Full Stack Web Developer Nanodegree Program Syllabus

Build Complex Web Applications

Program Overview

The goal of the Full Stack Web Developer Nanodegree program is to equip learners with the unique skills they need to build database-backed APIs and web applications. A graduate of this program will be able to:

- Design and build a database for a software application
- Create and deploy a database-backed web API (Application Programming Interface)
- Secure and manage user authentication and access control for an application backend
- Deploy a Flask-based web application to the cloud using Docker and Kubernetes

This program includes 4 courses and 5 projects. Each project you build will be an opportunity to apply what you've learned in the lessons and demonstrate to potential employers that you have practical full-stack development skills.

Estimated Length of Program: 4 months
Frequency of Classes: Self-paced
Prerequisite Knowledge: A well-prepared learner is able to:

- Write and test software with Python or another object-oriented programming language
- Query a SQL database using SELECT
- Write to a SQL database using INSERT
- Write software for front end applications and websites using JavaScript to:
    - Fetch and display data from an API using AJAX or Fetch
    - Organize data using JSON (JavaScript Object Notation)

Project 1: Design a Venue Booking Database

For your first project, you'll be building out the data models and database for an artist/venue booking application. The fictitious startup Fy-yur is building a website that facilitates bookings between artists who can play at venues, and venues who want to book artists. This site:

- Lets venue managers and artists sign up, fill out their information, and list their availability for shows.
- Lets artists browse venues where they can play, and see what past/upcoming artists have been booked at a venue.
- Lets a venue manager browse artists that would like to play in their city, and see what past/upcoming venues where the artist has played/will be playing.

The goal of this project is to build out the data models for this booking application. A prototype design of the web app will be provided. You'll use SQLAlchemy and Postgresql to build out the data models upon

which this site will rely. You'll write out both the raw SQL and SQLAlchemy commands to run for powering the backend functionality of the website.

Supporting Course Content: SQL and Data Modeling for the Web

Lesson	Learning Outcomes
Connecting and Interacting with Databases	➔ Describe and explain the client-server model ➔ Describe and explain the TCP/IP communication protocol ➔ Describe and explain the base unit of database work: transactions ➔ Install the PostgreSQL database management system ➔ Create and manage Postgres databases with the psql client ➔ Install the psycopg2 Python+Postgres database driver ➔ Create and manage Postgres databases using the psycopg2 Python database driver
Intro to SQLAlchemy and SQLAlchemy ORM Basics	➔ Describe and explain the use cases for an Object Relational Mapping (ORM) library ➔ Describe and explain the abstraction layers of SQLAlchemy ➔ Connect to and manage a database using composable SQL expressions ➔ Define data model objects with Python using SQLAlchemy ORM ➔ Connect data models to a lightweight Flask web application ➔ Build data models using different types of data
SQLAlchemy ORM in Depth	➔ Explore and retrieve data using the SQLAlchemy Model.query object ➔ Create database sessions for executing database transactions ➔ Execute database transactions within a connection session ➔ Describe and explain the SQLAlchemy object lifecycle ➔ Build a lightweight data app using SQLAlchemy ➔ Describe and explain the Model-View-Controller (MVC) application architecture ➔ Retrieve from data from a webform using Flask ➔ Update data models using data migrations ➔ Migrate data using Flask-Migrate and Flask-Script ➔ Define and code relationships between tables and objects using SQLAlchemy ➔ Implement database methods to query relationships between data models
Build a CRUD App with SQLAlchemy ORM - Part 1	➔ Use the CRUD (Create, Read, Update, Delete) model to build a small database backed app ➔ Capture user input from a webform to add and modify data to a database ➔ Manage data using database sessions in an application controller
Migrations	➔ Modify a data schema using Flask-Migrate and Alembic ➔ Write migration scripts to update data schemas using Flask-Script
Build a CRUD App with SQLAlchemy ORM - Part 2	➔ Update database models using webforms and application routing ➔ Delete information from a database using SQLAlchemy ➔ Model and control relationships between different types of data

	objects ➔ Implement one-to-many and many-to-many relationships using SQLAlchemy ➔ Execute complex database queries on related data models

Project 2: Trivia API

In this project, you will use the skills you've developed to build a Trivia API. The API will allow users to:

- Search for trivia questions and answers via category and difficulty
- Add new questions
- Modify the difficulty rating of questions.

The goal of this project is to use APIs to control and manage a web application using existing data models. You'll be given a set of data models and the application front end. Your task will be to implement the API in Flask to make the Trivia game functional.

Supporting Course Content: API Development and Documentation

Lesson	Learning Outcomes
Introduction to APIs	➔ Describe and explain the definition and use cases of APIs (Application Programming Interface) ➔ Describe and explain how APIs are used to connect application front ends to server backends
HTTP and Flask Basics	➔ Describe and explain the Hypertext Transfer Protocol (HTTP) ➔ Describe and explain the components of an HTTP request ➔ Describe and explain the different HTTP methods (verbs) ➔ Describe and explain HTTP status codes ➔ Request information from a server using cURL and HTTP requests ➔ Install the Python Flask micro application framework ➔ Set up and Configure a Flask application ➔ Create a Flask endpoint (route)
Endpoints and Payloads	➔ Structure and Organize API Endpoints ➔ Describe and explain Cross-Origin Resource Sharing (CORS) ➔ Manage CORS requests using HTTP headers ➔ Manage CORS controls using Flask-CORS ➔ Parse request path and body from an HTTP request ➔ Implement HTTP POST, PATCH and DELETE methods using Flask ➔ Handle application errors using Flask
API Testing	➔ Describe and explain the purpose and benefits of API testing ➔ Test a REST API using Flask and unittest ➔ Develop an application iteratively and safely using Test Driven Development (TDD)

API Documentation	➔ Read and explore API documentation from a number of API developers ➔ Write effective documentation for your own API

Project 3: Coffee Shop Full Stack

In the third project of the program, you will build the backend for a coffee shop application. You'll add user accounts and authentication to your application and use role-based access management strategies to control different types of user behavior in the app. The application must:

- Display graphics representing the ratio of ingredients in each drink.
- Allow public users to view drink names and graphics.
- Allow the shop baristas to see the recipe information.
- Allow the shop managers to create new drinks and edit existing drinks.

This project will give you a hands-on chance to practice and demonstrate your new skills, such as:

- Implementing authentication and authorization in Flask
- Designing against key security principles
- Implementing role-based control design patterns
- Securing a REST API
- Applying software system risk and compliance principles

Supporting Course Content: Identity Access Management

Lesson	Learning Outcomes
Foundations	➔ Describe and explain the use cases and differences between authorization and authentication ➔ Describe the problem of security and the risks of unsecured or improperly secured application systems ➔ Describe different types of security attack ➔ Inspect requests and responses for an application using Postman ➔
Authentication	➔ Describe common methods for application authentication ➔ Explain why passwords are not the ideal method for authentication ➔ Implement an application authentication layer with Auth0 ➔ Secure API communications using JSON Web Tokens (JWT)
Passwords	➔ Describe the risks associated with password controlled systems ➔ Mitigate access risks associated with SQL injection by validating and sanitizing database inputs ➔ Secure database data in a database using standard encryption practices ➔ Describe how an attacker can use rainbow tables to gain access to

	a system ➔ Improve security of hashed passwords and encrypted data using the 'salt' method ➔ Increase application security by using best practices to avoid logging and serializing sensitive data
Authorization	➔ Describe the concept of authorization and access control ➔ Define 'permissions' in the context of an application ➔ Constrain permissions in an application by using role-based access control (RBAC) ➔ Define permission roles using Auth0 ➔ Identify user permissions and roles from JWTs (JavaScript Web Tokens)
Thinking Adversarially	➔ Prevent accidental access to privileged information in Git repositories by using environment variables ➔ Mitigate risks to Git master branch changes by developing in feature branches ➔ Employ code review as a practice to mitigate security risks ➔ Test API and authentication practices with integration testing ➔ Describe common types of adversarial attacks on network systems

Project 4: Deploy Your Flask App to Kubernetes Using EKS

In this project, you will create a container for your Flask web app using Docker and deploy the container to a Kubernetes cluster using Amazon EKS. By the end of the project, you will have deployed your application live to the world, where it should be accessible by IP address. You'll use automated testing to prevent bad code from being deployed and monitor your app's performance using AWS tools.

Supporting Course Content: Server Deployment and Containerization

Lesson	Learning Outcomes
Containers	➔ Describe and explain the benefits and use cases for containerized environments ➔ Install Docker on a local machine ➔ Define a container environment using a Dockerfile ➔ Download and launch a Docker container ➔ Store and share a docker container
Deployment	➔ Describe and explain container orchestration, how it works and the general use case ➔ Describe and explain how Kubernetes manages container clusters ➔ Deploy a Docker container to a Kubernetes cluster using AWS EKS and the AWS command line interface (CLI) ➔ Manage Kubernetes clusters using the AWS CLI ➔ Implement Continuous Delivery (CD) and Continuous Integration (CI) with AWS CodePipeline and AWS CodeBuild

Project 5: Full Stack Web Developer Nanodegree Program Capstone

In this final capstone project, you will combine all of the new skills you've learned and developed in this course to construct a database-backed web API with user access control. You will choose what app to build and then you'll design and build out all of the API endpoints needed for the application and properly secure them for use in any front end application (web or mobile).