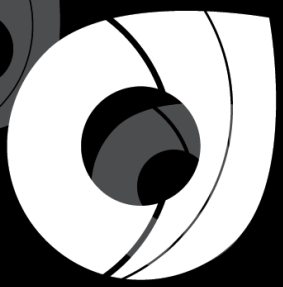
Corndel DevOps Engineering Programme

in association with Softwire

Module 1: Project Exercise



Corndel Digital.

A5J

Project Exercises

The technical concepts that we'll cover during this course can sometimes feel quite abstract when reading about them or following examples. Putting those concepts into practice by actually using them to achieve a goal is often the best way to understand them in detail. It also provides an opportunity to dig beneath the surface and get to know some of the pitfalls and finer points that hands-on experience brings.

The course project is your opportunity to get some real-world experience with these concepts. Each module has an associated **Project Exercise**, which will demonstrate your mastery of the core concepts introduced in that module. As you progress through the course, these exercises will build on each other to produce a final project that makes use of all the key concepts, tools and skills we'll cover.

Each exercise has a set of **Core Goals** that you must complete and submit, plus some **Stretch Goals** that go beyond the concepts covered by the reading material and provide an opportunity for you to delve deeper into some more advanced topics. You can choose to include these goals in the code you submit, but it is not a requirement for the course.

Let's get started!

Module 1

Building a To-Do Web App

Introduction

In this first exercise of the course, we're going to create one of the classic starter projects: **a To-Do app!**

We'll be developing this app throughout the course, with the ultimate goal of having a fully automated pipeline to build, test and deploy it to a cloud platform.

This exercise gives you a chance to practice your Python and become familiar with a popular web framework called Flask (https://palletsprojects.com/p/flask/).

Exercise structure

Each project exercise will follow the same structure:

1. **Introduction:** what we'll be adding to the course project in this exercise.
2. **Setup:** any steps you need to follow first, in order to be able to complete the exercise.
3. **Exercise:** the core goals that you'll need to complete before submitting your work.
4. **Stretch goals:** optional additions you can make to the project that go beyond the core concepts for this module.
5. **Hints:** some pointers to helpful resources if you find yourself stuck.

Setup: Checkout the starter project

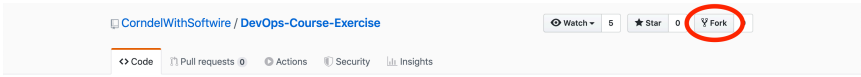
Before you start, this exercise assumes you've already installed Python and Git on your machine and created a GitHub account. If not, follow the relevant instructions here:
https://www.python.org/downloads
here:
https://git-scm.com/downloads
and here:
https://github.com/join

Step 1: Clone the starter project

We've created a repository on GitHub containing the initial project files to get you up and running.

Start by ***forking*** this repo — essentially making a copy of it under your own GitHub account that you can then modify. Click the "fork" button at the top-right of this GitHub page:
https://github.com/CorndelWithSoftwire/DevOps-Course-Starter

The fork button should look like this:



Once you've got your own fork of the project, you can clone it to your local machine. Choose a suitable folder on your hard drive to contain the files for this course (we suggest `C:\Work` if using Windows). If using Windows, open that folder in File Explorer, right-click anywhere within it and select "Git Bash Here" from the menu that appears. This will open a Git Bash terminal with that folder set as the current working directory. You can then run `git clone` to create a local copy of your repository:

```
$ git clone https://github.com/your-username/DevOps-Course-Exercise.git
$ cd DevOps-Course-Exercise
```

(**Note:** be sure to replace `your-username` with your GitHub username in the above command!)

Step 2: Install the Flask framework

We'll be using the Flask web framework and a couple of other packages in our app. The codebase contains a setup script to create a virtual environment for the project and install the required packages using pip. Execute that script by running the following command in your terminal:

```
$ source setup.sh
```

Submitting your work

We'll be using Pull Requests on GitHub to review your exercise submissions during the course.

To create a pull request once you're happy with your work:

1. Make sure you have committed all your changes.
2. Create a new branch, e.g.
   `git checkout -b exercise-1`
3. Push that branch to your remote repository on GitHub, e.g.
   `git push origin exercise-1`
4. Visit your repository on GitHub and create a pull request for the branch you just pushed.
5. Submit the URL for the pull request as your exercise solution on Aptem.

You can submit further changes to the pull request once it's created by committing those changes to your local branch and then pushing them to the remote branch on GitHub. The pull request will update automatically.

You can read more about pull requests on GitHub here: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

Step 3: Run the app and check it works

Flask comes with a built-in web server you can use to run your web app locally. Check that the app works as expected by running the following command:

```
$ flask run
```

You should see output similar to the following:

```
 * Serving Flask app "app" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C
to quit)
 * Restarting with fsevents reloader
 * Debugger is active!
```

You should then be able to visit **http://localhost:5000/** in your web browser and see the app in all its glory!

Don't forget to commit!

Make sure you use **`git commit`** to save your work regularly.

It's good practice, and lets you revert back to a previous revision if you find you've broken everything horribly!

Try to make your commit messages concise and descriptive. Using source control tools effectively is just as important as the code you submit.

Exercise: View and create todo items in the web app

In this exercise, you will need to:

- Make the app render the index page HTML template.
- Display the list of saved todo items on the index page.
- Create a route to add a new todo item to the list.

Show the index page template

The starter project contains a template for the index page (`index.html` in the "templates" subdirectory). For more details about how to use templates, have a read through the official

Flask tutorial (https://flask.palletsprojects.com/en/1.1.x/tutorial/templates/).

The `render_template()` function allows you to render the specified template as HTML and return it as the response. Use it to replace the "Hello World!" response in `app.py` with the index page content.

Display the list of saved todo items

We have provided a module containing various functions to get, create and save todo items, which you can call directly from your route functions in `app.py`.

The main functions of interest are:

- **`get_items()`** — returns the list of saved todo items.
- **`add_item(title)`** — adds a new item with specified title.

Modify the `index()` function to get the list of items and update the `index.html` template to display their titles as a list.

Note: *the helper module contains code to pre-populate some todo items for you, so you won't start with an empty list!*

Create new todo items

We want to be able to add new items to our todo list. In order to support this functionality, we'll need to add a new route to the app.

CRUD and REST

Web apps typically have dedicated routes to handle specific user actions, and those actions often follow the standard CRUD pattern (Create, Read, Update, Delete). By convention, most

CRUD

If you think about most actions users take within a typical application in the context of *resources* — i.e. the thing those actions are acting on — then a few common patterns tend to emerge.

Users can typically **create** resources (e.g. create a new order on a shopping web app, or add a new event in a calendar desktop app).

They can also view, or **read**, existing resources (view their order or see a calendar of all upcoming events). This could either be reading the details of a specific resource or viewing a list of all resources.

Often users can also edit, or **update**, existing resources (add items to their order; update the start time of an event).

Finally, they can **delete** an existing resource (cancel their order or remove an event).

These four actions — Create, Read, Update and Delete — are so common that they've come to be referred to as **CRUD** actions.

Note that most apps will usually have other, non-CRUD actions, too!

modern web apps aim to implement RESTful interfaces by adopting the appropriate HTTP method for each of those actions, i.e.

CRUD action	HTTP method
Create	POST (or sometimes PUT)
Read	GET
Update	PUT (or sometimes PATCH)
Delete	DELETE

We want to follow this convention when creating new items, so add a new route to the app that accepts HTTP POST requests. You can use the **`methods`** argument of the `route()` decorator to specify which HTTP method(s) it will handle.

In order to capture the title for the new item, you'll also need to add a **form** to the `index.html` template. This form should use the URL path of the route you've just created as its **`action`** value and use **`method="post"`** to submit a POST request.

Your form will need to contain an **`input`** field for the item title and a **`button`** element to submit the form data.

Finally, update the function for your new route so that it retrieves the item title from the form data contained in the request, using **`request.form.get('field_name')`** — where `field_name` is the name of the input field in your form.

Once the item has been added to the list, redirect the user back to the index page again and check that the list updates as expected.

Stretch goals

- Add functionality to mark an item as completed.
  - We've provided `get_item(id)` and `save_item(id)` functions as part of the `session_items.py` module which should help.

- Improve the styling of the app.
  - We're using Bootstrap (https://getbootstrap.com/) to style the pages; their documentation should provide some inspiration.

- Sort the item list by status ("Not Started", then "Completed")
  - Python's built-in `sorted()` function is a good place to start.

- Add functionality to remove an item.
  - You will need to add a function to `session_items.py` to do this, as well as a route and HTML elements to trigger it.

Stretch yourself

You'll notice that these stretch goals are more ambiguous and open-ended than the core goals for the exercise.

These are an opportunity for you to apply your creativity and decide how best to implement new features in the app.

You'll also probably need to do some additional research to find out how to achieve these goals, so break out a search engine and get exploring!

Hints

- We've provided a number of helper functions to retrieve and save todo items. You can use these functions directly in your routes within `app.py` — for example:

  ```
  items = session.get_items()
  ```

  To see how the functions work, take a look at the code in `session_items.py`

- The Flask tutorial (https://flask.palletsprojects.com/en/1.1.x/tutorial/) is a good resource for info and examples if you get

stuck, particularly the sections on templates and views.

- If you are unfamiliar with HTML, it might be worth reading this tutorial about forms: https://www.w3schools.com/html/html_forms.asp