# Get project and setup build environment

## Get the source code

Use the provided source code.

Unzip the source code archive to a directory of your choice.

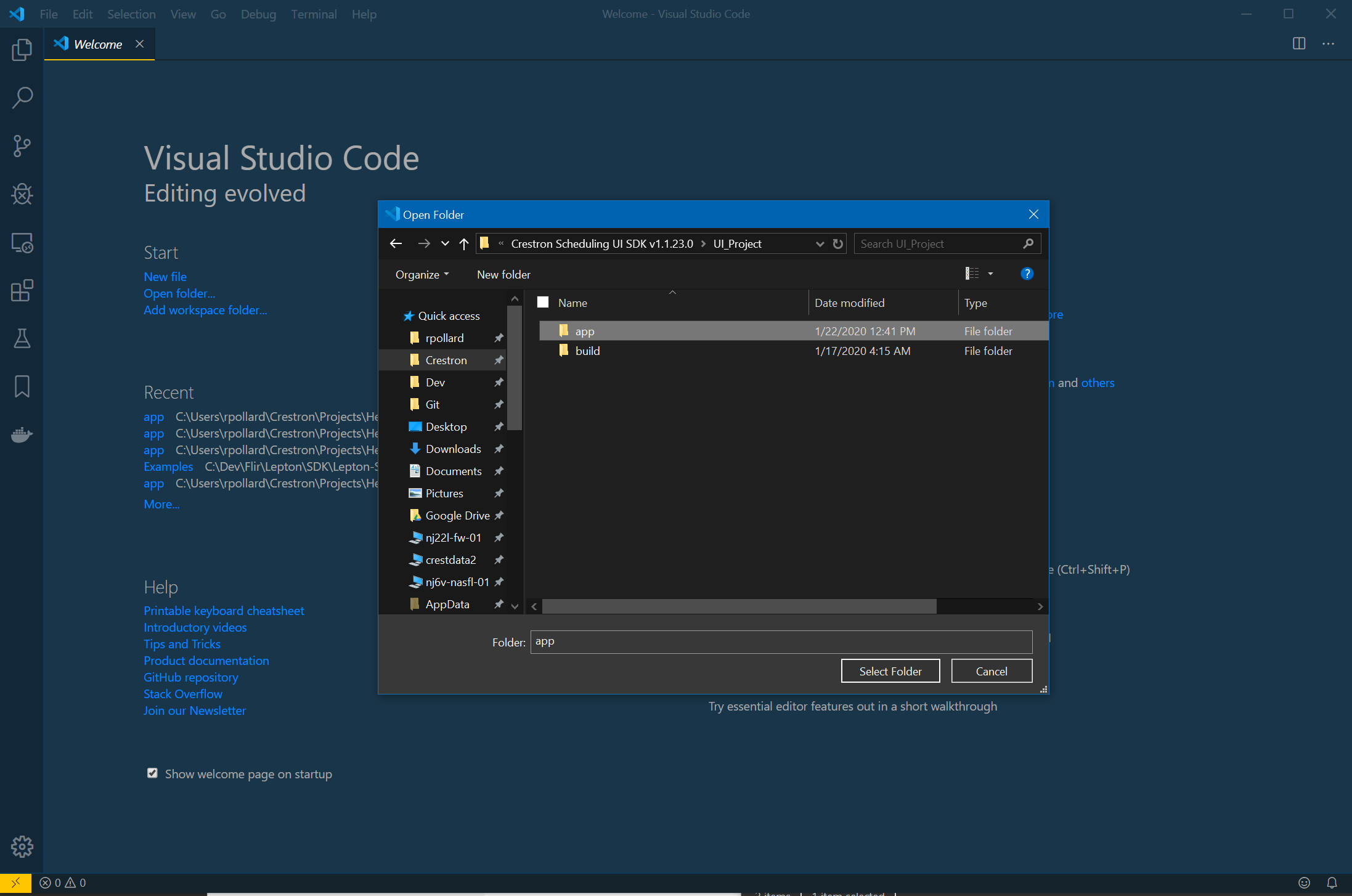
## Get the code editor

Download and install Visual Studio Code. Visual Studio Code is being used for this demo because it is free and cross-platform, but this is the developer’s choice.

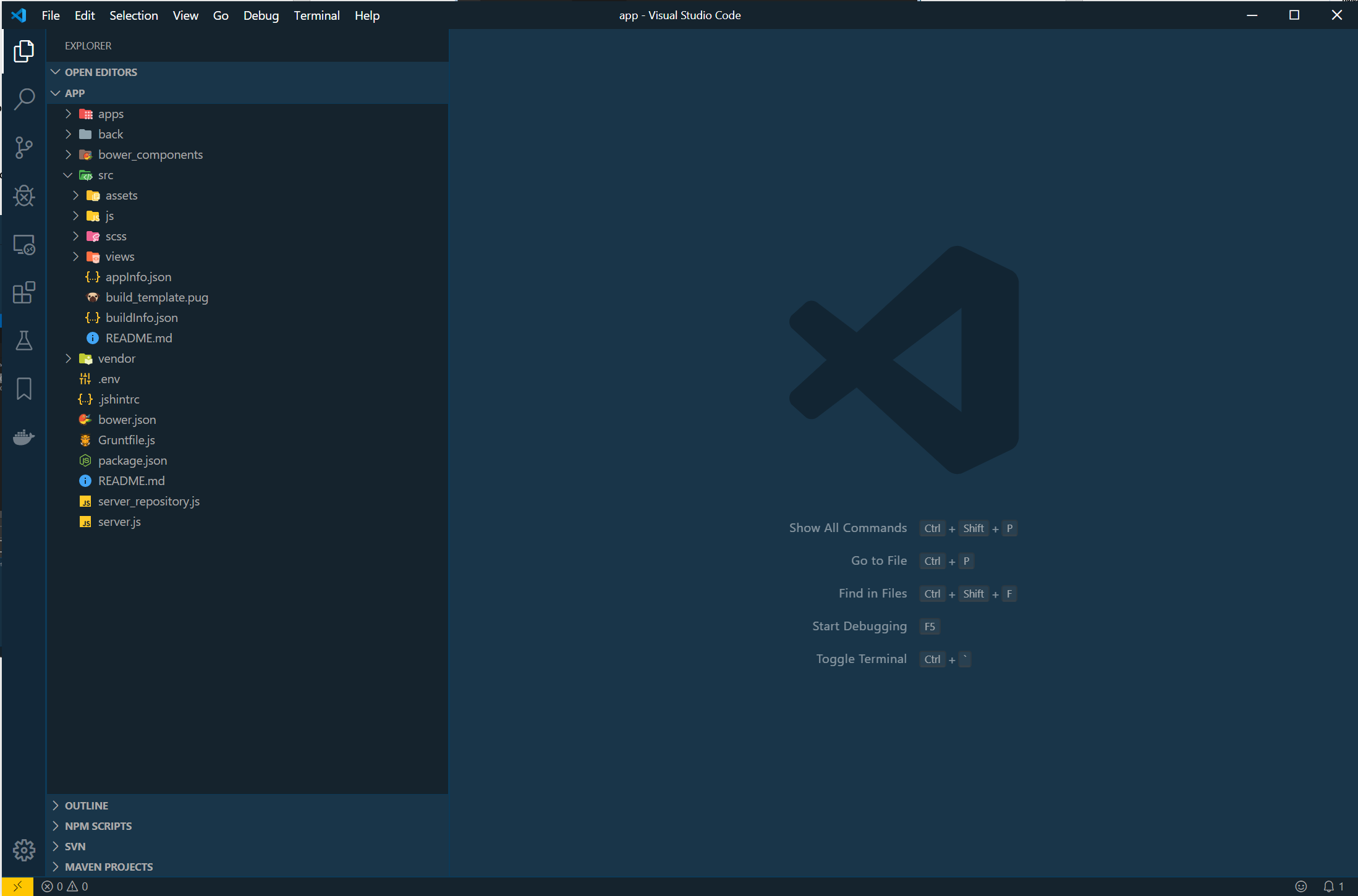
- <https://code.visualstudio.com/Download>

## Open the project

After installing Visual Studio Code, open the project by selecting the “UI\_Project/app” folder, within the source code folder, in the “Open folder” dialog.



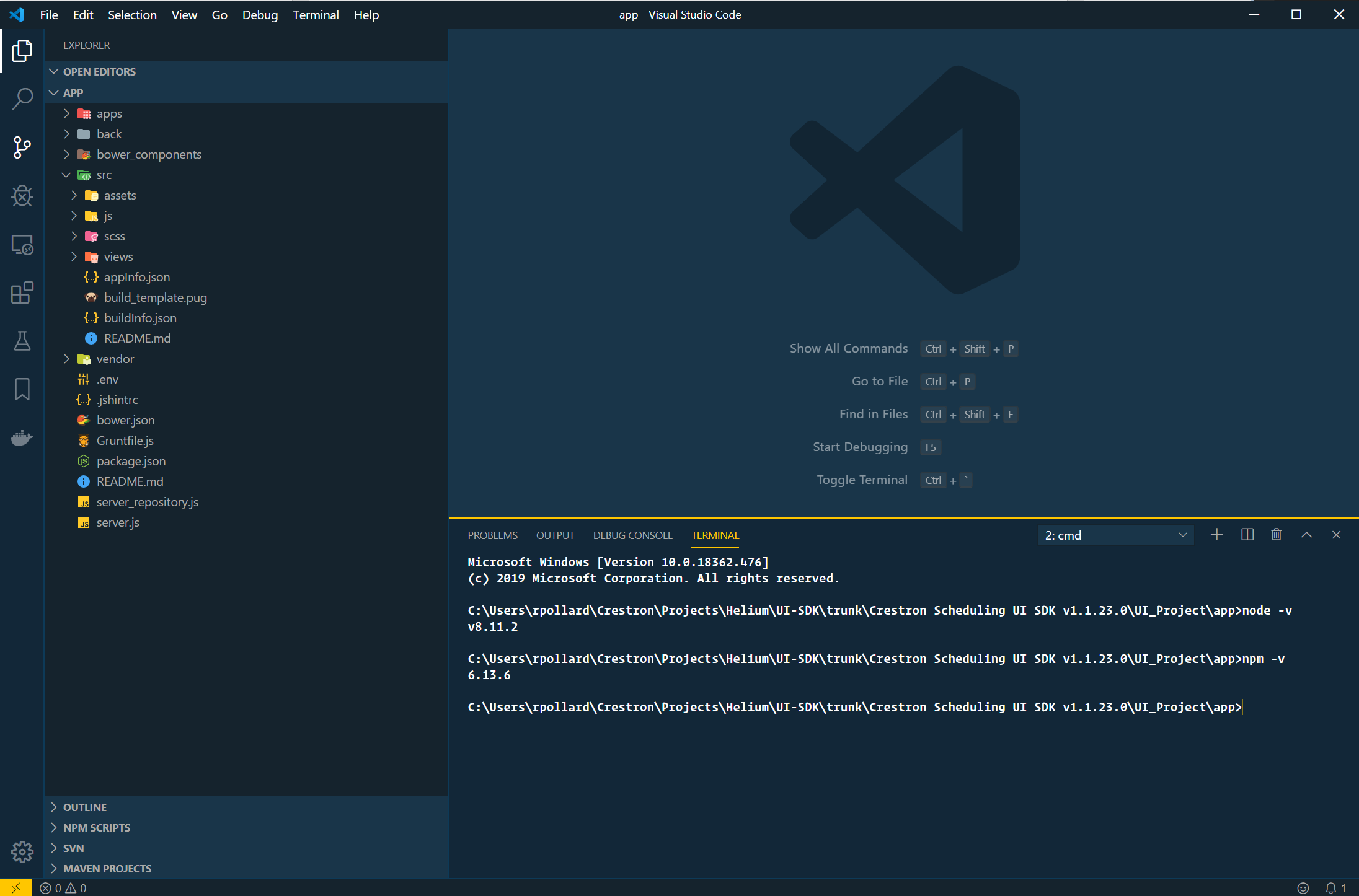
You should now see something like this in VS Code:



NodeJS is required to fetch the necessary packages, so download it for your platform and install. There is an install option to add Node and npm to PATH by default. This should be left on.

- <https://nodejs.org/en/download/>

It’s a good time to check that the NodeJS environment variables were added successfully during install. In the Windows command prompt, which can be accessed directly from VS Code by selecting “View” 🡪 “Integrated terminal”. When you have the prompt, type “node –v” and “npm –v”. Both should return a version number as seen below. If the commands are not recognized, try restarting VS Code. Worst case, you may have to add them manually.



## Install development dependencies

Next, install the project dependencies.

- In the VS Code terminal, type the command “npm install --g bower” and press enter. This will install the latest version of Bower.

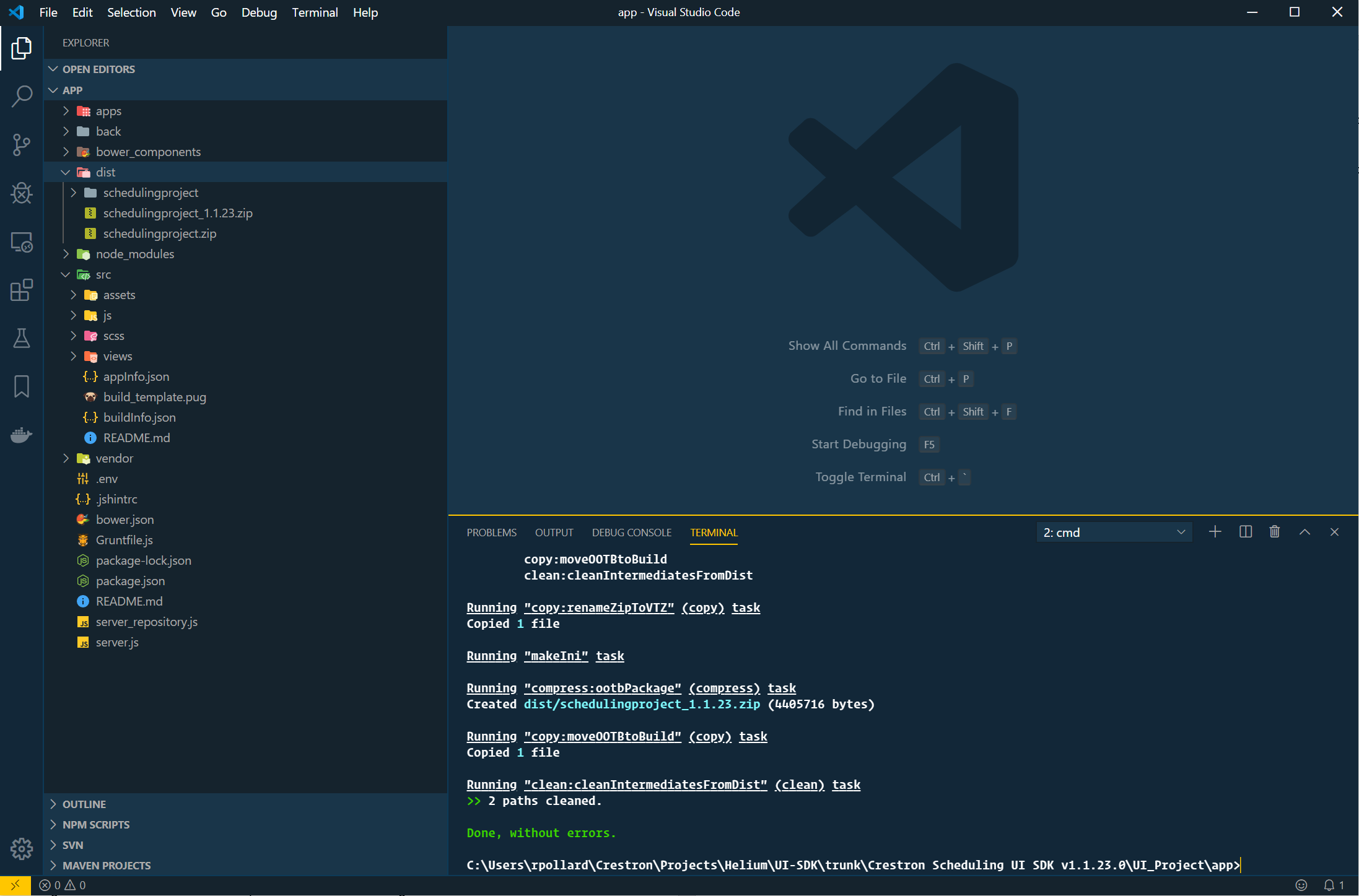
- In the VS Code terminal, type the command “npm install --g grunt” and press enter. This will install the latest version of Grunt.

- In the VS Code terminal, type the command “npm install” and press enter. This should install all Node and Bower dependencies. You will see output logging in the terminal.

# Build the project

## Try a full build

You should now be able to build the existing project. If the terminal is still open and the prompt is pointing to the project’s “app” folder (as seen in screenshot below), type “grunt --app=schedulingproject --dist” and press enter.



If the build was successful, “Done, without errors” should be printed to the terminal, and a “dist” directory should have been created that includes the output files.

Either of the zip files can then be FTP’d to the panel’s “firmware” directory and installed with the “OOTBPROJECTLOAD” Text Console command.

It is also possible to load the project from the panel’s Web Configuration page. To do so, select the option from the “Actions” menu and browse to the file.

## Build notes

Arguments that can be passed into grunt build:

- “--app=”: (Required) Must be the name of a folder in the “Apps” directory. Also determines the output filename.

- “--initbuild”: Used when CSS customizations are needed for a project. To use, first create a folder with the name of the project in the “apps” directory. Then, run “grunt --app=[appname] --initbuild”. This will compile all of the HTML, JavaScript and CSS, and copy the files into the project folder, where changes can be made that do not affect the source code.

- “--build”: Used when customizations are needed for a project. After making customizations to files in the project folder, run the command “grunt --app=[appname] --build” to copy the modified files into the “dist” directory and create a project archive.

- “--debug”: Used for debugging. Files will be compiled and copied to the “public” directory, where they can be viewed and debugged using your web browser. (Note: Debugging with the local browser may be used in conjunction with the “--emulate” command to associate the UI with a JavaScript file that will emulate a scheduling provider. That file is SchedulingPanelEmu.js and can be found in the “vendor” directory.

- “--emulate”: Links the project with a JavaScript file (vendor/crestron/schedulingPanelEmu.js) that emulates a calendar connection. Used for testing and debugging with the local browser. Note: Emulation works by using a Node server instance running on the local machine. The default port is 3000. This can be changed in the “.env” file.

- “--dist”: This is only used when the project needs to be loaded to a Crestron panel. It will package the project in a zip file that contains that necessary components for firmware to recognize it as a scheduling project. The output file will be copied to the “dist” directory and should have the version number appended to the filename (ex. “schedulingproject\_1.1.23.zip”). The zip file can then be placed in the panel’s “firmware” directory and installed with the “OOTBPROJECTLOAD” Text Console command.

The only time “initbuild” and “build” should be used is if a small change needs to be made for a project. This keeps all the code in “src” intact. All files are compiled, and the resulting HTML, CSS, and JavaScript files are copied to the new project folder, where customizations can be made.

If making significant changes to the source code, just make the changes directly in the “src” folder files and use the “grunt –app=schedulingproject” command to build.

It is possible to create a debug build with emulation. This is useful during development. Just append “—debug –emulate” to the end of the build command.