# Exercise: Continuous Integration – Part II

Exercises for the ["Back-End Test Automation"](https://softuni.bg/trainings/4399/back-end-test-automation-february-2024) course @ SoftUni

## Install Jenkins

Our first task is to install Jenkins on our machines.

In order to do that, follow this link: <https://www.jenkins.io/download/> and chose the package that is suitable for you and your machine.  
The installation for the different operating systems and their distributions are different. You can find the instructions that you need here: <https://www.jenkins.io/doc/book/installing/>. Simply chose your OS and follow the instructions.

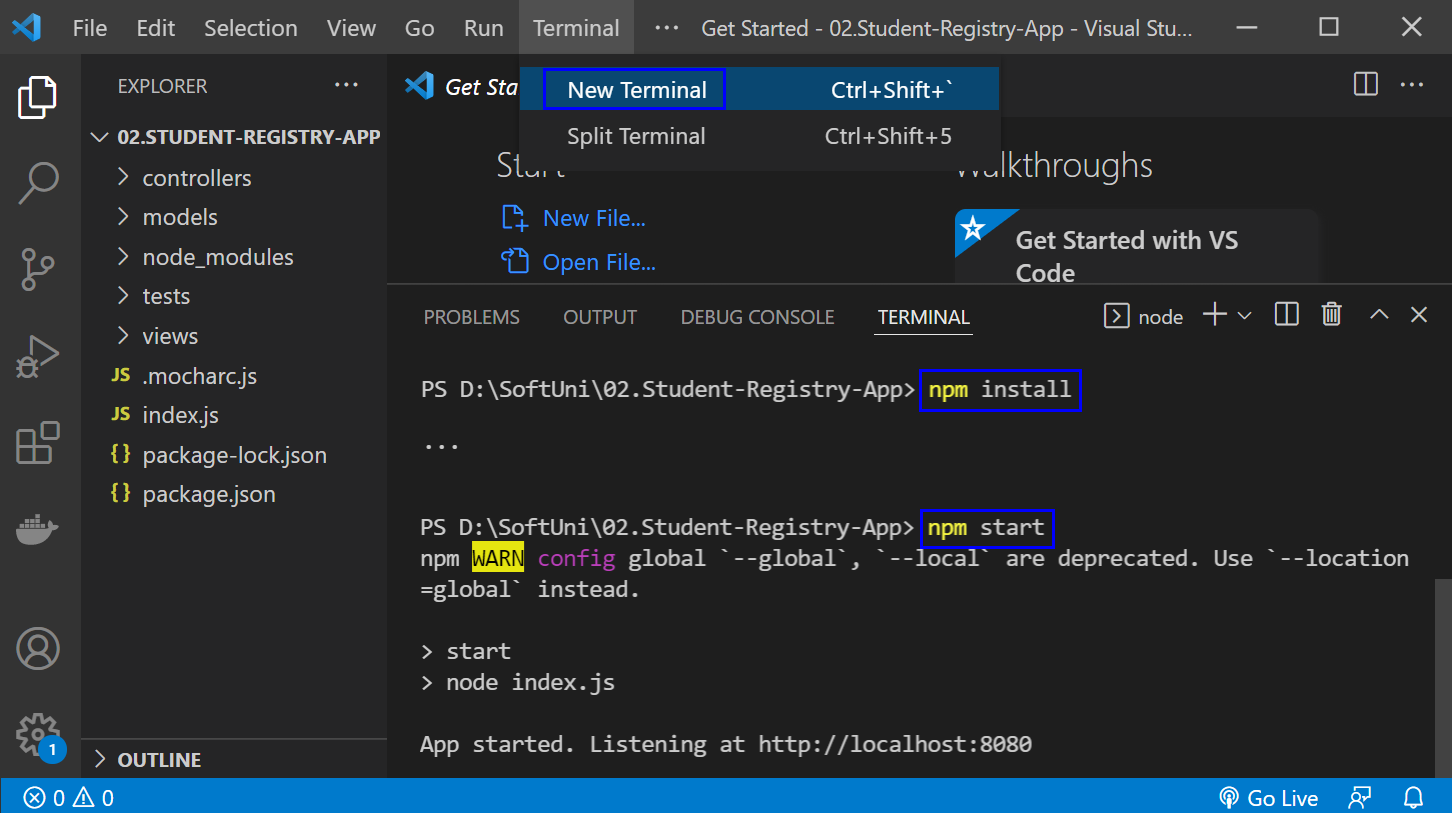
After you have installed Jenkins, follow the **Post-installation setup wizard** in order to **start** using Jenkins. Without completing the steps from it, you won't be able to use it. This is a one-time setup, so don't worry – you won't need to complete those steps each time you want to work with Jenkins.

## "Student Registry" App – Node.js App

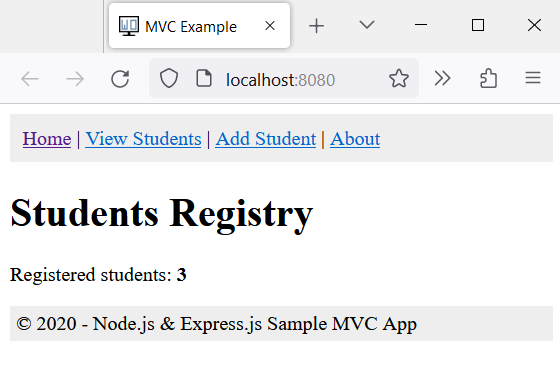
### Step 1: Run the App Locally

We have the "Student Registry" Node.js **app** in the **resources which has some integration tests already**. Your task is to **create a CI workflow** with **Jenkins** to **start and test the app**:

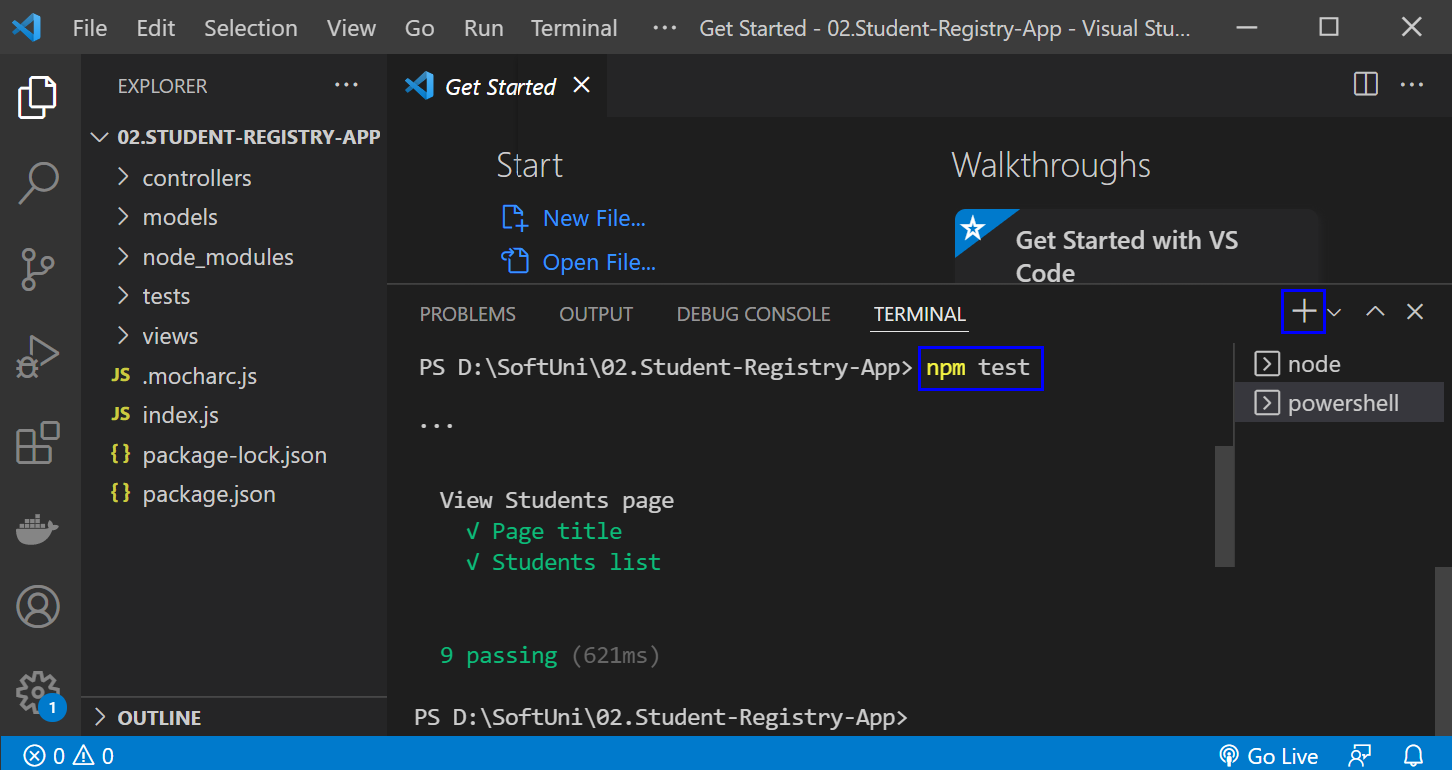
Let's first **start the app locally** in Visual Studio Code. To do this, you should **open the project**, open a **new terminal** from [Terminal] 🡪 [New Terminal] and **execute** the "npm install" and "npm start" **commands**:



The "npm install" **command** **installs app dependencies** from the package.json **file** and "npm start" **starts the app**. You can **look at the app** on <http://localhost:8080>:



Then, you can **return to** VisualStudioCode, open a **new terminal** with [+] and **run** "npm test" to **run the app tests**. They should be **successful**:



**NOTE**: if the **app was not started**, **tests would fail** because these are integration tests and they are executed on the running app.

### Step 2: Create a GitHub Repo

Now you should **upload the app code to** GitHub.

It's a good practice to start using the console and not the interface of GitHub, in case you haven't started doing so yet.

If you don't have Git already installed on your machine, follow the provided installation instructions from the resources.

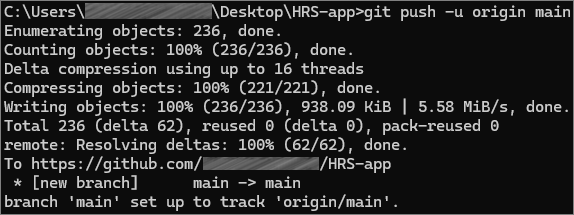
Try using the following commands in order to initialize a repository in your project directory, add the code to the repo, commit and push:









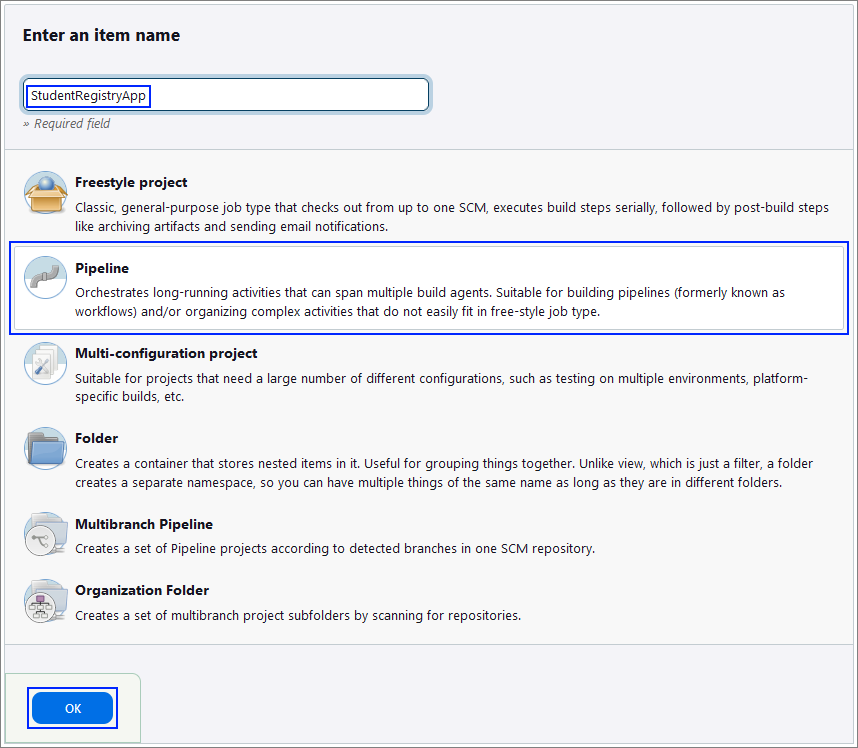


After running the commands, check you GitHub repo – the application code should be visible.

### Step 3: Create a New Job

Now, let's access Jenkins. Open the Jenkins interface in a web browser. This is usually at [**http://localhost:8080**](http://localhost:8080), but it depends on the **port that you had set up during the installation.**

Let's create a new job by selecting **[New Item]** from the **Jenkins dashboard**. Choose **Pipeline** and give it a **meaningful** name, after that click on the **[OK]** button.



### Step 4: Create the Jenkinsfile

**Best practice** for using a **Jenkinsfile** is to keep it **within** **your** **source** **control** **repository**.

This approach has several advantages like version control and branch specific pipelines. Placing the Jenkinsfile in the repository, means that it will be versioned alongside your application code and the versions can later be reviewed. Also, you can have different Jenkinsfile **versions** in **different** branches, which allows for testing changes to the build process in a feature branch before merging them to the main branch.

The Jenkinsfile should contain **steps** for:

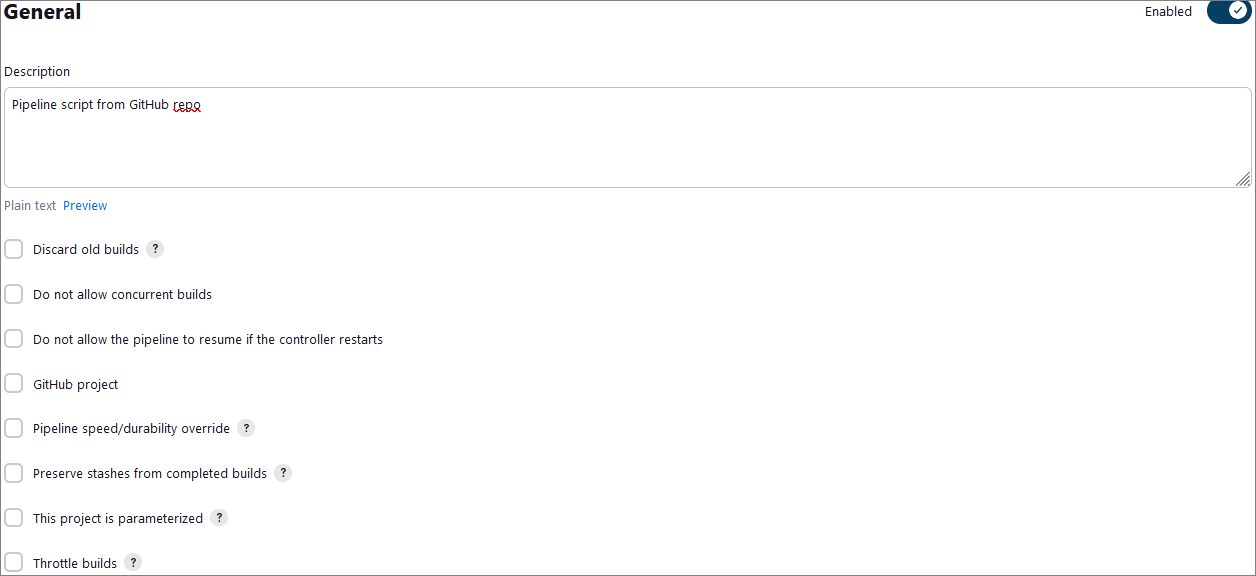
* Checking out the code
* Setting up Node.js
* Installing dependencies
* Starting the application
* Running tests

Create your file and upload it to your GitHub repository, containing the code for the application.

### Step 5: Configure the Job

Now, let's **go back** to **Jenkins** to finish **configuring** your **job**.

First, in the **General** section give a **Description** for the job.

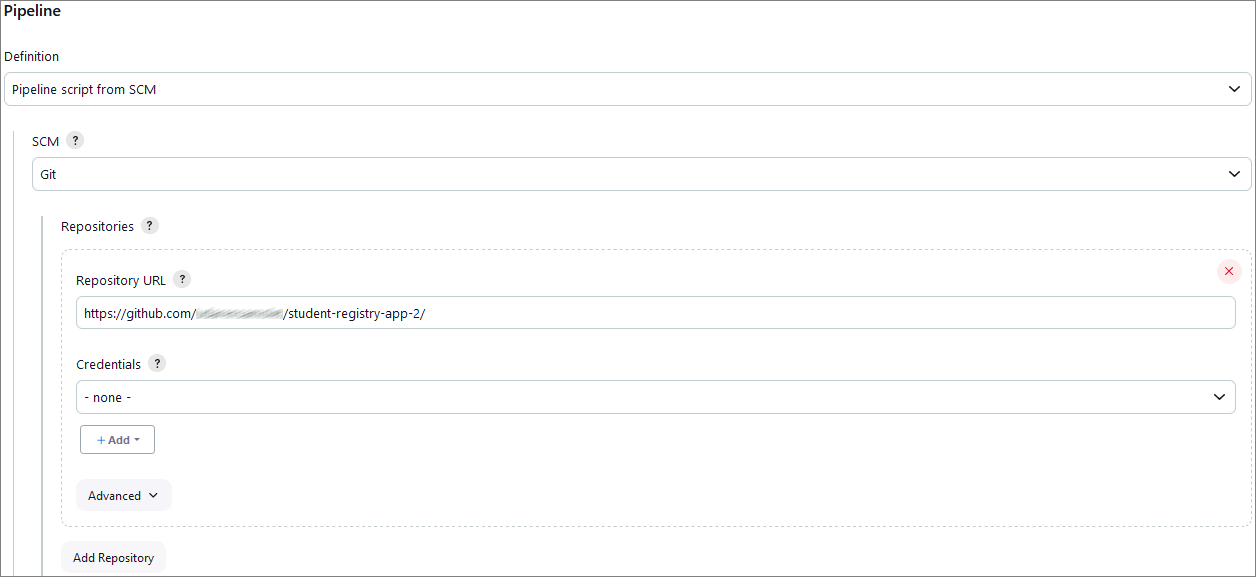


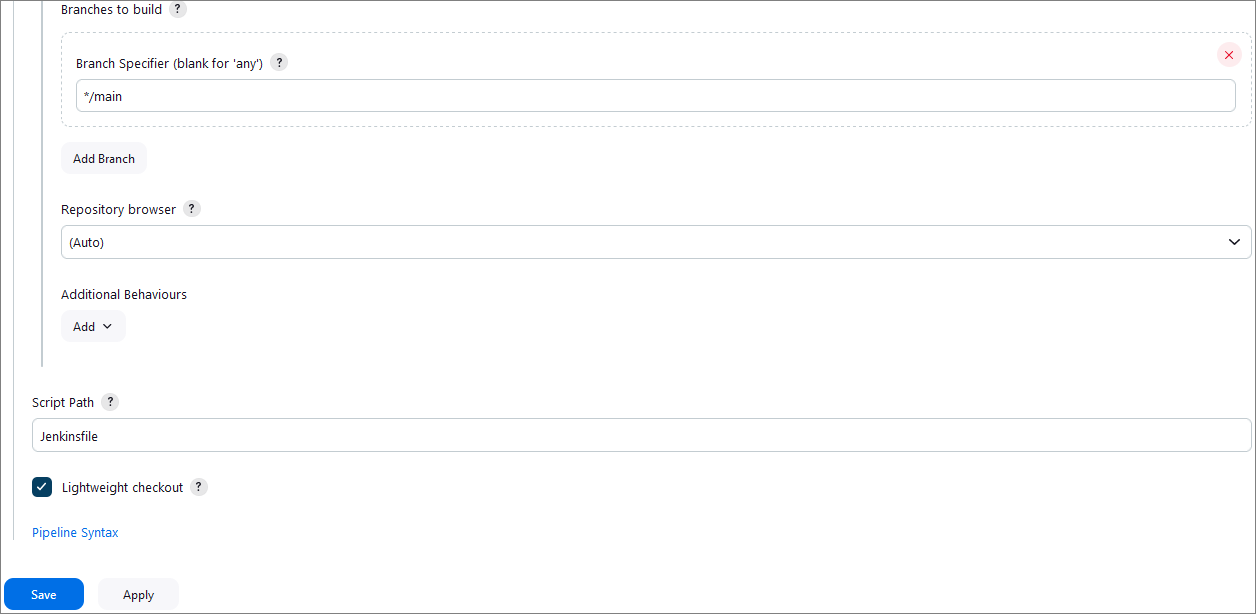
Then, scroll down to the **Pipeline** section in the job configuration, and from the **Definition** dropdown menu, select the **Pipeline script from SCM** option.

After that, select **Git** as the **SCM** and enter **your** **GitHub** **repository** **URL**.

Under **Branches to build**, enter the **branch** **name** that contains your **Jenkinsfile**.

Under **Script Path**, ensure it points to your **Jenkinsfile** (for example, type in **Jenkinsfile** if it's in the repository root).

Your configuration should look like the images below:  
****

****

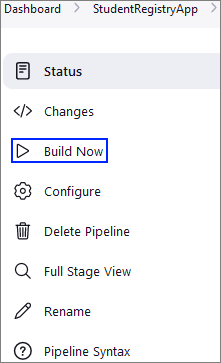
Finally, click on the **[Save]** button.

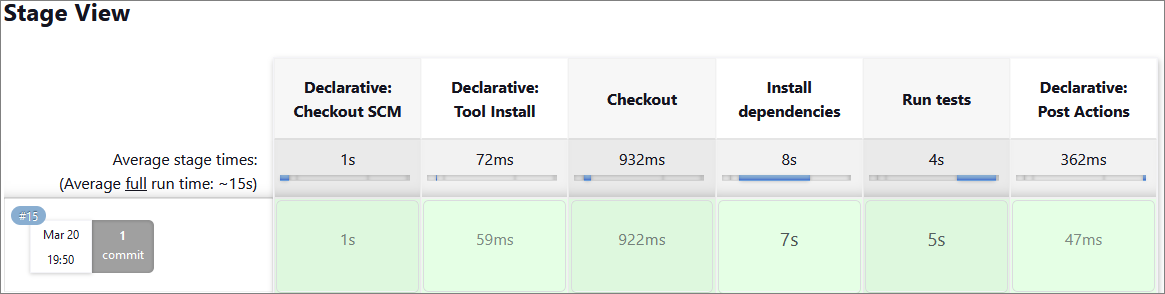
### Step 6: Test the CI Pipeline

After completing those steps, we are ready with the CI pipeline and it's time to test if it's working as expected.

First, click on the **Build Now** option to start a new build manually.

You can monitor the build progress by clicking on the build number and then **Console Output**.





### Step 7: Build Triggers

Setting up **build triggers** in **Jenkins** to initiate **builds on commits** to the GitHub repository involves configuring a webhook in GitHub. This **webhook** will **notify** Jenkins **each time a commit is pushed to the repository**, **triggering a** **build automatically**.

To do that, we have to configure webhooks in GitHub and configure the Jenkins job.

As our Jenkins server is **not** on a **public** IP address, first, we are going to use a tunneling service to expose our local Jenkins server to the Internet **temporarily**. Here's how to do it:

* Download and run **ngrok**:
  + Download **ngrok** and run it on your machine.
  + Use the command **ngrok** **http 8080** (this is the port that your local Jenkins installation has been configured to use during the installation)
  + **ngrok** will provide you with a public URL (e.g., **https://xxxx-xx-xxx-xx-xxx.ngrok-free.app**).
* Update Webhook in GitHub:
  + Use the **ngrok** URL followed by **/github-webhook/** as the payload URL in the webhook settings on GitHub.
* Keep **ngrok** running:
  + **Ensure** that **ngrok** is running whenever you want GitHub to trigger Jenkins.

Now, navigate to the GitHub repository that is used for the application. Click on the **Settings** tab in the GitHub repo. In the settings menu, find and click on **Webhooks.** Click the **[Add webhook]** button.

The webhook settings should be the following:

* **Payload URL**: Enter your Jenkins server's URL followed by **/github-webhook/**. For example,  [**https://xxxx-xx-xxx-xx-xxx.ngrok-free.app/github-webhook**](http://abc123.ngrok.io/github-webhook/)/.
* **Content type**: Choose **application/json**.
* **Secret**: Optionally, you can set a secret token for additional security (make sure to remember this as you will need it in Jenkins).
* **Which events would you like to trigger this webhook?**: Select **Just the push event**.
* **Active**: Ensure this checkbox is selected.

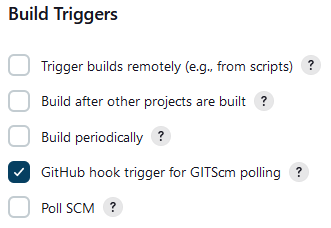
Finally, click on the **[Add webhook]** button to save the settings.

With that, we have set up GitHub to notify Jenkins for each new commit.

Now, let's modify our Jenkins job to trigger on GitHub webhook notifications.

To do that, go back the Jenkins dashboard and open the job that we created for the application. Click on **Configure** and select **Source Code Management** again.

This time, in the **Build Triggers** section, select **GitHub hook trigger for GitHub hook trigger for GITScm polling.**

****

### Step 8: Test the CI Pipeline

After completing those steps, we are ready with the CI pipeline again and it's time to test if it's working as expected.

First, make a minor change in the app code and commit and push this change to the repo, holding the application. This will trigger the Jenkins job and in the console output we can check if there are any errors.

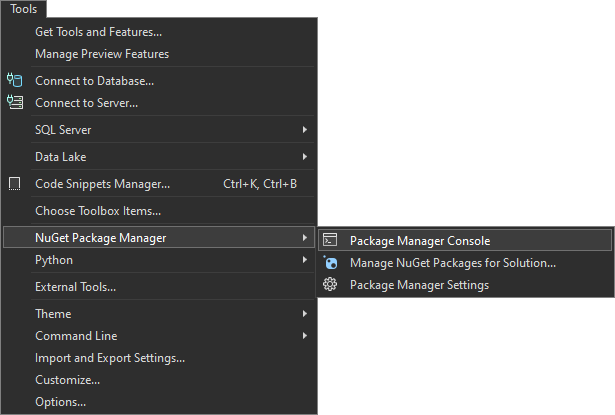
## "HouseRentingSystem" App – ASP.NET Core MVC app

### Step 1: Run the App Locally

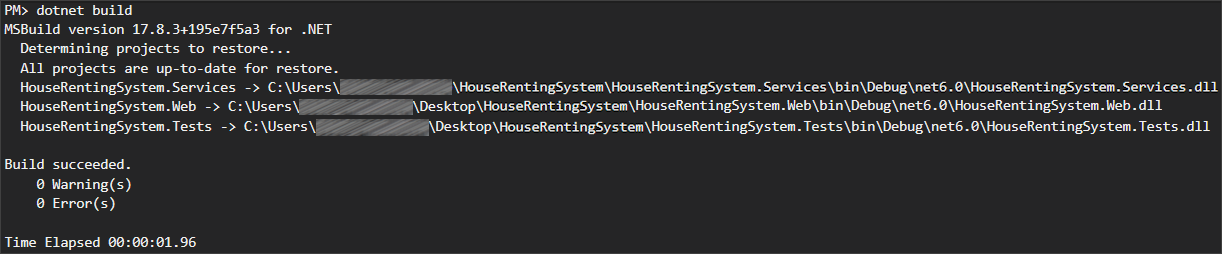
We have the "HouseRentingSystem" ASP.NET Core MVC **app** in the **resources which has some unit and integration tests already**. Your task is to **create a CI workflow** with **Jenkins** to **start and test the app.**

It's a good practice to first **start the app locally** in Visual Studio, in order to be sure everything works properly and as expected.

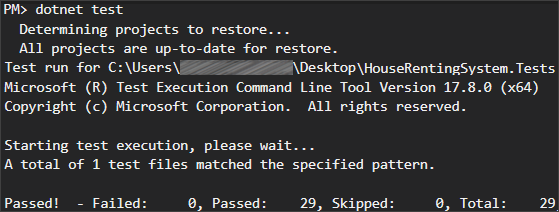
Open **Visual Studio** and from there navigate to the **Tools** menu. Select **NuGet Package Manager** and select **Package** **Manager** **Console**:



Let's first build the application by using the **dotnet build** command:



After you have **ensured** that the **build** was **successful**, you can **run** the **tests**, too, by using the **dotnet test** command:



**NOTE:** Visual Studio has built-in test runners that allow you to run your tests directly from the IDE. This is the simplest way to execute tests if you're already working within Visual Studio. However, it's **better** to get used **using** the **console**.

**After** we have ensured that the **tests** **run** **successfully**, we can proceed with the next step.

### Step 2: Create a GitHub Repo

Now you should **upload the app code to** GitHub. Try using the CLI and the commands from the previous task to add the code to the repo and commit it.

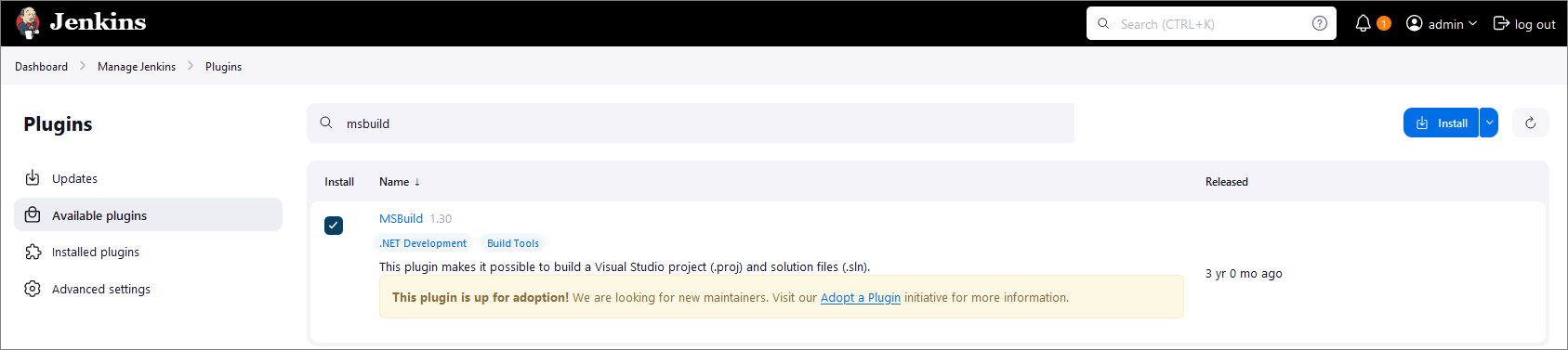
### Step 3: Configure Tools in Jenkins

To run an **ASP.NET Core MVC app** in Jenkins, you need **two** plugins: **Git** and **MSBuild**.

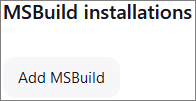
Usually, **Git** is being **installed** when you are **configuring** your **Jenkins** installation and we **already** used it in the previous task.

Let's focus on configuring the **MSBuild** plugin.

Go to **Manage Jenkins** menu and select **Plugins**. From the menu on the left, select **Available plugins** and type **MSBuild** in the search field. Select the plugin and click on the **[Install]** button:



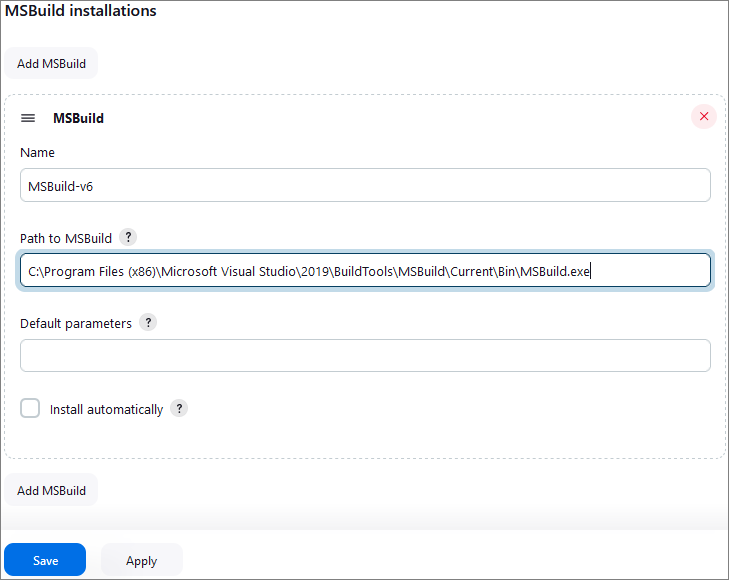
Once you have the needed plugin installed, go back to **Manage** **Jenkins** and select **Tools.** Scroll down to find the **MSBuild installations** section and click on **[Add MSBuild]** button:



Give a **meaningful** **name** to your MSBuild and provide the path to your MSBuild.exe file.

**NOTE: MSBuild.exe** is the **command-line tool** for **Microsoft** **Build** **Engine**, which is used to **build applications**. This engine uses **XML-based** project **files** to **compile** and **link** the **code**, manage **project dependencies**, and **execute** other **build tasks**. It's a vital **component** of the **.NET framework** **development** **process** and is also used in building software projects in other languages. **MSBuild** comes **included** with several **Microsoft** products, including **Visual** **Studio**. Usually, the path to your MSBuild.exe file is something like **C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\MSBuild\Current\Bin\MSBuild.exe**.

The configuration should look like the image below:



Finally, click on the **[Save]** button.

### Step 4: Create and Configure a New Job

Open the **Jenkins** **interface** in a **web browser**.

Create a new job by selecting **[New Item]** from the **Jenkins dashboard**. Choose **Pipeline** and give it a **meaningful** name, after that click on the **[OK]** button.

Next, on the **General** section, type in a proper description.

Select **GitHub project** as the **Source Code Management** option and input the **URL** of your **repository**.

If you want, you can play around a little bit and add a **build** **trigger**, as you already know how to do that.

Go to the **Pipeline** section and select **Pipeline script from SCM** as you already know this is the **best** **practice** for where to keep the **Jenkinsfiles**. Configure the path to the repository and to the Jenkinsfile. The steps are the same as in the previous task.

### Step 5: Create the Jenkinsfile

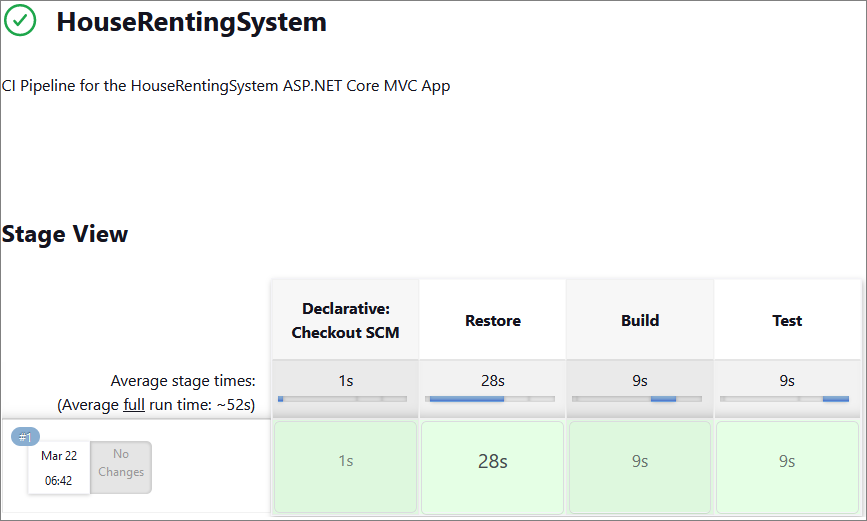
The Jenkinsfile should contain **steps** for:

* **Restore**
  + Restore the NuGet packages needed for the project to ensure all dependencies are downloaded and available during the build process.
* **Build**
  + Build the project to check for compilation errors.
* **Test**
  + Execute the tests to ensure that they're running properly

**Create** your **file** and **upload** it to your GitHub **repository**, containing the code for the application.

### Step 6: Test the CI Pipeline

After completing those steps, we are ready with the CI pipeline and it's time to test if it's working as expected.

****

First, click on the **Build Now** option to start a new build manually (in case you haven't configured the build triggers).

You can monitor the build progress by clicking on the build number and then **Console Output**.