped to deploy.

Required Resources

> ⚠ **MANDATORY**
>
> 1. **VPS Hosting**: A Virtual Private Server (VPS) is mandatory to ensure reliable performance and security. Shared hosting environments are not supported for this deployment method.
> 2. **Node.js Support**: The server must support Node.js , as it is essential for running the application with seo.
> 3. **Memory Requirements**: The server should have at least 3-4 GB of free RAM to handle the application's processes effectively.
> 4. **SSH Root Access**: The server must provide SSH root access to execute Node.js commands and manage the application.

Server Setup

> ⚠ **WARNING**
>
> (For SEO) Deployment of Next.js requires knowledge of `node.js`, `npm`, and `pm2` technologies. We assume you are using a `debian`-based OS with apt as your

package manager. If you are using another Linux distribution, replace apt with your system's package manager.

How to setup Web Version

Install Node.js

Visit Node.js official website: https://nodejs.dev/en/learn/how-to-install-nodejs/ for the full installation guide.

Run the project

1. Unzip the downloaded code. After unzipping you will have News - Web Code Folder. Open it in Visual Studio Code.

2. Open VS Code terminal by typing **CTRL+J** in Windows/Linux, and for macOS **CMD+J** and execute the command --> **npm install**
   This will take some time to download packages so wait for a few minutes.

3. After **npm install** finishes, run this command --> **npm run dev**
   This command will start the development mode. Check if everything is working fine before proceeding further.

> ⚠ **MANDATORY**
>
> The Web domain must be SSL for security reasons

> ⓘ **IMPORTANT**
>
> News Web Version is built with Next.js v14.0.3 so you need to setup your Local Computer first.

Configuration Options

For detailed configuration options, see the following sections:

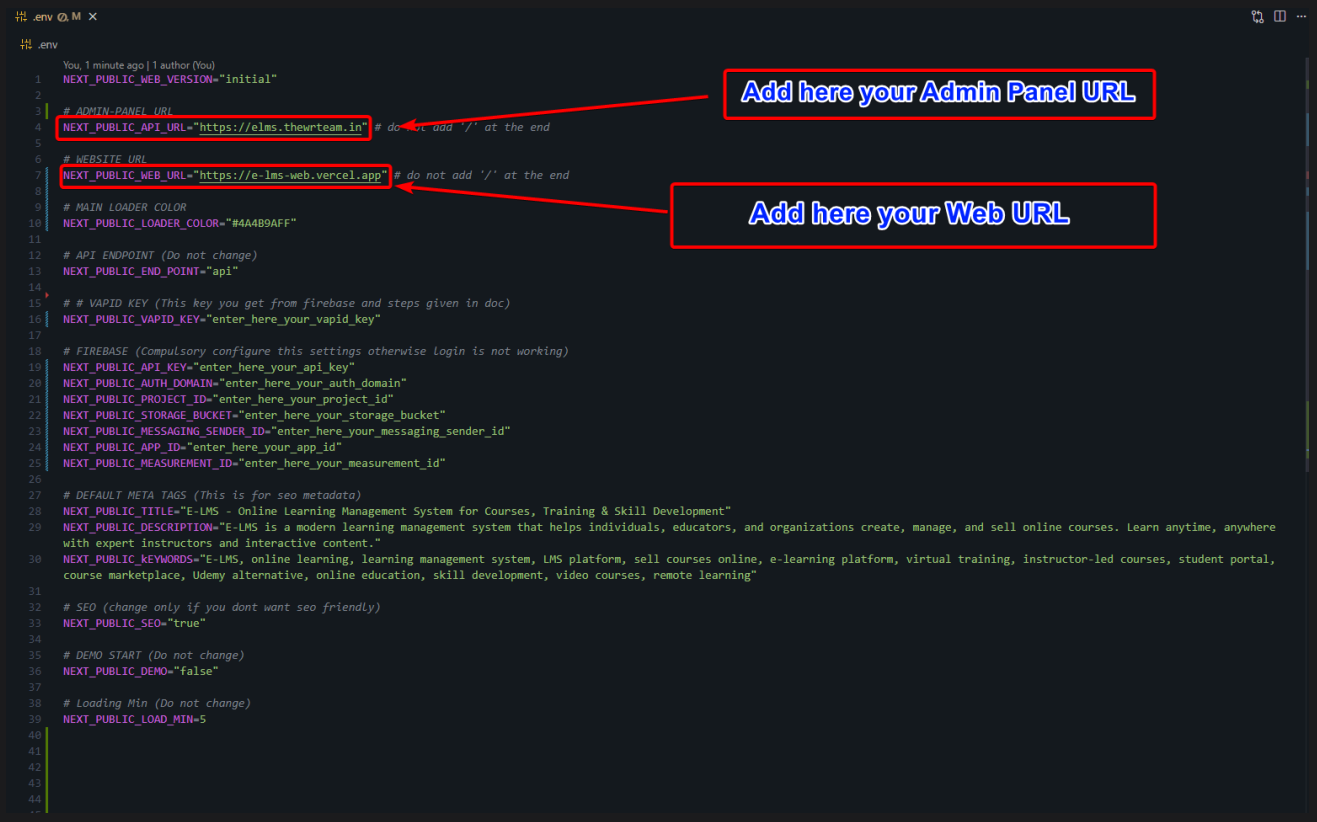
- Setting API URL

- Integrating Firebase

Where to Set API URL (Admin) and Web URL

API URL is your admin panel URL which is used to fetch the data from admin panel.

Open `.env` file and change the following details:

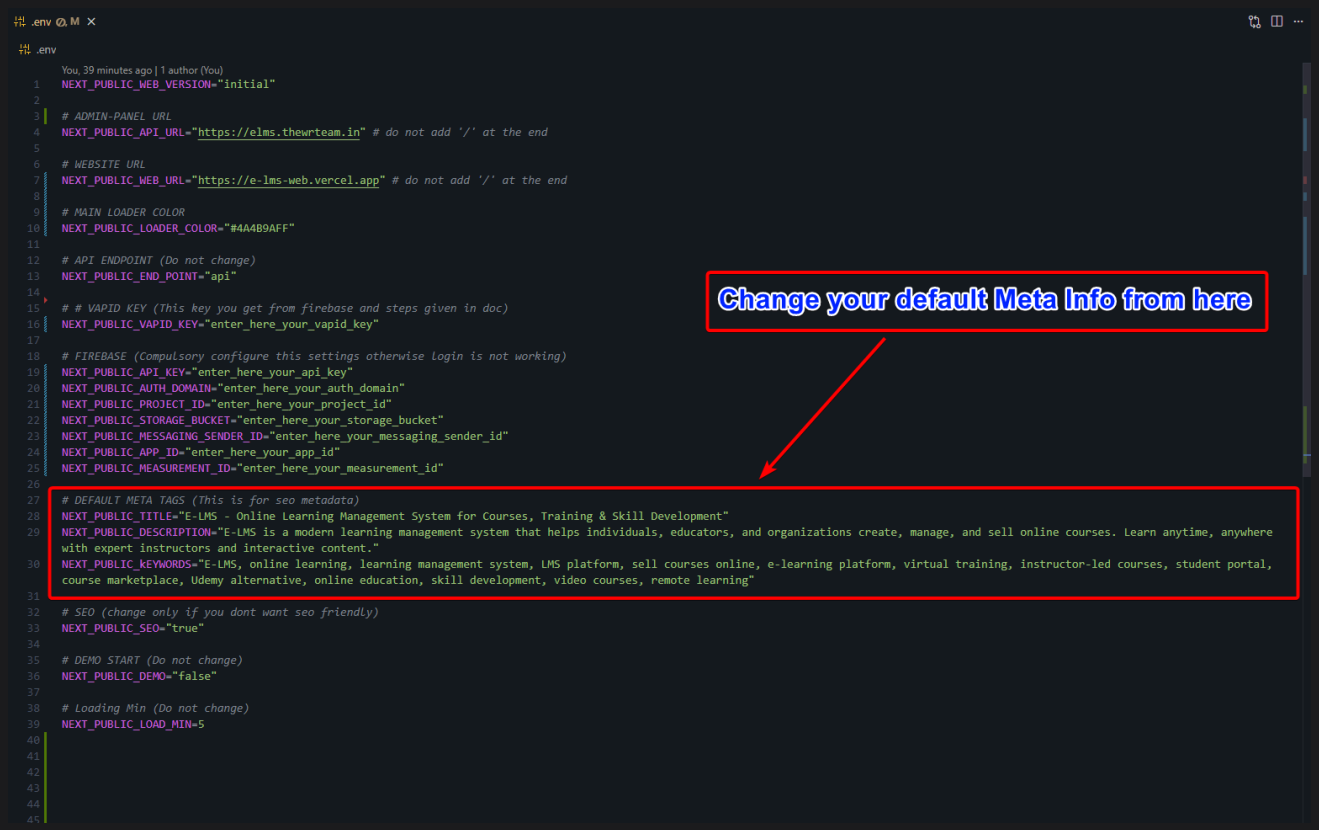


How to change Meta Information

To change the meta information for your Web application:

1. Go to `.env` file

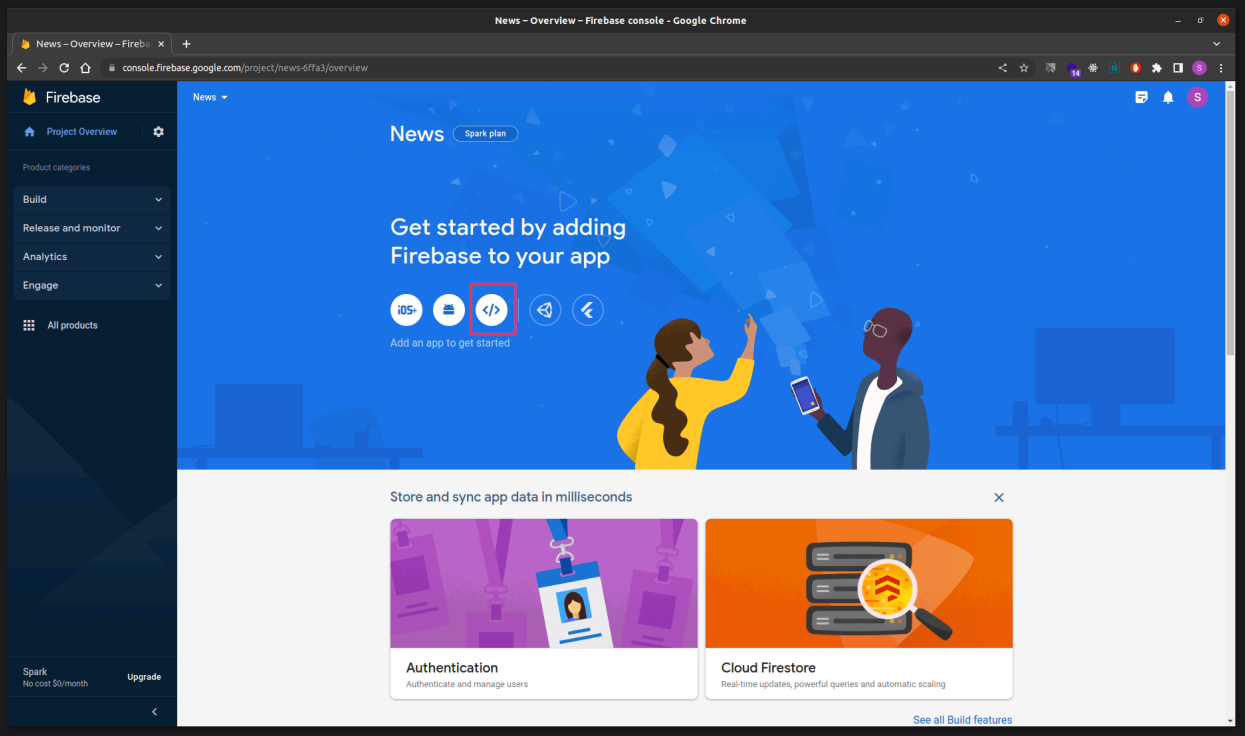
2. Update the NEXT_PUBLIC_TITLE,NEXT_PUBLIC_DESCRIPTION and NEXT_PUBLIC_kEYWORDS in the environment variables



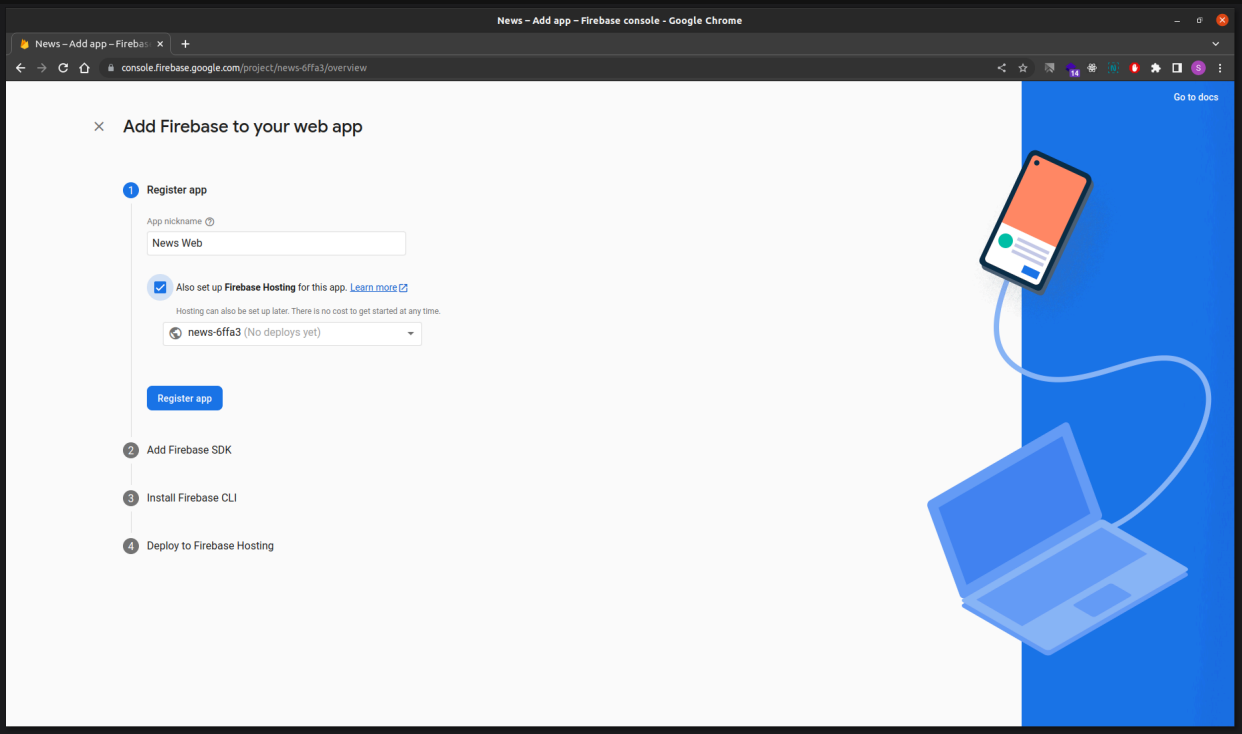
The changes will be reflected throughout your web application.

How to Connect Firebase with your Web

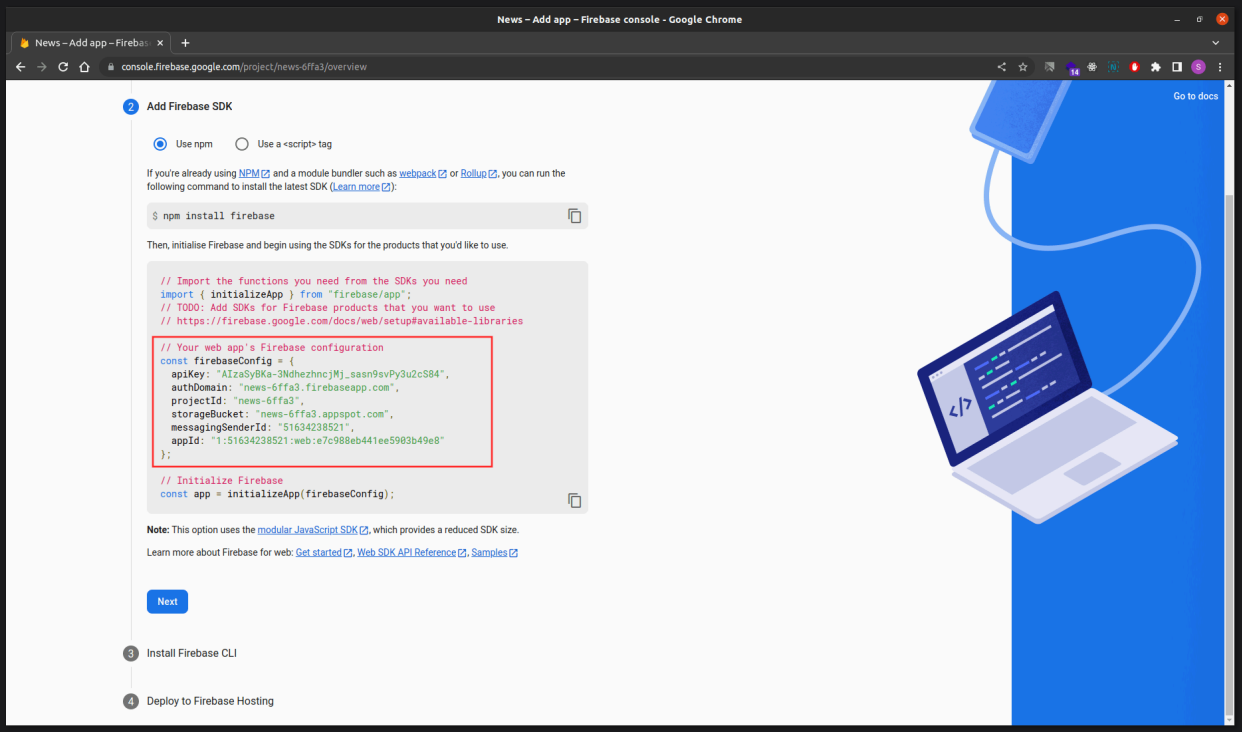
1. First you need to create a Web project inside your Firebase Project.
   For that open your firebase project in console and click on this tiny web icon:



2. After clicking on Web app you will be able to see this Screen.
   Add your web app name and also make sure you don't forget to check the checkbox.



3. Now Copy this Firebase Credentials.



4. And paste this your Credentials in **.env** and **public/firebase-messaging-sw.js** File.

5. Open firebase account go to project settings -> cloud messaging -> Web configuration and select key and paste in .env file

6. **Note: Skip this Step if you have already copy pasted the credentials**
   If you have forget to copy your firebase credentials then you can always find your credentials by following these steps:

7. For Login Provider open authentication -> sign in method

8. **Now you have to add your Web domain to your Firebase Project**

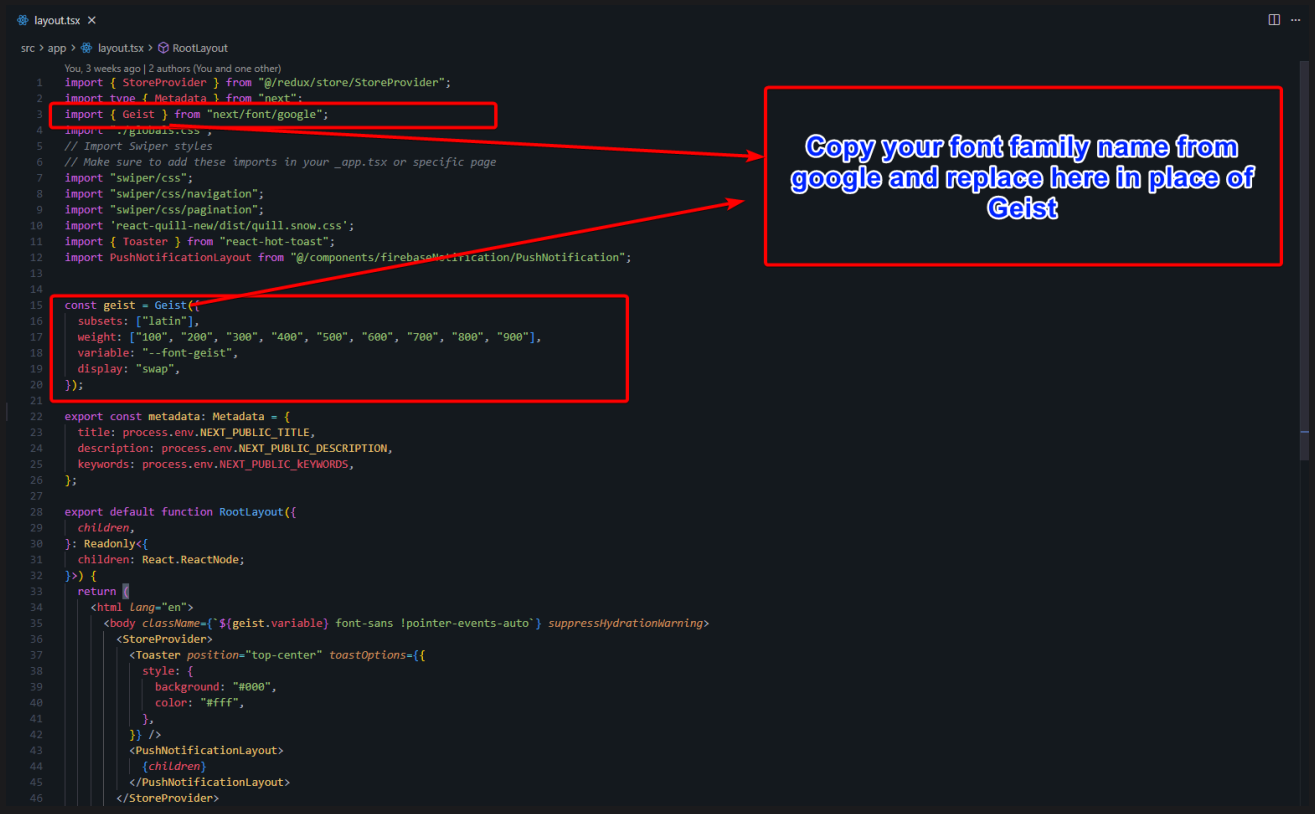
    i. Open your Firebase Project

ii. Go you Authentication/settings/Authorized Domain 3. Click on Add Domain 4. One Popup will open add your domain name without http/https in that popup and click submit.

Congratulations. You have successfully connected your Web application to your firebase project. Now you are good to go ahead.

How to Change Fonts

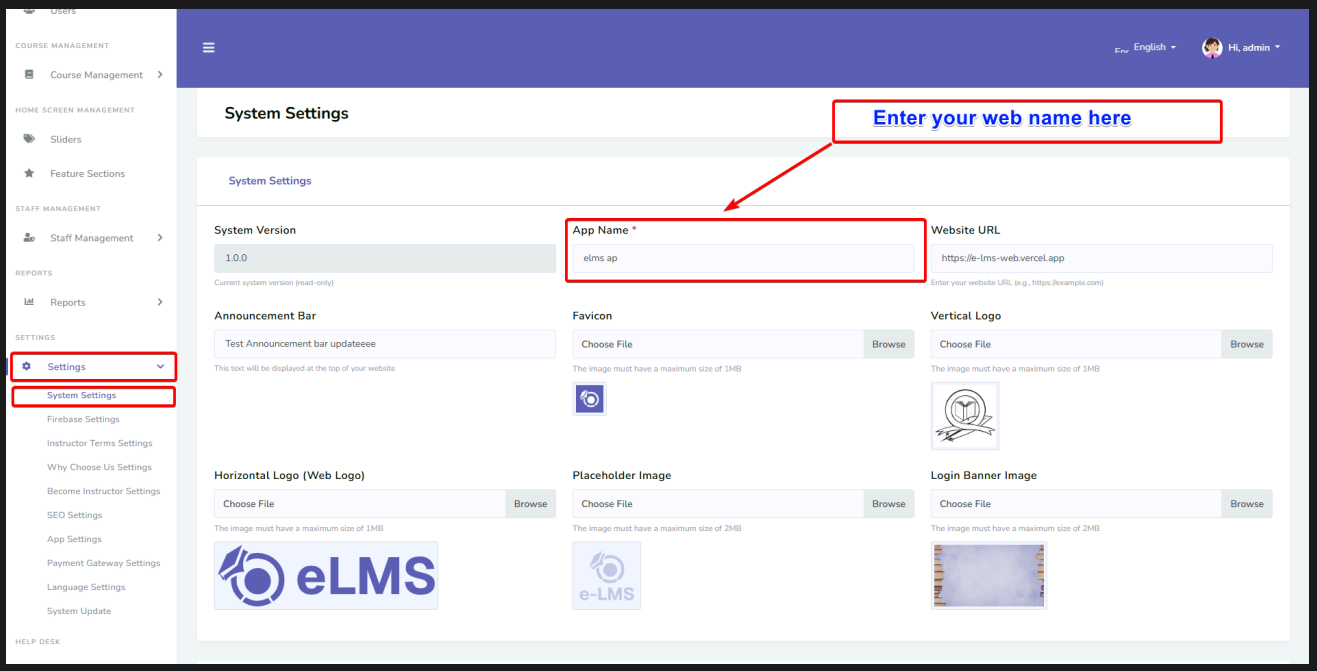
1. Open google font and select the font then copy the font-family name.
2. Open `src\app\layout.tsx` and replace the font-family name in this file with yours as shown in below image.



How to change Web Application name

To change the application name for your Web application:

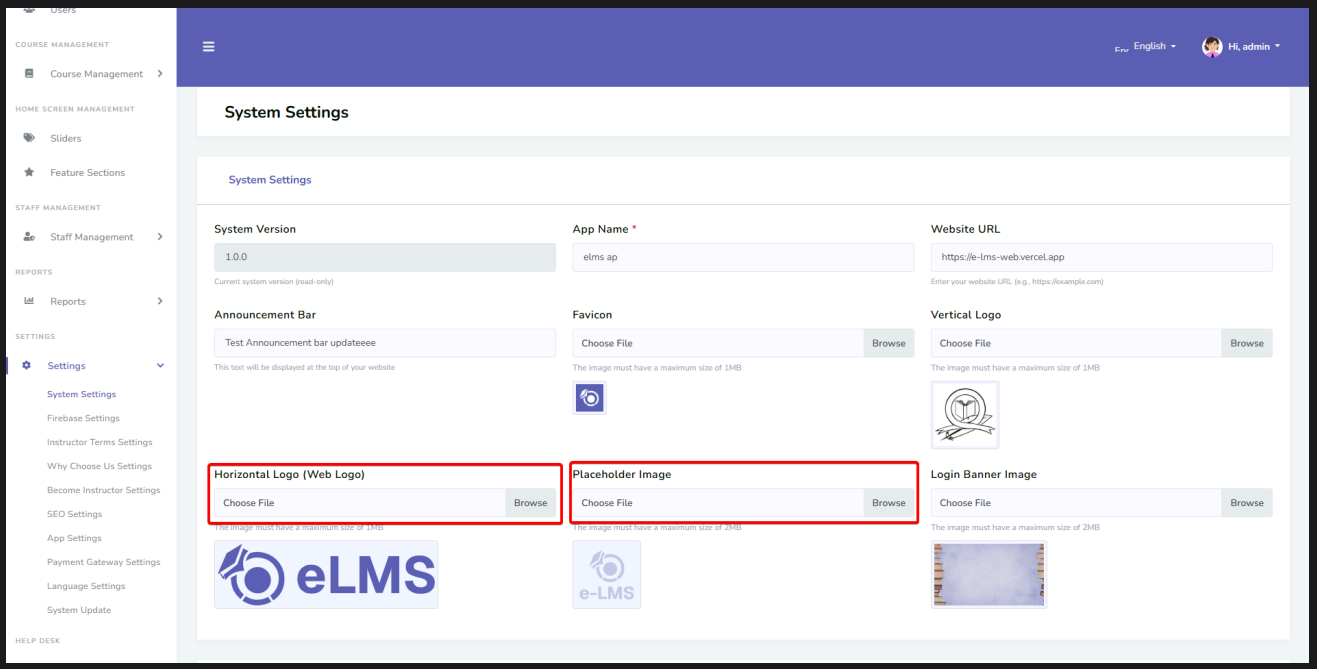
1. Go to admin panel **settings -> system-settings**

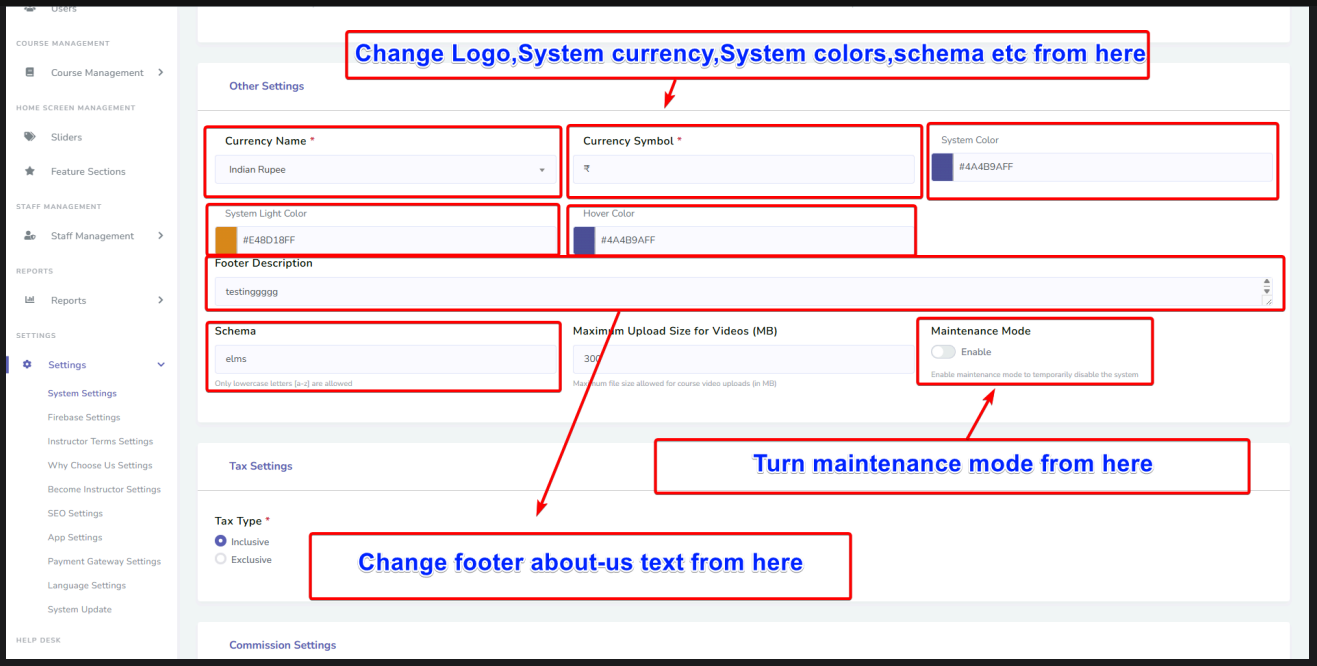


The changes will be reflected throughout your web application.

How to Set Logo, Theme Color, Footer Description,Maintenance Mode, System Currency, Schema etc

1. Go to admin panel **settings -> system-settings**

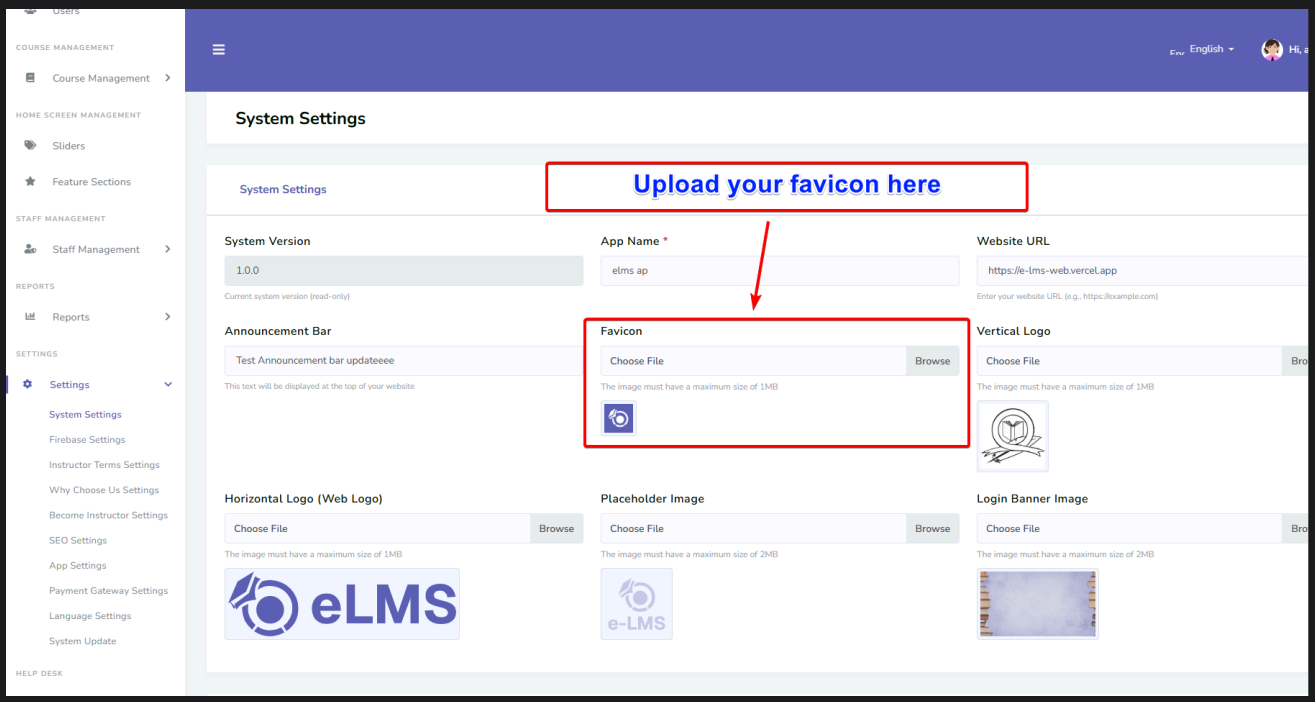



Change Logo,System currency,System colors,schema etc from here
Other Settings
Currency Name *
Indian Rupee
Currency Symbol *
₹
System Color
#4A4B9AFF
System Light Color
#E48D18FF
Hover Color
#4A4B9AFF
Footer Description
testingggggg
Schema
elms
Only lowercase letters [a-z] are allowed
Maximum Upload Size for Videos (MB)
300
Maximum file size allowed for course video uploads (in MB)
Maintenance Mode
Enable
Enable maintenance mode to temporarily disable the system
Turn maintenance mode from here
Tax Settings
Tax Type *
Inclusive
Exclusive
Change footer about-us text from here
Commission Settings
COURSE MANAGEMENT
Course Management
HOME SCREEN MANAGEMENT
Sliders
Feature Sections
STAFF MANAGEMENT
Staff Management
REPORTS
Reports
SETTINGS
Settings
System Settings
Firebase Settings
Instructor Terms Settings
Why Choose Us Settings
Become Instructor Settings
SEO Settings
App Settings
Payment Gateway Settings
Language Settings
System Update
HELP DESK

How to Set Favicon Icon

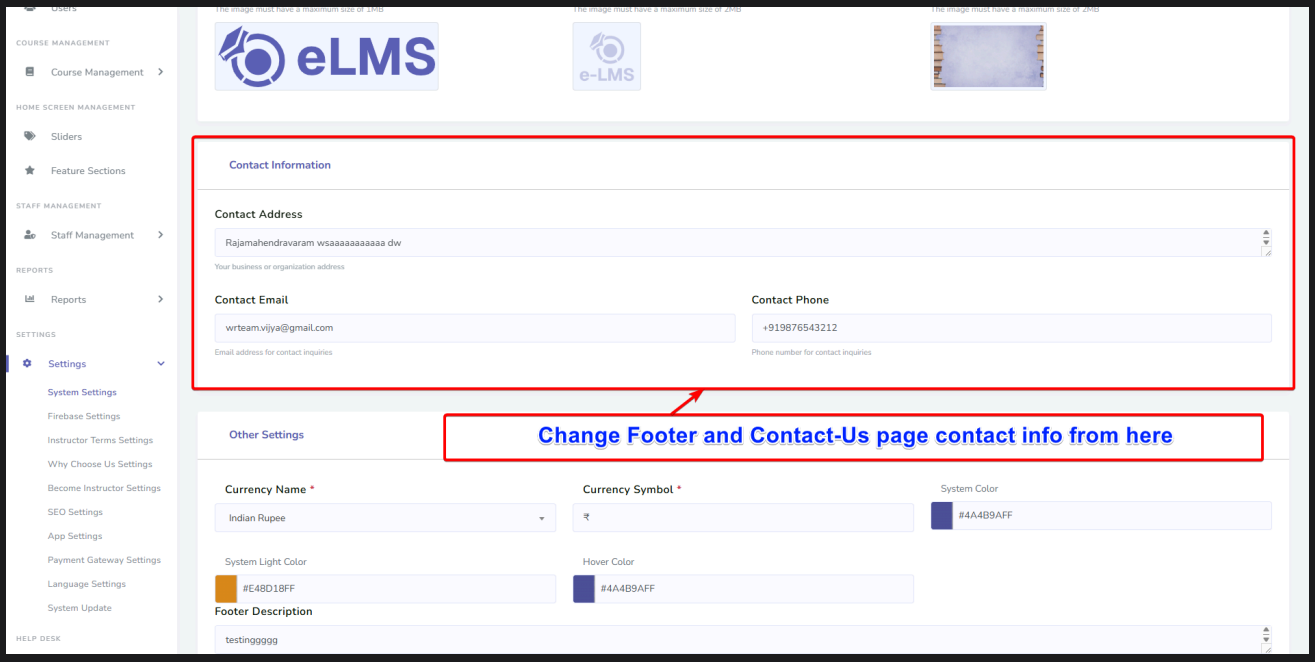
A favicon is a small icon that appears in browser tabs and bookmarks for your website.

1. You can generate favicon icons from this site: Favicon Generator
2. Go to admin panel **settings -> system-settings**
3. Upload your favicon icon



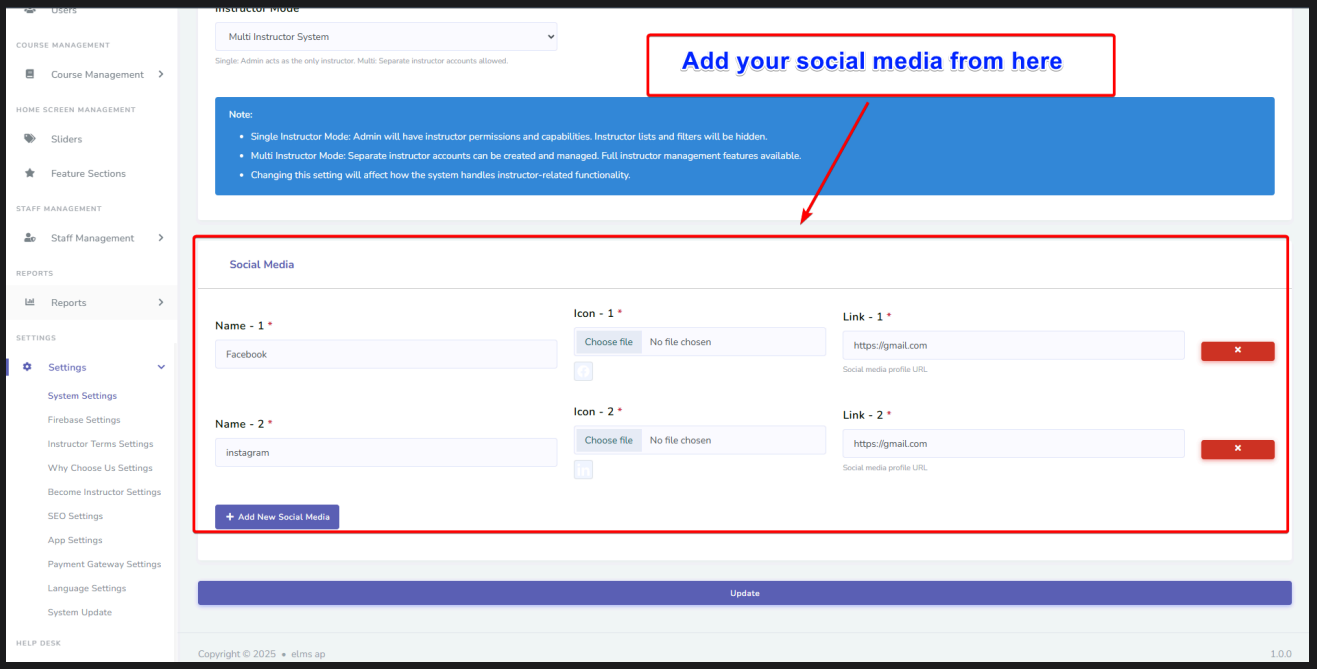
How to change contact details

1. Go to admin panel **settings -> system-settings**



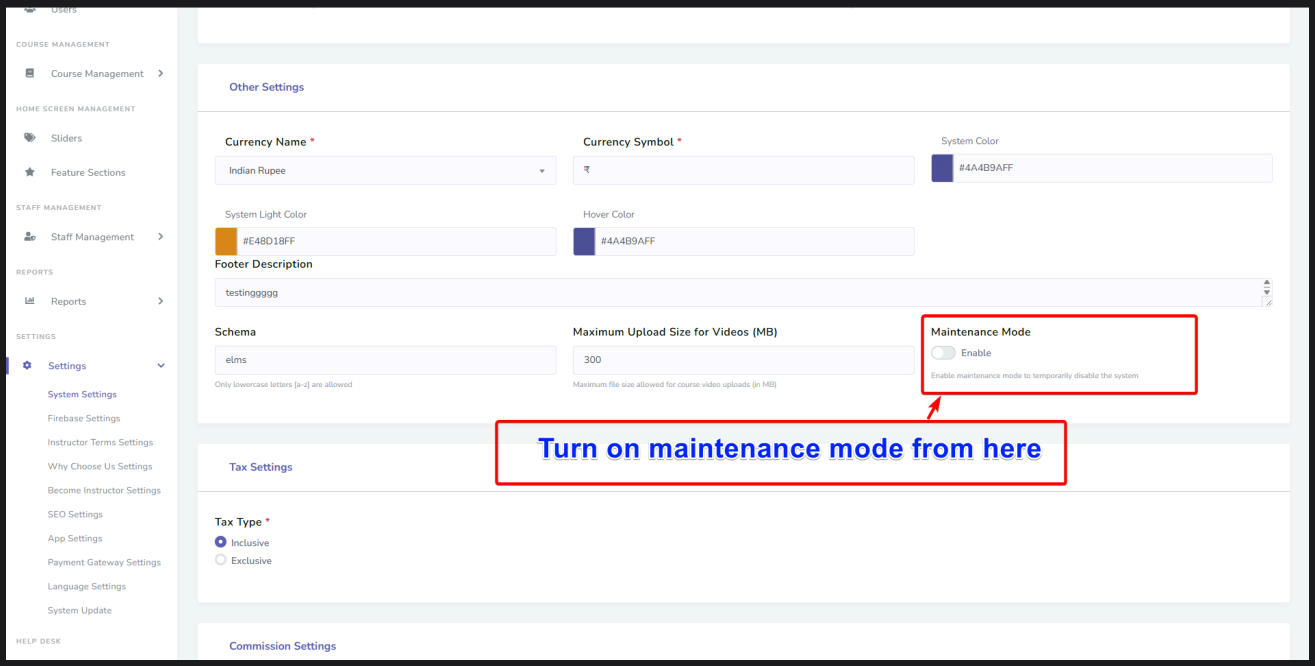
How to add Social Media Link and its Icon

1. Go to admin panel **settings -> system-settings**



How to On Maintenance Mode

1. Go to admin panel **system-settings -> general-settings**



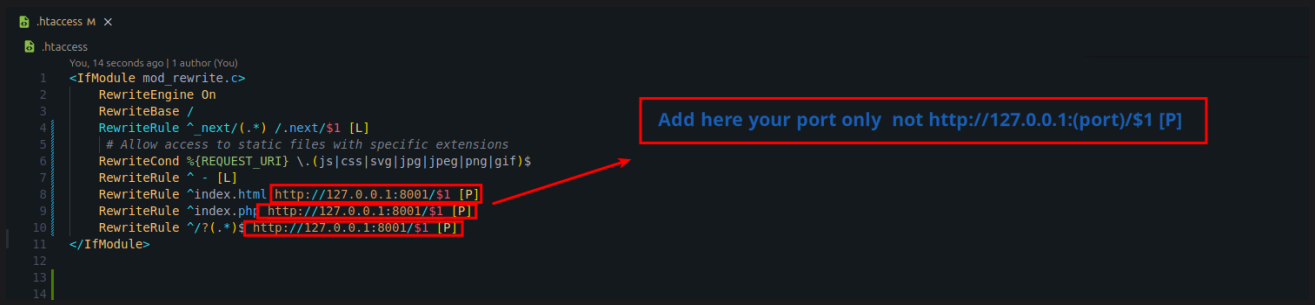
> **WARNING**
>
> If you on the Maintenance Mode it will be for both Web And App

How to run this project locally

1. Unzip the downloaded code. After unzipping you will have News - Web Code Folder.
2. Open VS Code teminal by typing **CTRL+J** in Windows/Linux, and for MacOS **CMD+J** and execute the command --> **npm install** This will take some time to download a few Packages so wait for a few minutes.
3. After **npm install** finishes run this command --> **npm run dev**. This Command will Start the development mode.Check If everything is working fine then your are good to go ahead.

.htaccess File Code

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /
RewriteRule ^_next/(.*) /.next/$1 [L]
# Allow access to static files with specific extensions
RewriteCond %{REQUEST_URI} \.(js|css|svg|jpg|jpeg|png|gif)$
RewriteRule ^ - [L]
RewriteRule ^index.html http://127.0.0.1:8001/$1 [P]
RewriteRule ^index.php http://127.0.0.1:8001/$1 [P]
RewriteRule ^/?(.*)$ http://127.0.0.1:8001/$1 [P]
</IfModule>
```



For Nginx Servers

If your server is running Nginx, copy the following code and paste it into your server's `nginx.conf` file's server block:

```
# nginx configuration by winginx.com
location ~ ^(.*)$ { }

location /_next {
 rewrite ^/_next/(.*) /.next/$1 break;
}

location / {
 if ($request_uri ~ "\.(js|css|svg|jpg|jpeg|png|gif)$"){
 rewrite ^/index.html http://127.0.0.1:8001/$1 redirect;
 }
```

```
 rewrite ^/?(.*)$ http://127.0.0.1:8001/$1 redirect;
}

location /index {
 rewrite ^/index.php http://127.0.0.1:8001/$1 redirect;
}
```

MANDATORY

Make sure you have paste this code inside server blocks.

Deploy the Site

This guide will walk you through deploying the Web application.

Prerequisites

> ⚠ **WARNING**
>
> If you possess a VPS server along with some familiarity with Node.js, npm, and pm2, you're well-equipped to deploy.

Required Resources

> ⚠ **MANDATORY**
>
> 1. **VPS Hosting**: A Virtual Private Server (VPS) is mandatory to ensure reliable performance and security. Shared hosting environments are not supported for this deployment method.
> 2. **Node.js Support**: The server must support Node.js , as it is essential for running the application with seo.
> 3. **Memory Requirements**: The server should have at least 3-4 GB of free RAM to handle the application's processes effectively.
> 4. **SSH Root Access**: The server must provide SSH root access to execute Node.js commands and manage the application.

Server Setup

> ⚠ **WARNING**
>
> (For SEO) Deployment of Next.js requires knowledge of `node.js`, `npm`, and `pm2` technologies. We assume you are using a `debian`-based OS with apt as your

> package manager. If you are using another Linux distribution, replace apt with your system's package manager.

Project Upload

Before starting deployment, upload your project to the server. You can use FileZilla or other file transfer methods.

Installing NodeJS

NodeJS can be installed using NVM by which multi Node version can be controlled easily:

```
sudo apt install curl
curl -o- https://raw.githubusercontent.com/nvm-
sh/nvm/v0.39.7/install.sh | bash
source ~/.bashrc
command -v nvm
nvm install node 20.*
```

Check if node js is installed correctly:

```
node -v
```

For more information, refer to the official documentation.

Installing PM2 Server

By running the following command, PM2 server can be installed globally:

```
npm install pm2 -g
```

Deployment Process

Refer to the NextJS deployment section for detailed steps on how to:

1. Set available ports
2. Configure the server
3. Install dependencies
4. Build the project
5. Run the PM2 server

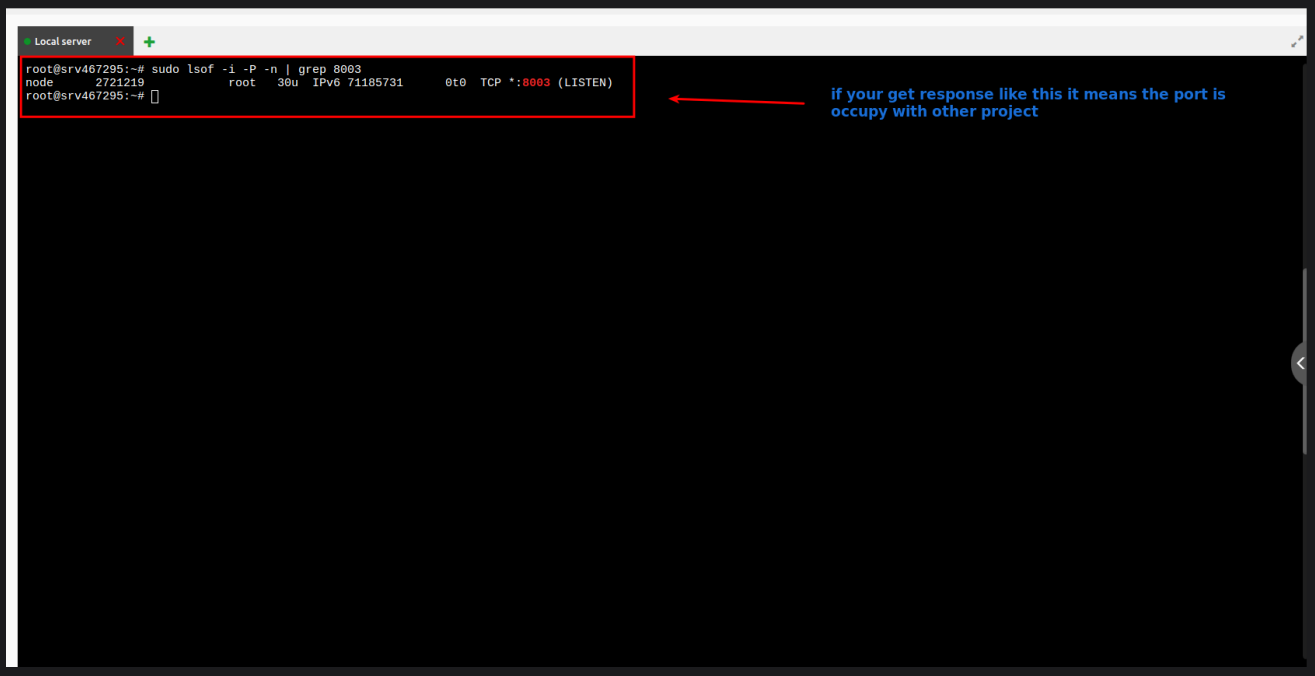
After successfully deploying with SEO, your site will have improved search engine visibility and performance.

NextJS Deployment

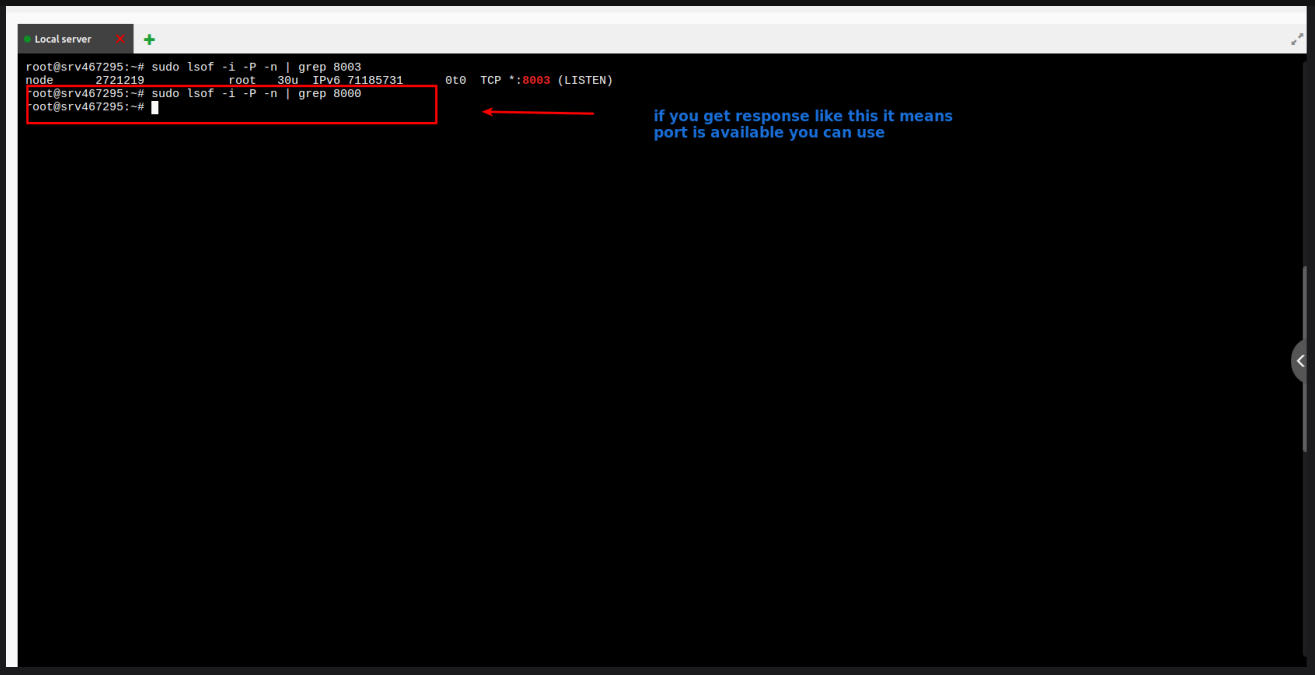
Port Configuration

Before setting up the port, check available ports with this command:

```
sudo lsof -i -P -n | grep 8003
```

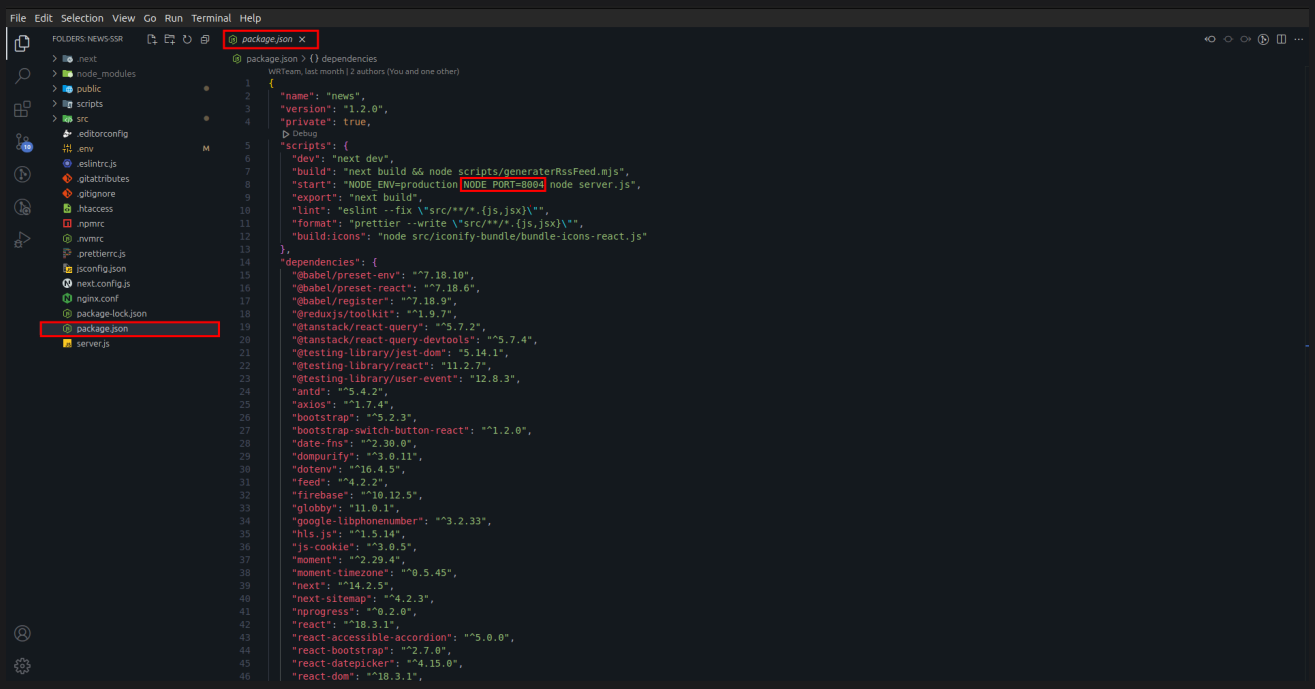


If you see a response like above, it means this port is occupied by another project. Try other ports like 8000, 8001, 8002, etc.



If you get no response, it means the port is available and you can use it.

Now add the available port to your `package.json` and `.htaccess` files:



Copy the .htaccess file code and change the port number as needed.

Project Setup

> ⚠ **WARNING**
>
> Make sure you have `node_modules` installed in your directory.

Install project dependencies:

```
npm install
```

This will install all the node modules in your directory.

After that, build the production application:

```
npm run build
```

Running the PM2 Server

Go to the project root and start the PM2 server:

```
pm2 start "npm start" -n "YOUR_PROJECT_NAME"
```

Check if PM2 process is running correctly:

```
pm2 ls
```

When you run `pm2 ls`, you will see one of two types of output:

1. Error:

2. Success:

If you're getting errors, run `pm2 logs` and check the error details.

If successful, set up a startup script to ensure PM2 restarts automatically after a system reboot:

```
pm2 startup
```

After setting up PM2 with the startup command, save the current process list

```
pm2 save
```

If you want to restart your pm2 process then run pm2 restart id // Replace id with your process id

For example here id is 0 in the above screenshot **pm2 restart 0**

For deleting the previous project running in the PM2 server, use the following command

```
pm2 delete "YOUR_PROJECT_NAME"
```

Your News Web application should now be running with full SEO capabilities.