What is Django?

The lab project at the end of this module will feature a very simple web application created using ***Django***. Django is a ***full-stack web framework*** written in Python. For this project, you'll only need to interact with it through HTTP requests, but it's still a good idea to understand what it is, and when it would be a good tool for you to use.

A full-stack web framework handles a bunch of different components that are typical when creating a web application. It contains libraries that help you handle each of the pieces: writing your application's code, storing and retrieving data, receiving web requests, and responding to them. If you need to build an application that has a web frontend, using a web framework like Django can save you a lot of time and effort, because a lot of challenges are already solved for you.

Web frameworks are commonly split into three basic components: (1) the application code, where you'll add all of your application's logic; (2) the data storage, where you'll configure what data you want to store and how you're storing it; and (3) the web server, where you'll state which pages are served by which logic.

Splitting your code like that helps you write more modular code, promotes code reuse, and allows for flexibility when viewing and accessing data. For example, you could have a simple web page where users of the system can access the information already stored in it, and a separate programmatic interface that can be used by other scripts or applications to transmit data to the system.

When you're writing a web application, there's a ton of little decisions to make. Relying on a framework like Django is similar to using external libraries for your code. There are a lot of features, which you can use very easily, instead of writing everything from scratch and re-making all of the same mistakes that we all make when writing a web application for the first time.

Django has a ton of useful components for building websites. In the lab project, Django will be used for serving the company website, including customer reviews. It does this by taking the request for a URL and parsing it using the ***urlresolver*** module. This is a core module in Django that interprets URL requests and matches them against a list of defined patterns. If a URL matches a pattern, the request is passed to the associated function, called a ***view***. This allows you to serve different pages depending on what URL is being requested. You can even build complex logic into the function handling the request to make more dynamic, interactive, and exciting pages.

Django can also handle reading and writing data from a database, letting you store and retrieve data used by your application. In the lab, the database holds the customer reviews for the company. When a user loads the website, the logic will ask the database for all available customer reviews. These are retrieved and formatted into a web page, which is served as a response to the URL

request. Django makes it easy to interact with data stored in a database by using an ***object-relational mapper***, or ***ORM***. This tool provides an easy mapping between data models defined as Python classes and an underlying database that stores the data in question.

On top of this, the Django application running in the lab includes an ***endpoint*** that can be used to add new customer reviews to the database. This endpoint is configured to receive data in JSON format, sent through an HTTP POST request. The data transmitted will then be stored in the database and added to the list of all reviews. The framework even generates an interactive web form, that lets us directly interact with the endpoint using our browser, which can be really handy for testing and debugging.

Django is one of many popular web frameworks. Alternative Python-based web frameworks similar to Django include Flask, Bottle, CherryPy, and CubicWeb. There are a host of other frameworks written in other languages too, not just Python.