CLOUD CONTRAPTIONS
COMPUTER PROGRAMMING & TRAINING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Full-Stack Web Development with Svelte and Flask-RESTX

Duration

5 days

Description

The Full-Stack Web Development with Svelte and Flask-RESTX course covers Svelte, a modern JavaScript framework, and its integration with Flask, a micro web framework written in Python. The course starts with the fundamentals of Svelte, including how to set up a development environment, create static and dynamic pages, and handle event and state management. It then dives into the basics of HTTP and URLs, databases, and Python web frameworks. The course concludes with a detailed guide on how to build a REST API with Flask, including how to connect to a database, handle cookies, handle errors, document and version APIs, and secure a REST API with JWT. This course is perfect for professionals who want to build robust, full-stack applications using Svelte and Flask.

Objectives

- Understand the fundamentals of Svelte and how it compares to other frameworks.
- Set up a development environment for Svelte and learn how to use SvelteKit.
- Create static and dynamic pages using Svelte, including understanding their structure and how to handle images, CSS, and JavaScript content.
- Understand the principles of template reactivity and component basics in Svelte.
- Learn about event handling, data binding, and forms in Svelte.
- Understand how to handle lifecycle events and state management in Svelte.
- Learn about routing, error handling, and asynchronous data in Svelte.
- Understand the basics of HTTP and URLs, databases, and Python web frameworks.
- Learn how to build a REST API with Flask, including how to connect to a database, handle cookies, and handle errors.
- Understand how to document and version APIs, and secure a REST API with JWT.

Prerequisites

All students must have Python, JavaScript, and HTML programming experience. Experience with CSS is helpful, but not required.

Training Materials

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub in the form of documentation and extensive code samples. Students practice the topics covered through challenging hands-on lab exercises.

Software Requirements

Students will need a free, personal GitHub account to access the courseware. Student will need permission to install Python, Node.js and Visual Studio Code on their computers. Also, students will need permission to install Conda/Pip Packages, NPM Packages, and Visual Studio Extensions. If students are unable to configure a local environment, a cloud-based environment can be provided.

Outline

- Introduction
  - What is Svelte?
  - What problem does Svelte solve?
  - Svelte vs. Other Frameworks
  - Svelte Compiler
- Development Environment
  - Requirements
  - SvelteKit
  - Svelte Files
  - Svelte Extension for Visual Studio Code
  - Run/Debug Svelte App in Visual Studio Code
  - Svelte Extension for WebStorm
  - Run/Debug Svelte App in WebStorm
- SvelteKit Overview
  - Vite Tooling
  - Development Server
  - Routing
  - Deployment
  - Server-side rendering
  - Unit Testing
- Getting Started
  - Exploring the REPL

- – Svelte Layout
- – Svelte Page
- – Svelte Component
- – Svelte Architecture
- – Svelte Element Directives
- – Compiling Svelte Files
- Static Pages
  - – What is a Static Page?
  - – What problem do Static Pages solve?
  - – Static Page File Structure
  - – Setting Head Content
  - – HTML Content
  - – CSS Content
  - – Comments
  - – Scoped CSS
  - – Handling Images
  - – Hot Module Reloading
  - – Server Pre-rendering
  - – Page Routing
- Dynamic Pages
  - – What is a Dynamic Page?
  - – What problem do Dynamic Pages solve?
  - – Client-Side Rendering
  - – Dynamic Page File Structure
  - – JavaScript Content
  - – Using Variables
  - – Using Expressions
  - – Data Binding
  - – Class and Style Directive
  - – Event Binding
  - – Logic Blocks
  - – Debug Tag
- Template Reactivity
  - – Principles
  - – Changing Data through Assignments

- – Reactive Statements
- – Updating Arrays and Objects
- Component Basics
  - – What is a Component?
  - – What problem does it solve?
  - – Calling Components vs HTML Elements
  - – Component File Structure
  - – Component Props
  - – Component Events
- Component Composition
  - – What is Component Composition?
  - – What problem does it solve?
  - – Nested Components
  - – Passing Data to Child Components
  - – Handling Events and Receiving Data from Child Components
  - – Component Tree Best Practices
- Event Handling
  - – Event Handling Element Directives
  - – DOM Events
  - – Adding Event Handlers
  - – In-line Handlers
  - – Event Modifiers
  - – Dispatching Component Events
  - – Forwarding Events
- Data binding
  - – Top-down data binding by default
  - – Communication with props and events
  - – Using two-way data binding
- Forms
  - – HTML Form Element
  - – Named Form Actions
  - – Form Validation
  - – Form Submission
  - – Progressive Enhancement
- Lifecycle

- – Mount
- – Destroy
- – Before Update
- – After Update
- – Tick
- State Management
  - – Stores
  - – Writable Stores
  - – Auto-subscriptions
  - – Readable Stores
  - – Derived Stores
  - – Custom Stores
  - – Store Bindings
  - – Page Store
  - – Navigation Store
  - – Updated Store
- Routing
  - – What is Routing?
  - – What problem does Routing solve?
  - – Pages
  - – Layout
  - – Route Parameters
  - – API Routes
- Errors and Redirects
  - – Handling Errors and Redirects
  - – Error Pages
  - – Fallback Errors
  - – Redirects
- Asynchronous Data
  - – Promises & async/await
  - – Fetching data from a REST API
  - – Subscriptions
  - – Stores
- Quick Review of HTTP and URLs
  - – What is HTTP?

- What are HTTP Verbs?
- What are the parts of a URL?
- Path Parameters
- Query String Parameters
- Common HTTP Headers
- Common HTTP Status Codes
- Quick Review of Databases
  - What is a Database?
  - Connecting to a Database
  - Query data from a Database
  - Modify data in a Database
- Overview of Python Web Frameworks
  - Django
  - Flask
  - FastAPI
- Representational State Transfer
  - What is REST?
  - HTTP Verbs and their use in REST
  - OpenAPI
  - JSON
  - Swagger
- Building a REST API with Flask
  - Create an Application
  - Define a Hello World Route
  - Configure a Path Parameter
  - Configure a Query String Parameter
  - Reading Request Body
  - Writing a Response Body
  - Handling Cookies
  - Handle Errors
  - Return Specific Status Codes
  - Asynchronous Routes
  - Serving Static Files
- Database Access
  - Connecting to a Database

- – Querying Data
- – Modifying Data
- – Transactions
- – SQLAlchemy
- – Running a Database with Docker
- Document and Version APIs
  - – Swagger
  - – OpenAPI
  - – Versioning
- Security
  - – Secure a REST API with JWT
  - – Configure CORS
- Conclusion