

Open in app ↗

Sign up

Sign In



Search Medium



Published in FAUN Publication



Akhilraj Nambiar

Follow

Sep 5 · 7 min read · Listen



Save



Deploy a FastAPI website to Railway

Hi all, this post aims to achieve the goal of deploying a full scale api, built using [FastAPI](#). With slight modifications you could be able to do this **Flask** and **Django** as well.

Why Railway:

Well Heroku is no longer going to allow free deployment as of November 28th 2022. For first timers, Railway seems to have become the best bet now. You can find more about Railway over [here](#).

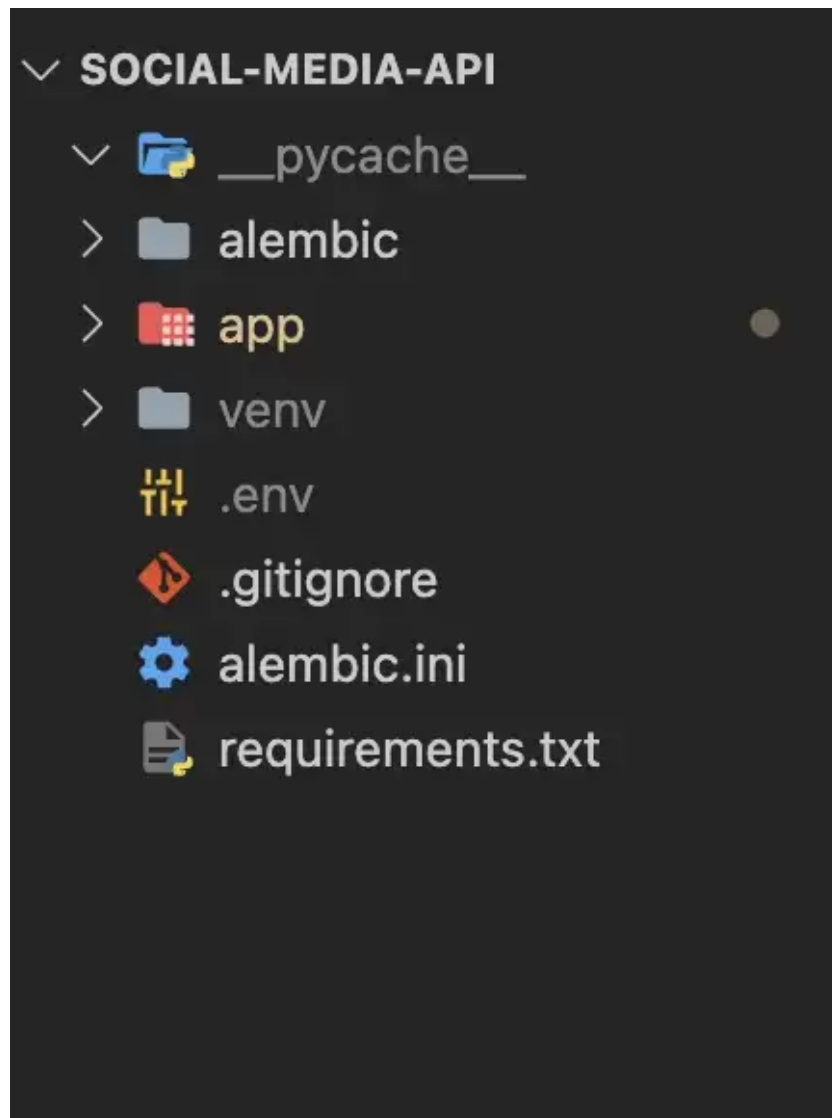
Prerequisites:

Before reaching this step you must a working FastAPI application for the development environment. You can learn how to create one over [here](#). You could also be using any other Python web framework like Flask or Django.

Have knowledge of using Git!

Project Overview:

This project is created as per this [video](#), and has the following file structure on completion.



File structure of this FastAPI project

This project creates a blog type social media app. The root directory contains various folders that contain code specific to them. I have used a **postgres database** for storing the info about users, posts, etc, and **SqlAlchemy** as the ORM.

All the project code is situated in the **app directory**. Be in the database specific part, the schemas, or the routes.

The **venv directory** is responsible for managing the virtual environment and the environment-specific dependencies.

The **alembic directory** and **alembic.ini** file are used to manage the database migrations.

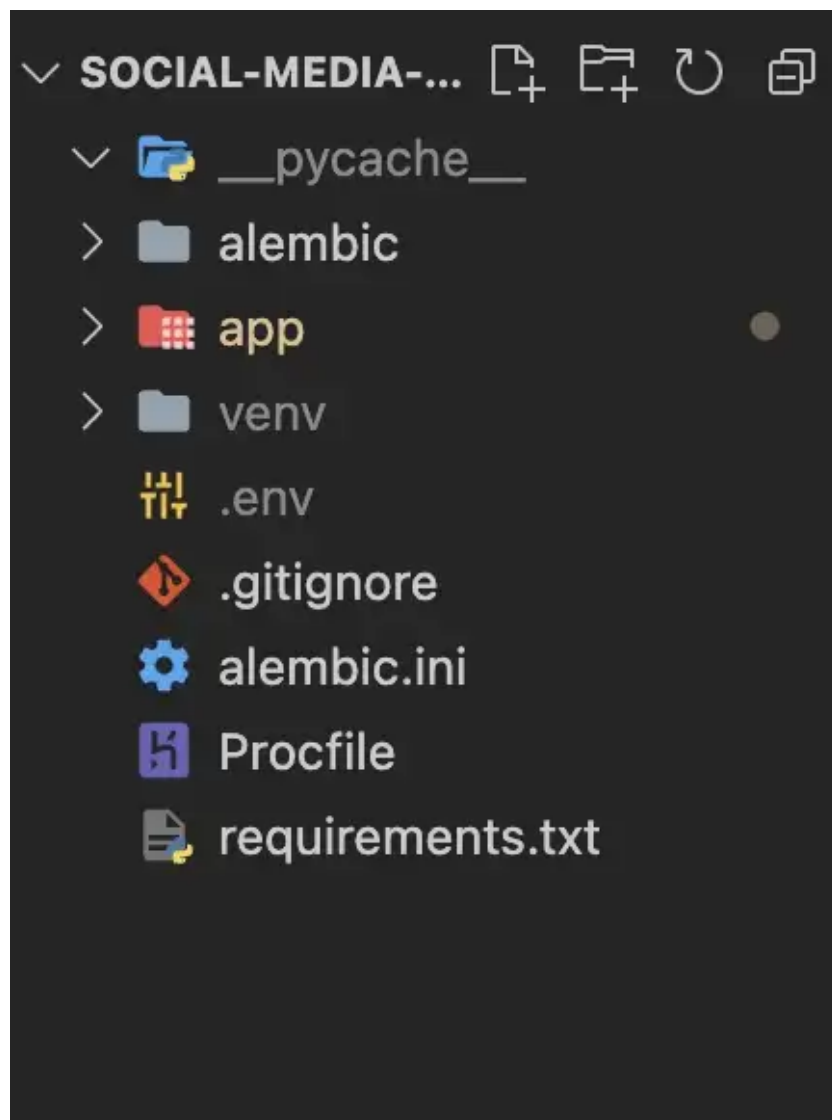
The **.env file** contains the environment variables of this project. This is not added to Git.

The **.gitignore** file as the name suggests, holds the files and directories that you don't want to add to git.

The **requirements.txt** file contains the dependencies as present in the virtual environment.

Create a Procfile:

Well, for all those who have deployed apps using Heroku, this name should sound familiar. Basically you just have a file that runs the execution command that you call in the command line during development, in the Railway production environment. In the root directory of your app, create a Procfile without any extension.



Procfile added to the file structure in VS Code

Now for running the development server we use the following command in FastAPI:

```
uvicorn app.main:app
```

I use app.main to start as my main.py file is situated within the app directory. Based on your file structure, the command will differ. We add this same command with some flags and a keyword in the Procfile.

```
web: uvicorn app.main:app --host=0.0.0.0 --port=${PORT:-5000}
```

This command is pretty obvious once you get what it means. First we indicate that we want to start a **web server** using the **web** keyword. Then comes the command that we run as usual. Setting the host flag as 0.0.0.0 indicates that we are open to receiving requests from any IP address. Then we set the port to be equal to the port provided by Railway to us. Hence we use the PORT variable. If we don't provide this flag the app will by default run on port 8000.

Saving your code to Git:

Before doing this be sure to add the modules required for the current working virtual environment to a requirements.txt file, since we won't commit the venv folder to git. Use the following command:

```
pip freeze > requirements.txt
```

Next we deploy our code to git. If you already have a repository created then, simply commit the current changes to git. Else, initialize and commit to a new git repository using the following commands:

```
git init
git add .
git commit -m "Current changes"
```

Create a new repository on GitHub, and set it as the remote origin for this repository.

```
git remote add origin <Your github repository URL>
```

Now push the committed changes to GitHub using the following command:

```
git push origin master
```

Deploying to Railway:

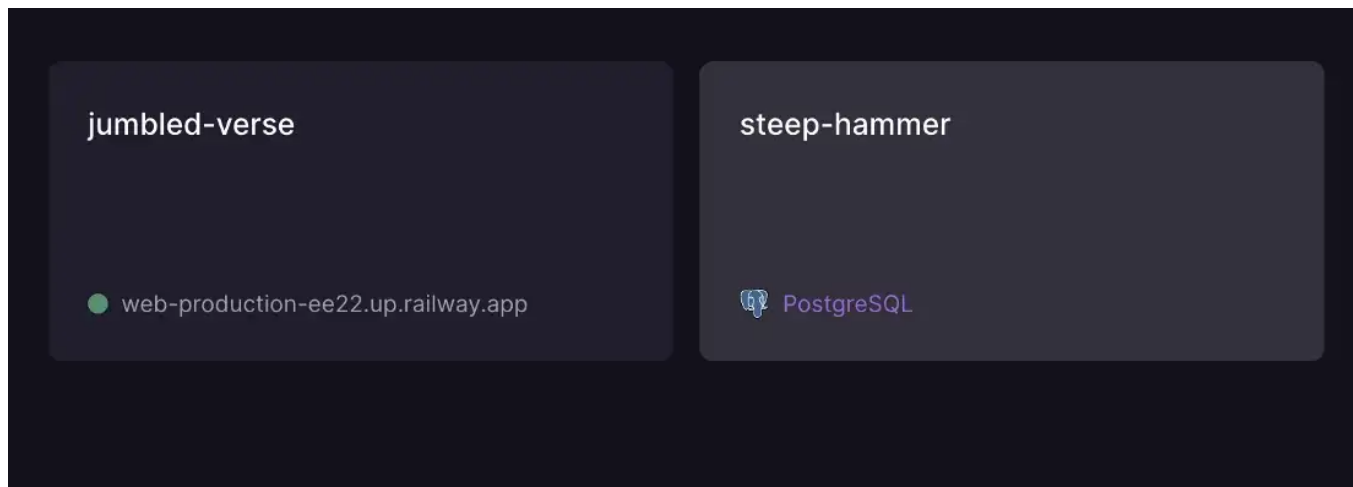
Now that the setup is ready, let's deploy the project to Railway. For that we need to create a Postgres instance on Railway. Let's do that at first.

1. Login to Railway if you haven't already.
2. Go to your dashboard and click on New Project Button.
3. Select the option **Provision Postgres**.
4. Your postgres db would be live on Railway soon.

Let's add our project now from Git to Railway. The process is fairly similar.

1. After logging in to Railway, go to your dashboard and click on New Project Button.
2. Select the option **Deploy from GitHub repo**.
3. Provide Railway app to access your GitHub repositories.
4. Soon, your project should be live.

This is how your dashboard should look like:



The fastAPI project and the Postgres database on Railway dashboard

Psycopg2 issue:

But the build was not successful the first time. If you guys have have used sqlalchemy before you probably must have run into this issue as well:

```
Error: pg_config executable not found.
```

```
pg_config is required to build psycopg2 from source.
```

This error comes up in both Linux and Mac machines. This is because of a module called **psycopg2** — a python driver for Postgres databases. This module is required by SQLAlchemy. To solve this issue just do the follwing steps:

- Uninstall psycopg2 from the current virtual environment of your project:

```
pip uninstall psycopg2
```

- Once it is uninstalled completely, check the modules in your current environment to see if there are any other versions of psycopg2 installed

```
pip freeze
```

- Now all you have to do is install **psycopg2-binary** using the following command.

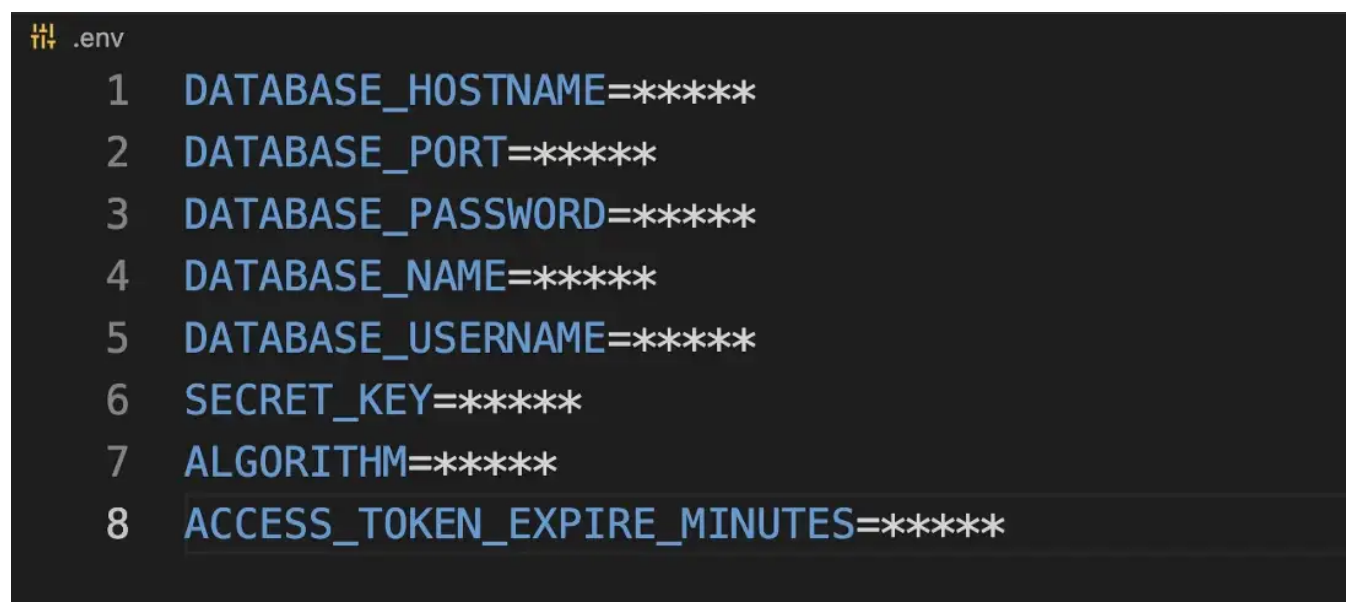
```
pip install psycopg2-binary
```

- Build the project and make sure there are no bugs. Once that's sorted refresh your requirements.txt file with the new module.
- Commit and Push your changes to GitHub.
- Railway will automatically rebuild the updated code without any build errors!

Environment Variables issue:

This issue should definitely popup on everybody's deployment. If it does not, it means there is some issue with your code 😊. The error is basically that it cannot find the values of all these keys in our code. It shouldn't be able to, since I did not commit my .env file to Git.

Here is what my .env file looks like (values are hidden 🙈)



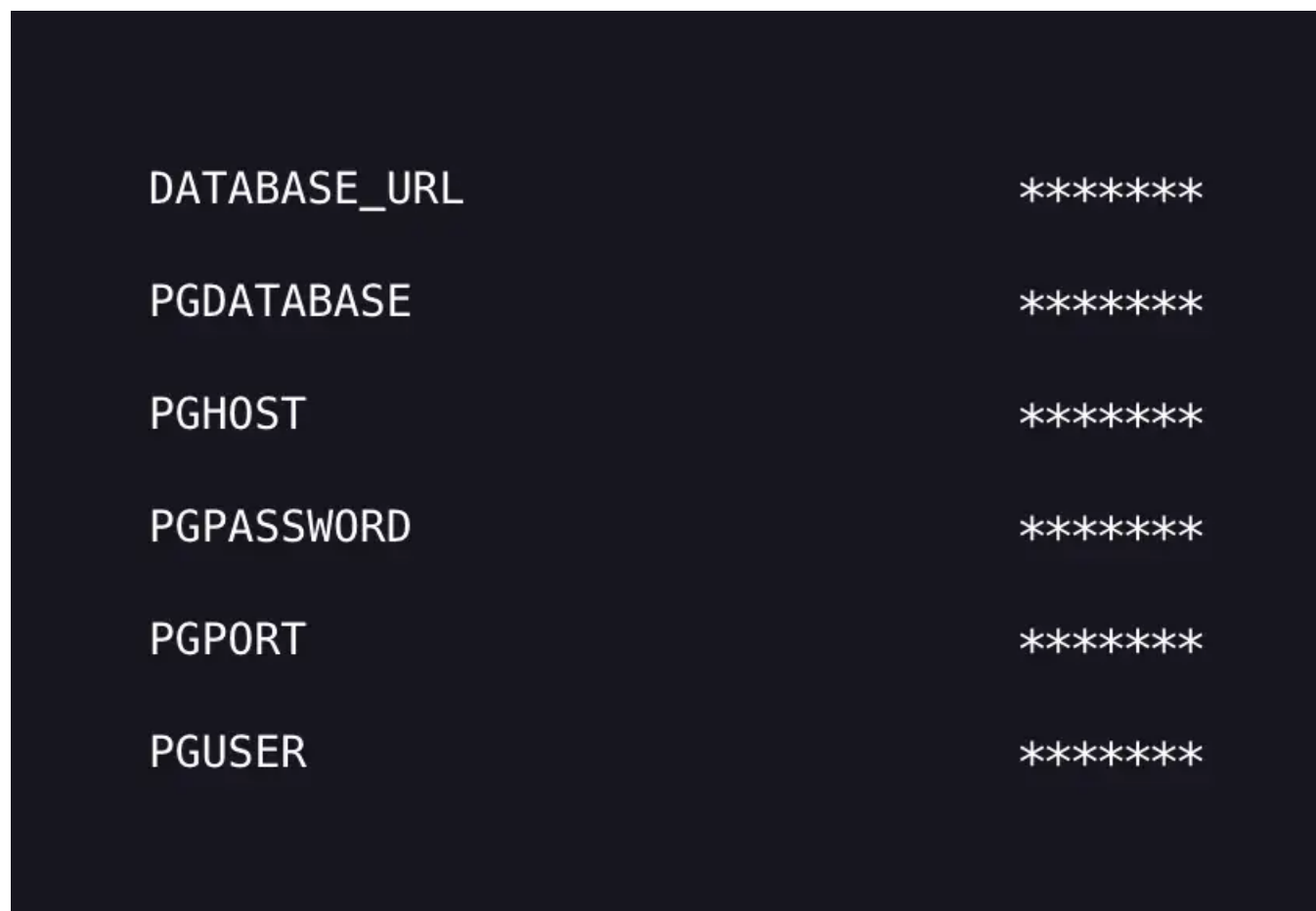
```
1 DATABASE_HOSTNAME=*****
2 DATABASE_PORT=*****
3 DATABASE_PASSWORD=*****
4 DATABASE_NAME=*****
5 DATABASE_USERNAME=*****
6 SECRET_KEY=*****
7 ALGORITHM=*****
8 ACCESS_TOKEN_EXPIRE_MINUTES=*****
```

My environment variable keys, as required by the project in the .env file

In the development environment, the values were mostly the local db setup values, as created through PgAdmin 4. However for the production, we have a Postgres db deployed on Railway already. Let's use the info from this db and some features provided

by Railway to fix this issue.

- First identify the variables from the deployed Postgres db. For that go to your dashboard, click on the db project (steep-hammer in my case), then select the Postgresql card.
- Then from the various options like Data, Connect, Metrics, etc. select the **Variables** tab.
- You should be able to find all the configurations of this db over here, like this:



The config keys of the database deployed on Railway

As you can see, this is the information that we do need for our environment variables as well. All we have to now do is map this info to our FastAPI project deployed on Railway. For this follow the below steps.

- Select your FastAPI project (jumbled-verse in my case), and then select your project card.

- Then from the various options like Data, Connect, Metrics, etc. select the **Variables** tab.
- Set the environment variables like so:

```
ACCESS_TOKEN_EXPIRE_MINUTES    *****  
  
ALGORITHM                      *****  
  
DATABASE_HOSTNAME              *****  
  
DATABASE_NAME                  *****  
  
DATABASE_PASSWORD              *****  
  
DATABASE_PORT                  *****  
  
DATABASE_USERNAME              *****  
  
SECRET_KEY                     *****
```

Setting the environment variables on the Railway

The mappings will be as follows:

1. PGHOST -> DATABASE_HOSTNAME
2. PGDATABASE -> DATABASE_NAME
3. PGPORT -> DATABASE_PORT, and so on.

The non-database variables were assigned by me. Nothing complicated in that.

Now your program should run successfully, right?

Tables not found issue:

One final issue that remains is the tables not being created. You would possibly get that error when doing something related to the db. To solve this, we need install the Railway console in our project and write some code!

Install the railway console by going to VS Code terminal and following [this](#). In my case:

```
brew install railway
```

Then login to and link your project using the following commands:

```
railway login  
railway link <projectId>
```

If you don't know the project id, just type **railway link** and press enter. Your available projects should show up with their Id.

Now run this following command using railway like so:

```
railway run alembic upgrade head
```

This command calls on alembic to implement all the database migrations up to the head of the migration chart. Once, the execution is finished you should be able to see your tables in the **Data tab** of the Postgres db deployed on Railway.

And that's it. Your project has been completely deployed on Railway and is easily available for public use!



14



1

That's it from me. Thank you for reading till the last line. Appreciate it 🥳.



If this post was helpful, please click the clap 🙌 button below a few times to show your support

for the author 👉

Sign up for FAUN and grow by keeping up with what matters, JOIN FAUN.

By FAUN Publication

Medium's largest and most followed independent DevOps publication. Join thousands of developers and DevOps enthusiasts. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.



Get this newsletter

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

