



```
29  useEffect(() => {
30    const setInitialImagesQty = async (userId) => {
31      const data = await getUserImagesQuantity(userId);
32      setStateData({ ...stateData, userImagesQuantity: data });
33    };
34    if (isAuthenticated && user) {
35      setInitialImagesQty(user.id);
36    }
37    if (!isAuthenticated) {
38      setStateData({});
39    }
40    }, [isAuthenticated, user]);
41
42    return (
43      <AppContext.Provider value={{ auth, stateData, ... }}>
44        <Router>
45          <div className="App">
```

STASHCHUK.COM

How-to setup guide

Guide for setting up and running course project application “**Images Gallery**” locally

By **Bogdan Stashchuk**



OUTLINE

OUTLINE	2
Course project description.....	3
Required software.....	4
Project files.....	5
How to move to the specific project version	7
Installing NPM dependencies in the frontend application.....	9
Initializing Python Virtual Environment and installing dependencies for the backend application	11
Creating files with environment variables	13
Running Images Gallery application without Docker	15
Running Images Gallery application with Docker	16
Managing services which are running using Docker	17
Launching frontend application and sending requests to the backend API.....	18



Course project description

In the course you will build one application called “**Images Gallery**”. Users of this application are able to search for images and save them in the database.

Image search involves Unsplash API which gives ability to search large library of the free images. You could use such images for free but should include link to the profile of the author who created corresponding image.

Following information about the images will be saved in the Mongo database:

- Title
- Description
- Author
- Link to the actual image at unsplash.com

Users of the frontend application will be able to retrieve previously found images from the database.

Course project application consists of the following services:

- Frontend React Application
- Backend Python Flask
- MongoDB database

This document contains details and instructions about the project setup on your local Windows or Mac computer.

After setting the local development environment, you will be able to run both frontend and backend applications locally and develop the course project yourself.

IMPORTANT NOTE: *You could continue development from the specific moment in the course. Some lectures have project snapshots attached.*

Just unzip, perform setup as described in this document and proceed with the development





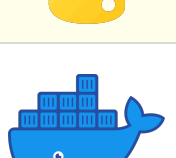



Required software

For the development on your local computer you need to have several programs installed. You could install all of them at the beginning of the course or as you progress through the course sections.

When it becomes necessary I will tell you which additional software you need and how to install it.

Software required for the development

Visual Studio Code Free IDE developed by Microsoft https://code.visualstudio.com/	
Git Distributed source control system https://git-scm.com/	
Node.js with NPM Node.js is JavaScript runtime environment NPM is package manager for Node.js applications https://nodejs.org/	
Python with PIP Python is high-level programming language https://www.python.org/	
Docker Docker is an application build, run and deployment tool https://www.docker.com/	
Postman API platform for building and using APIs https://www.postman.com/	



Project files

Course project files are available in the public GitHub repository. You could clone this GitHub repository to your local computer. It has single branch called `main`. If you need to jump to specific moment in the development history, you could checkout specific commit.

Course Project GitHub Repository

*Start by cloning course repository to your local computer.
You could use it for reference or in case you want to continue
development from the specific moment*

github.com/bstashchuk/images-gallery



How to clone remote GitHub repository

1. Install Git from git-scm.com
2. Open any terminal application. I suggest to use following applications
 - **iTerm2** on Mac
 - **PowerShell** or **Git Bash** on Windows
3. **IMPORTANT** (ONLY for Windows users). Disable automatic conversion of the **LF** line endings in the text files to **CRLF** by Git during cloning process

```
git config --global core.autocrlf input
```

4. Change directory to the Desktop or other folder where you want to create project folder

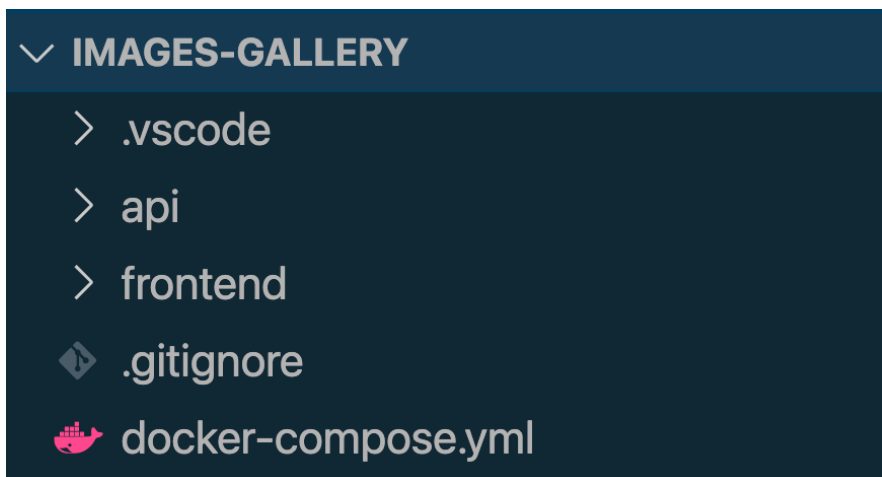
```
cd Desktop
```



5. Clone remote GitHub repository.

```
git clone https://github.com/bstashchuk/images-gallery
```

6. During cloning Git will create new folder titled `images-gallery`
7. Install **Visual Studio Code** editor which you could download from visualstudio.microsoft.com.
8. After installation launch VS Code application and open `images-gallery` folder. In order to open folder go to **File -> Open** on Mac or **File -> Open Folder** on Windows. Find and select entire `images-gallery` folder and open it. You should see entire **Images Gallery** project opened in the VS Code



9. You are **done** with cloning of the course GitHub repository! 👍

NOTE: I suggest you to **keep cloned repository only for reference** and do development yourself. For that you could create separate folder `images-gallery-my` and open it in another window in the VS Code.

```
mkdir images-gallery-my
```



How to move to the specific project version

During development of the **Images Gallery** course project I committed changes using **Git** at specific moments after reaching specific milestones. You could “jump” to specific version of the project by checking out specific commit.

How to checkout different commits

1. Clone GitHub repository github.com/bstashchuk/images-gallery as described in the previous section
2. Change directory to the `images-gallery` folder which was created by Git during cloning process

```
cd images-gallery
```

3. List all commits with their SHA1 hashes and commit messages

```
git log --oneline
```

4. You will see such list of commits. *On the screenshot there is only **part** of commits*

```
da0c798 update react-scripts version
dab9224 welcome component
8515f8c added text logo
f23785c display and delete images
8ba15d0 save images in the state
4f16737 adjust vs code tab size
2725817 fixed code with linter
d02a919 eslint and prettier setup
597e459 reset search input upon submit
80036f3 first api call
4dcf62d controlled search component
ab7357d initial header component
a51e5dd changed favicon
2dfcb86 git ignore .eslitcache
5e60337 remove .eslintcache from index
8afcfc4 initial frontend app
```



5. Now select specific commit and copy its SHA1 hash. For example let's take such commit with SHA1 hash `597e459`

```
597e459 reset search input upon submit
```

6. Let's checkout the project at this commit

```
git checkout 597e459
```

7. Now you are at the moment after the commit “**597e459 reset search input upon submit**”.

If you now open the project in the VS Code you will see following folders and files inside (*there is just one folder and one file since it's very early phase in the project development*)

✓ **IMAGES-GALLERY**

> frontend

📁 .gitignore

8. Now you could continue development from this moment. In order not to lose your future development work I suggest you to create new Git branch where all your own commits will be stored.

NOTE: *You could choose any names for your own branches. Also you could create different own branches*

```
git checkout -b my-dev-branch
```

9. Now you could continue development and commit changes same as I do in the lessons.

10. You could *jump* to the last version of the project by checking out `main` branch

```
git checkout main
```

11. Happy coding! 🥳



Installing NPM dependencies in the frontend application

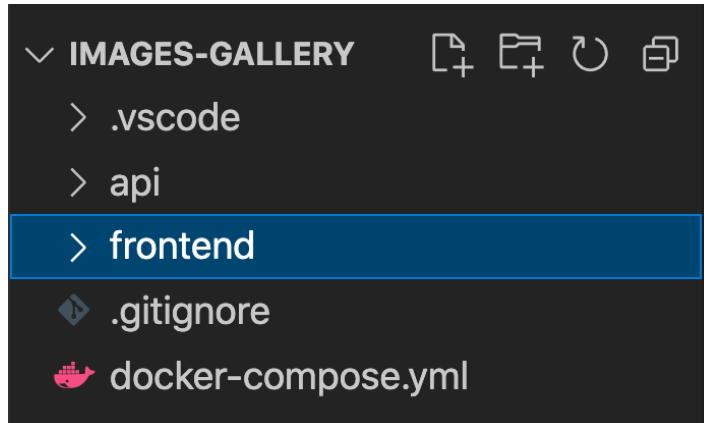
Frontend application is React JavaScript application. In order to run it, you need to have `Node.js` along with `NPM` installed on your computer.

After introduction of the Docker you don't need to have Node.js and NPM to run frontend application because it will be started inside of the Docker container. Corresponding Docker image will have `Node.js`, `NPM` and all necessary frontend application dependencies installed *inside of it* during image build phase.

NOTE: But while using Docker you still need Node.js and NPM to do project development because NPM dependencies are required for such Visual Studio Code feature as **IntelliSense**.

Steps for the installation of the NPM dependencies

1. Course project `images-gallery` has `frontend` subfolder



2. In the Terminal application make sure that you are already inside of the `images-gallery` folder and change directory to the `frontend` folder

```
cd frontend
```



3. Inside of the `frontend` folder there is a `package.json` file which contains list of all frontend application dependencies

▼ frontend	3	"version": "0.1.0",
> public	4	"private": true,
> src	5	"dependencies": {
🚫 .dockerignore	6	"@testing-library/jest-dom": "^4.2.4",
🔒 .gitignore	7	"@testing-library/react": "^9.5.0",
🐳 Dockerfile	8	"@testing-library/user-event": "^7.2.1",
📄 package-lock.json	9	"axios": "^0.20.0",
📄 package.json	10	"bootstrap": "^4.5.3",
📖 README.md	11	"react": "^17.0.0",
	12	"react-bootstrap": "^1.4.0",
	13	"react-dom": "^17.0.0",

4. Install all NPM dependencies

```
npm install
```

5. You are **done** with the setup of the `frontend` part! 👍



Initializing Python Virtual Environment and installing dependencies for the backend application

Backend application is Python application. In order to run it, you need to have Python and Pipenv installed on your computer.

After introduction of the Docker, like the frontend application, backend application will run inside of the Docker container and Python along with all necessary Python dependencies will be installed inside of the Docker image.

NOTE: *But while using Docker you still need Python, Pipenv and activated Python virtual environment to do project development because Python dependencies are required for such Visual Studio Code feature as **IntelliSense**.*

Steps for the initialization of the Virtual Environment and installation of the project dependencies such as Python Flask

1. Course project `images-gallery` has `api` subfolder
2. In the Terminal application change directory to the `api` folder

```
cd api
```

3. Inside of the `api` folder create new empty folder `.venv`. In this folder Python will create Virtual Environment related files

```
mkdir .venv
```





4. Install `pipenv` globally on your computer. It is needed for creation of the Virtual Environment

```
pip install --user pipenv
```

NOTE: Mac users should enter following command because both versions Python 2 and Python 3 are usually installed on Mac

```
pip3 install --user pipenv
```

5. Make sure to add path to the `pipenv` to PATH. Refer to the following lectures regarding details how to accomplish that

-  *Installing Python, Pip and Pipenv on MacOS*
-  *Installing Python, Pip and Pipenv on Windows*

6. Make sure you are in the `api` folder. Create virtual environment and install all packages listed in the `Pipfile`

```
pipenv install
```

7. Activate Virtual Environment in the shell

```
pipenv shell
```

8. You are **done** with setup of the backend part! 👍



Creating files with environment variables


Images Gallery application relies on the Unsplash API for the images search. In order to be able to search images using Unsplash API you have to create user account at the unsplash.com and generate **Access Key**.

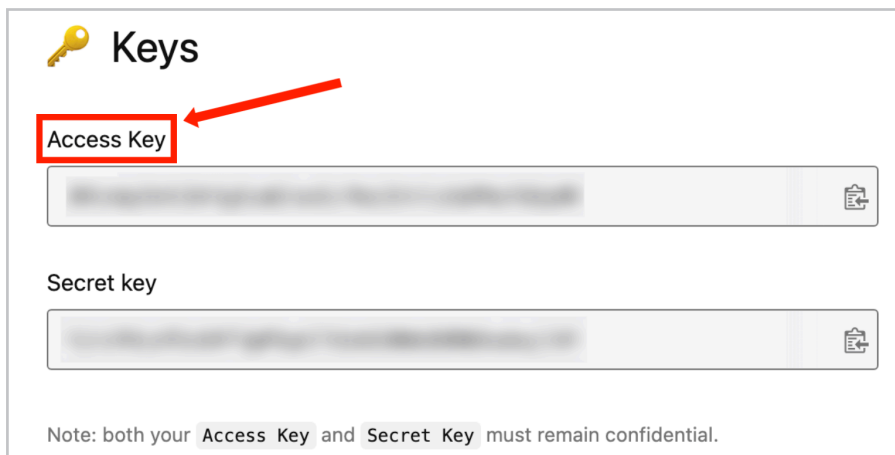
After the generation of the **Access Key** you need to add it to the `images-gallery` application. You should store such sensitive information as keys in the files that will **not** be committed to Git.

Therefore you will insert this key and other sensitive credentials like username and password for the database to the locally significant files `.env.local`. Those files will be ignored by Git (this is specified in the `.gitignore` file).

After start, both `frontend` and the `api` services will read `.env.local` files and create environment variables like **UNSPLASH_KEY**

Steps for the creation of the files with environment variables

1. Create an account at the unsplash.com
2. After registration navigate to the unsplash.com/oauth/applications and create new application. Details how to create new application are available in the lecture
 -  *Creating account at Unsplash and registering new App*
3. After creation of the application find and copy **Access Key** which you will set as one of the environment variables





4. **In the first part of the course** API requests to the Unsplash API will be made directly from the `frontend` application. In order to insert Unsplash Key as environment variable in the frontend React application do the following:

1. In the `frontend` folder create new file `.env.local`
2. In the `.env.local` file add one environment variable as in the example

NOTE: Use your own **Access Key** from your App at the unsplash.com as key in this example is not valid

```
REACT_APP_UNSPASH_KEY=2MJApIkV1fg2LmQlneILfH2ttLzSdPKfGOKM
```

5. **In the remaining part of the course** API requests to the Unsplash API are made from the backend `api` service. Backend API service will also communicate with MondoDB database and you need to set username and password for the database in the `.env.local` file as well.

NOTE: you need to create `.env.local` file in the `backend` folder starting from the following section

-  Importing env variables from the file in the Python app

In order to create a file with environment variables in the backend Python application you need to do the following:

1. In the `api` folder, create new file `.env.local`
2. In the `.env.local` file, add such environment variables as in the example

NOTE: Keep values for `MONGO_USERNAME` and `MONGO_PASSWORD` as in the example. Use your own **Access Key** from your App at the unsplash.com as value for the `UNSPASH_KEY`.

Key in this example is not valid

```
MONGO_USERNAME=root  
MONGO_PASSWORD=very-strong-db-password  
UNSPASH_KEY=2MJApIkV1fg2LmQlneILfH2ttLzSdPKfGOKM
```

6. You are **done** with the creation of the files with environment variables! 🍌



Running Images Gallery application without Docker

In the first part of the course till section "**Dockerizing Backend Flask API Service**" you need to start frontend and api services separately.

NOTE: *In order to run application successfully you have to go through all previous steps in this Guide.*

Steps to run application without Docker

1. In the Terminal change directory to the `frontend` folder

```
cd frontend
```

2. Run `frontend` React application in the *development* mode

```
npm start
```

3. Keep previously started process in the terminal running and open **new tab** in the terminal (if supported by the terminal application) or open **new terminal window**
4. Change directory to the `api` folder

```
cd api
```

5. Run Python backend `api` application

```
python main.py
```

6. Keep both processes running. In the web browser open frontend application

```
http://localhost:3000
```

7. You are **done** with start of both services without Docker! 👍



Running Images Gallery application with Docker

Starting from the section "**Dockerizing Backend Flask API Service**" you will run all services using **Docker Compose**. While running services using **Docker Compose** it will become much easier to add other services like MongoDB database and run all services using just single docker command.

NOTE: *You could still run `frontend` and `api` services separately and don't use Docker but in such case you will need to install MongoDB on your computer or use remote MongoDB cloud hosting service*

Steps to run application without Docker

1. In the Terminal, change directory to the main `images-gallery` folder where `docker-compose.yml` file is located

```
cd images-gallery
```

2. Bring all services defined in the `docker-compose.yml` up simultaneously. With `-d` option all containers will be started in the detached mode.

NOTE: *If you start project containers for the first time Docker will also build Docker images for the `api` and `frontend` services*

```
docker-compose up -d
```

3. You are **done** with start of all services using Docker! 👍



Managing services which are running using Docker

While running services using Docker you could check status of all containers, stop all containers, run containers over again and if necessary rebuild images for such services as `api` and `frontend`.

You need to rebuild images each time when you install new dependencies in the `frontend` application or add new packages in the `api` application

Docker commands for management of the project services

1. Check statuses of all running Docker containers

```
docker ps
```

2. Stop all services

```
docker-compose down
```

3. Rebuild Docker images for all services and recreate all containers

```
docker-compose up -d --build
```



Launching frontend application and sending requests to the backend API

Both frontend and api applications are now running on your local computer and you could access them by using `localhost` name or `127.0.0.1` IP address.

Steps to launch frontend application and send requests to the backend API

1. In the Google Chrome or any other web browser open connection to the following web page in order to open client connection to the `frontend` application

<http://localhost:3000>

2. Backend `api` service is available at the following URL

localhost:5050

NOTE: Opening following URL in the web browser will essentially send GET request to the backend API service. You could send other types of requests like POST or DELETE using Postman

3. There is `mongo-express` service which is GUI for the `mongo` database service. It is available at the following URL

<http://localhost:8081>

4. Now you are **done** with project setup and could continue development! 🙌