1. Introduction

Thank you for purchasing the Miningcore Web UI. The product comes with full source code (.NET, Vue, CSS/SCSS). Before deploying the Web UI to your server(s) you should take the time to read this document carefully and perform at least the basic customization steps described in section 3.

2. Troubleshooting

You received a link to our customer support site with your order confirmation. Use it. We are happy to help!

3. Prerequisites

The following software needs to be installed on your production system(s):

- ASP.NET Core Runtime 6 (available here) on your webserver
- Miningcore Mining Pool with API enabled in configuration and reachable by web server

The following software needs to be installed on your development system:

- Visual Studio 2022 (Community Edition is sufficient)
- NodeJS LTS (available here)

4. Customization before deployment

- Edit `src/WebApp/appsettings.json`
  - Set `MiningCoreApiEndpoint` to your pool's internal API Url
  - Set `PoolDomain` to your pool's domain
  - Set `TwitterUrl` to your pool's twitter profile if applicable
  - Set `SupportEmail` to your pool's support Email if applicable
- Replace `src/WebApp/Resources/Images/logo.png` with your pool's logo
- Edit `src/WebApp/Resources/Strings.resx` and change at least the following properties:
  - `OpenGraphAppName`
  - `OpenGraphAppTitle`

    - `OpenGraphAppDescription`
    - `WelcomeH1Fmt`
    - `WelcomeH2_1`
- Customize `src/WebApp/Styles/main.css` to your liking
- Favicon
    - Open https://realfavicongenerator.net/ in a web browser
    - Select your logo image
    - On the next page scroll to bottom and click “Generate your Favicons and HTML code”
    - Click “Download your package”
    - Extract the downloaded .zip archive into `src/WebApp/wwwroot`

5. Running the Website locally for further customization and development (**not required**)

IMPORTANT: Please make sure that Miningcore’s REST API is running and reachable from your development machine. The location of the Miningcore REST API endpoint is configured in `src/WebApp/appsettings.json` with the `MiningCoreApiEndpoint` property.

You can even work with the API of a remote Miningcore Server by setting `MiningCoreApiEndpoint` to http://localhost:4000 and forwarding the API to your local machine using: `ssh -N -L 4000:localhost:4000 <address or hostname of your Miningcore Server>`

1. Open the solution (WebApp.sln) in Visual Studio 2022
2. Open a shell and navigate to `src/WebApp/`
3. In the shell run
    a. `npm install`
    b. `npm run dev`
4. Press F5 in Visual Studio 2022 to start debugging
5. Open http://127.0.0.1:5000 in a web browser

6. Creating a production build

To build the web application for deployment to your server, open a shell and navigate to src/WebApp then run:

```
dotnet publish WebApp.csproj -c Release --framework net6.0 -o ../publish
```

This will package the whole web application into the parent folder “publish”. The contents of the folder can then be copied to your server. It is commended to compress the folder into an archive before deployment to save time and space.

7. Deployment Notes

- To actually run the website on your server make sure that the ASP NET Core 6.0 Runtime is installed on the server (available here)
- To start the web server run the following command inside of the publish folder created in the previous section, run: `./WebApp` or `dotnet WebApp.dll`

- For production use you should create a service on your server that runs the above command. A template for a Linux systemd service is included in `devops/kestrel.service`.
- It is recommended to run the UI’s internal Web Server (Kestrel) behind a reverse proxy such as nginx. An example nginx site config is included in `devops/nginx.conf`

8. Localization

The web application was developed from the ground up to support multiple languages and culture. The shipping version just includes US-English but can be extended to support additional languages by following the following steps:

- The first step is to look up the culture code for the culture and language you want to support. Open this page, locate your desired culture and note the corresponding value from the right most column (“CultureInfo Code”). If you mostly care about the language and it is your priority to support as many people speaking a language regardless of their country and culture you should use the shorter value from the “Two Letter Lang Code” column (ie. en, de, zh etc.)
- Edit `src/WebApp/AppConstants.cs` and install your code by uncommenting `new("de")` and replacing “de” with your value.
- Now create a copy of the file `src/WebApp/Resources/Strings.resx` and rename it to `src/WebApp/Resources/Strings.<your code>.resx` (Example: for culture code “zh” the file should be named `src/WebApp/Resources/Strings.zh.resx`). Be careful to not change the file extension while renaming it. The extension must be .resx.
- Now open newly created file and begin translating all variables in the file. The easiest way to do this is using Visual Studio’s resource editor but you can also edit it in any Text- or XML-Editor.

- Open `src\WebApp\SPA\Resources\translations.json` using Visual Studio, Visual Studio Code or any other JSON capable text editor. Copy the “en” section until the start of the “de” section. Replace the key “en” again with your code (de, es, zh etc.). And begin translating all keys inside the newly created section.
- The next stop is a bit more involved. Create a copy of each and every file in src\WebApp\Views\Pool\Help\algos and change the ending from en.cshtml to <your code>.cshtml (Example: for culture code “zh” the file `generic-en.cshtml` should be named `generic-zh.cshtml`). Once again be careful to retain the file extension .cshtml.
- Now edit each every newly created file and translate the text inside it while being careful to not to mess up the code parts.