

0 - Setup

In this chapter, we will set up the development environment for the project. We'll walk you through the steps to install the necessary tools and libraries, so you can start working on the project in the next chapter ([1 - Svelte](#)).

Chapter overview

Learning goals

At the end of this chapter, you will be able to:

- (For Windows users:) Install Windows Subsystem for Linux (WSL) on your Windows machine.
- Setup a GitHub repository, and authenticate your device to clone and push code to it using SSH.
- Install Node.js and npm on your WSL environment.
- Setup a local MySQL database using XAMPP (or more specifically, LAMPP)
- Create a new Svelte project using the CLI.

Learning Resources

- [Windows Subsystem for Linux \(WSL\) Installation Guide](#)
- [GitHub SSH Authentication Guide](#)
- [Node.js Installation Guide](#)
- [XAMPP / LAMPP](#)
- [Svelte documentation](#)
- [Command Line Crash Course](#)
- [Codecademy's Command Line Course](#)
- [The Coding Train's Video on the Shell](#)

Installing WSL

If you are already using a **Linux-based OS**, you can skip this section. **macOS users** can continue, but should be aware that this tutorial is written with Linux in mind, so some commands and tools may differ).

If you are using **Windows**, it's time to get comfortable with a Unix-based environment, as that is what most of the web runs on. We will give you a bird's eye view on how you can install the

Windows Subsystem for Linux (WSL) to run a Linux distribution on your Windows machine.

⚠ It is out of the scope of this tutorial to cover interacting with WSL in detail. We will assume you are familiar with the basics of using the shell in a Unix environment. If you are not, then we recommend you to check out the [Command Line Crash Course](#) by Mozilla, or a similar resource, such as [Codecademy's Command Line Course](#), or, if you are more of a visual learner, [this beautiful video](#) by the Coding Train.

i I would recommend you use [Windows Terminal](#) or [Wezterm](#) as your terminal emulator, but don't worry about this too much: it all comes down to personal preference, and any terminal emulator should be fine!

Installation Steps

Always refer to the [latest instructions](#) on the official Microsoft documentation, but here are the general steps to install WSL:

1. **Open CMD or PowerShell as an Administrator:** Search for "cmd" or "PowerShell" in the Start menu, right-click, and select "Run as administrator".

2. **Install WSL:**

Run the following command:

```
wsl --install
```

This will install the default WSL distribution (usually Ubuntu).

3. Restart your computer when prompted.
4. To confirm things worked: try opening WSL by typing `wsl` in the command prompt or PowerShell. This should start a new shell session in your installed Linux distribution.


i You might be prompted to create a new user and password for your Linux environment. Use a username and password that you will remember, as you *will* need them later. You'll be entering this password quite a lot, so make sure it's [something strong, but memorable](#).

5. It's a good idea to update your package list and upgrade the installed packages. Run the following commands in your WSL shell:

```
sudo apt update && sudo apt upgrade -y
```

This will ensure you have the latest updates for your Linux distribution.

`sudo apt update` updates the package list using the `apt` package manager (the default package manager for Ubuntu and other Debian-based distributions). The `&&` operator allows us to chain commands together, and `sudo apt upgrade -y` upgrades all installed packages to their latest versions, with the `-y` flag automatically confirming any prompts.

 If you have no clue what `sudo` does here, it's probably a good idea to first look at the terminal basics courses listed above.

Cool. From now on, any further commands you see in this tutorial will be run in the WSL shell. Try not to get it confused with the Windows command prompt or PowerShell. Should you close your terminal window, you will have to reopen WSL by typing `wsl`.

Finding your Windows files in WSL, and vice versa.

WSL -> Windows

You can access your **Windows files from WSL** by navigating via the `/mnt/` directory. Try listing the contents of your `/mnt/` directory:

```
ls /mnt/
```

You'll probably see directories like `c`, `d`, etc., which correspond to your Windows drives. For example, your C: drive is located at `/mnt/c/`. A bit of tab-autocompletion does magic here.

Windows -> WSL

Should you want to instead access your **WSL** files from Windows, you can do so by navigating to the special URI `\\wsl$` in Windows Explorer.

Your WSL home directory is located at `\\wsl$\\Ubuntu\\home\\<your-username>` (replace `Ubuntu` with the name of your WSL distribution, if different). You can also access other directories in your WSL file system from Windows Explorer.

Editing WSL files from Windows (and vice versa)

Preferably, you should edit files in your WSL environment directly using WSL, and your Windows files directly using Windows. Mixing the two can be done, but it can lead to issues with performance and file permissions.


The first recommendation would be to use a terminal-based editor like `neovim` ([link](#)) to edit WSL files directly from the WSL shell. (In fact, I am going to take this opportunity to *highly* recommend you to learn to use `neovim` as your primary text editor (*or at the very least* [VIM motions](#) which can also be used in tons of other editors you already know). But truth be told,

there's a pretty steep learning curve, so please do seriously consider this recommendation, but it might be best to postpone that for later.)

Another solid (although technically rule-breaking) option is to run a graphical code editor like [Visual Studio Code](#) or [JetBrains IDEs](#) from Windows, and open your project directory (in the WSL directory) directly in those editors, using the WSL path listed above. Again, this is *technically* discouraged, but it works well enough for this tutorial.

Setting up Git and GitHub

In this section, we will set up git on our machine, setup a repository on GitHub, and authenticate our device to GitHub using SSH. This will allow us to clone and push code to our GitHub repositories.

 If you already installed and set up Git on your Windows machine, you should still set it up in WSL!

Install Git


Git comes pre-installed on most Linux distributions, but let's make sure you have it installed. Open your terminal and run the following command:

```
git --version
```

- If Git is installed, this command will return the version number of Git. [See if you have the latest version](#). If not, it's probably a good idea to update Git at this point.
- If it is not installed, the command will return something along the lines of `command not found: git`.

To update or install git, run:

```
sudo apt update && sudo apt install git
```

 If you are using a non-debian based Linux distribution or MacOS, the package manager and installation command may differ. Refer to your distribution's documentation for the correct command.


Now that Git is installed, you should be able to get some output from `which git`, or run `git --version` to see the installed version of Git.

Configuring Git

All right, now that we have Git installed, let's configure it with your name and email address. This is important because Git uses this information to identify who made changes to the code.

Run the following commands in your terminal:

```
git config --global user.name "<My Name>"
git config --global user.email "<my@email.com>"
```


 Replace `<My Name>` and `<my@email.com>` with your name and email address (If you've only worked with Git GUI's before, you probably use the same e-mail as you log in to GitHub with).

Now that Git knows who we are, we can move on to setting up a GitHub repository.

Setting up a GitHub repository

For this project, we just want an empty repository to start with.

- ☐ Name it `svelte-db-portal` (feel free to come up with your own memorable name).
- ☐ Give it a succinct description, such as "A Svelte-based web application for visualizing data from a MySQL database".
- ☐ Set it to `private` for now, so only you can access it.
- ☐ Do not initialize it with a README, `.gitignore`, or license file, as we will do that later.

 Refer to the [GitHub documentation](#) for more information on how to create a new repository.

Cool, we now have an empty repository which we can use to version control our code.

We can try cloning it to our local machine by clicking the green "Code" button on the repository page, and copying the **SSH URL**. Now, we can clone it to any directory on our local machine. Let's create a working directory under our home directory, and clone the repository there.

```
# Create a directory called 'work' in our home directory
mkdir -p ~/work
# Change to the 'work' directory
cd work
# Clone the repository using SSH
git clone git@github.com:myusername/myreponame.git
```

Oops! At this point, you might get an error, saying something like `Permission denied`. We haven't set up our machine to authenticate with GitHub just yet, so we can't access our repository. Let's fix that in the next section.


Authenticating with GitHub using SSH

Now that we have a repository, we need to authenticate our device to GitHub, so our device is allowed to clone and push code to the repository. We will do this using SSH, which is a secure way to connect to remote servers.

To authenticate with GitHub using SSH, refer to the latest instructions in the [GitHub SSH Authentication Guide](#). Specifically, the [Generating a new SSH key and adding it to the ssh-agent](#) and [Adding a new SSH key to your GitHub account](#) sections are most relevant.

Although the instructions are quite detailed (and may change over time), here is a quick summary of the steps you need to take as of May 2025:


- ☐ Generate a new SSH key: `ssh-keygen -t ed25519 -C "my@email.com"`
 - Press Enter to accept the default file location.
 - Press Enter again to leave the passphrase empty (or set a passphrase if you prefer).
 - Press Enter to confirm the passphrase (if you set one).
- ☐ Start the SSH agent: `eval "$(ssh-agent -s)"`
 - It should return something like `Agent pid 1234`.
- ☐ Add your SSH key to the SSH agent: `ssh-add ~/.ssh/id_ed25519` (or the file you specified in step 1).
- ☐ View your public key: `cat ~/.ssh/id_ed25519.pub`
 - Copy the output of this command to your clipboard.
- ☐ Add the public key to your GitHub account:
 - Go to your Settings page on GitHub.
 - In the left sidebar, click on "SSH and GPG keys".
 - Click on "New SSH key".
 - Give it a title (e.g., "My Laptop SSH Key") and paste the public key you copied earlier into the "Key" field.
 - Click "Add SSH key" to save it.

 Terminal emulators usually have some way to copy text other than CTRL+C, as that is reserved for sending a interrupt signal to the process running in the terminal. For example, you can use CTRL+SHIFT+C in most terminal emulators to copy text, or highlight the text with your mouse and right-click.

Now that we have authenticated our device with GitHub, we can try cloning the repository again:

```
# Change to the 'work' directory (or wherever you want to clone the repository)
```

```
cd ~/work
# Clone the repository using SSH
git clone git@github.com:myusername/myreponame.git
# List the contents of the current directory to see if the repository was
cloned successfully
ls
# Move into the cloned repository
cd myreponame
```

 You might need to confirm the authenticity of the host (github.com) by typing `yes` when prompted. This is a security measure to ensure you are connecting to the correct server.

Congratulations! You have successfully cloned your GitHub repository to your local machine. You can now start working on your project and push changes back to GitHub.


Installing Node.js and npm

Okay, we now have a GitHub repository set up, and we can push code to it. We just need to set up a few more things before we can get to the real work. To run our Svelte application, we will use **Node.js** and **npm** (Node Package Manager).

- **Node.js** is a JavaScript runtime that allows us to run JavaScript code outside of a web browser.
- **npm** is a package manager for Node.js that allows us to install and manage libraries and tools that we will use in our project.
- Node and npm are usually bundled together, so when you install Node.js, you also get npm installed.

The easiest way to install Node.js and npm is to use the **Node Version Manager (nvm)**, which allows you to easily install and manage multiple versions of Node.js on your machine.

Refer to the [Node.js installation guide](#) for the latest instructions.

 First-time WSL users should be aware that you should follow the Linux instructions, not the Windows instructions.


Installing XAMPP (or LAMPP)

With Git and Node.js set up, we now need to install XAMPP (or more accurately, LAMPP, which is the Linux version of XAMPP).


XAMPP is a free and open-source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB (or MySQL), and interpreters for scripts written in the PHP and Perl programming languages.

We will use XAMPP to run a local MySQL database, and view and alter our database using a GUI tool called phpMyAdmin.

Note that technically, you could use any MySQL server for this project, but XAMPP is a popular choice for beginners, as it is easy to set up and use.

 For those that want to be in the know: Technically, we'll be installing MariaDB, which is not exactly the same as MySQL, but it functions as a drop-in replacement for MySQL, so we'll refer to it as MySQL throughout this tutorial for simplicity and familiarity, just like XAMPP does.

To install XAMPP, download the latest version for Linux from the [Apache Friends website](#).

 WSL users: Use `wget` if you know how that works. If not, you can download the installer using your web browser, and then move it from your Windows file system to your WSL directory using the `mv` command. You can access the Windows file system from WSL by navigating to `/mnt/c/` for the C: drive. Thus, if you downloaded the installer to your Downloads folder in Windows, you can move it to your WSL home directory with commands similar to the following:


```
mkdir -p ~/Downloads
mv /mnt/c/Users/MyUsername/Downloads/xampp-linux-*installer.run ~/Downloads/
# Note that if you have spaces in your path (such as in the username), you
# need to wrap the path in quotes, or escape them with a backslash (\).
# Autocompleting the path using tab is handy here.
```

Now that we have download the installer, we need to run it. But to do that, we first need to alter the file permissions so we are allowed to run it. We can do this using `chmod` :

```
# Change to the directory where the installer is located
cd ~/Downloads
chmod +x ./xampp-linux-*installer.run    # +x means 'add execute permission'
```

With the execute permission set, we can now run the installer:

```
sudo ./xampp-linux-*installer.run
```

 This installation may take a while, so please be patient.

Verify that the installation was successful by running the following command:

```
sudo /opt/lampp/lampp status
```


This should return the status of the XAMPP services, such as Apache and MySQL. If everything is running, you should see something like:

```
Version: XAMPP for Linux 8.1.12-0
Apache is not running.
MySQL is not running.
ProFTPD is not running.
```

We can attempt to start the services by running:

```
sudo /opt/lampp/lampp start
```

Awesome - looks like we're almost done with the setup! We just need to create our Svelte project.

Creating a new Svelte project

To create our Svelte app, we will use SvelteKit. In the next chapter, we will be going into more details on why we are using Svelte(Kit), but for now, let's just set up the project.

First, let's head over to our repository.

```
cd ~/work/svelte-db-portal
```


Now, we will create our Svelte app using the sv CLI, which we can run using npx (which comes with npm. npx allows us to run commands from npm packages without installing them globally):

```
npx sv create app    # 'app' is just the name of our app. Feel free to
change it to something else.
```

You'll be prompted with a few options.

- ☐ Choose a template: Select Skeleton project for a minimal setup.
- ☐ Use TypeScript syntax: Choose Yes, as we will be using TypeScript in this project.
- ☐ For plugins, check:
 - ☐ ESLint, a linter for JavaScript and TypeScript code.
 - ☐ Prettier, an opinionated code formatter.
 - ☐ Tailwind CSS, a utility-first CSS framework.
 - ☐ Drizzle, an ORM for TypeScript and JavaScript.

- ☐ Add no tailwind plugins for now.
- ☐ Choose the MySQL database for Drizzle.
- ☐ Choose the mysql2 driver for Drizzle.
- ☐ Do not run the database with docker-compose (as we'll be using XAMPP).
- ☐ Use `npm` to install the Drizzle dependencies.

 If you are not familiar with any of these tools, don't worry! We will explain them in more detail later. For now, just select the options as described above.

Your log should look something like this:

```
11:55:52 › npx sv create my-app
Need to install the following packages:
sv@0.8.7
Ok to proceed? (y) y

Welcome to the Svelte CLI! (v0.8.7)

◇ Which template would you like?
  SvelteKit minimal

◇ Add type checking with TypeScript?
  Yes, using TypeScript syntax

◆ Project created

◇ What would you like to add to your project? (use arrow keys / space bar)
  prettier, eslint, tailwindcss, drizzle

◇ tailwindcss: Which plugins would you like to add?
  none

◇ drizzle: Which database would you like to use?
  MySQL

◇ drizzle: Which MySQL client would you like to use?
  mysql2

◇ drizzle: Do you want to run the database locally with docker-compose?
  No

◆ Successfully setup add-ons

◇ Which package manager do you want to install dependencies with?
  npm

◆ Successfully installed dependencies

◇ Successfully formatted modified files

◇ Project next steps
  1: cd my-app
  2: git init && git add -A && git commit -m "Initial commit" (optional)
  3: npm run dev -- --open
```

Verify that everything was installed correctly by running the Svelte development server:

```
cd my-app
npm i # Install the dependencies (should be done by sv, but just to be sure)
npm run dev # Start the development server
```

Now visit <http://localhost:5173> in your web browser. You should see a Svelte welcome page, which means everything is set up correctly!

Terminate the development server with `CTRL+C`, Go up one directory (to the root of your repository), and commit our new project:

```
cd ..
git add .
git commit -m "Init Svelte project"
git push origin main
```

Congratulations! You have successfully set up your first Svelte app.

Editor plugins

To make your development experience smoother, we recommend installing some plugins for your code editor.

If you use a different code editor than the ones listed below, try searching for similar plugins that provide Svelte, ESLint, Prettier, and Tailwind CSS support.

- **VSCode:**
 - [Svelte for VSCode](#): Svelte language support.
 - [ESLint](#): Linting support for JavaScript and TypeScript.
 - [Prettier - Code formatter](#): Code formatting support.
 - [Tailwind CSS IntelliSense](#): Tailwind CSS support.
- **Neovim:**
 - (A good starting point for Neovim configuration is [kickstart-modular](#))
 - Install the following language servers (e.g. using `lspconfig` from the kickstart example):
 - `eslint-lsp`
 - `prettier`
 - `svelte-language-server`
 - `tailwindcss-language-server`
 - `typescript-language-server`

Summary

To wrap up, we have:

- Installed WSL on our Windows machine (or used a Linux-based OS).
- Set up Git and GitHub, and authenticated our device to GitHub using SSH.
- Installed Node.js and npm using nvm.
- Installed XAMPP (or LAMPP) to run a local MySQL database.
- Created a new Svelte project using the Svelte CLI.
- Pushed our initial project to GitHub.
- Installed some handy editor plugins to improve our development experience.

In the [next chapter](#), we will start building the basic structure for our Svelte application.