**Project Report: Building an API with Go and PostgreSQL**

**Introduction**

This project involves building a RESTful API using Go as the programming language and PostgreSQL as the database. The API allows CRUD (Create, Read, Update, Delete) operations on an "employees" table in the PostgreSQL database. The project is structured into several folders: connection, models, controller, and router, each serving a distinct purpose in the API's architecture.

**Step 1: Establishing the Database Connection**

**Folder: connection**

* **Objective:** Establish a connection to the PostgreSQL database.
* **Implementation:**
  + A connection.go file was created in the connection folder.
  + The PostgreSQL connection string, which includes the username, password, and database name, was defined.
  + The connection was established using sql.Open, and DB.Ping() was used to verify that the connection to the database was successful.
  + Any errors during the connection process were handled using log.Fatal.

**Step 2: Defining Data Models and CRUD Operations**

**Folder: models**

* **Objective:** Define the data structure that mirrors the database table schema and implement the CRUD operations.
* **Implementation:**
  + A PostgresStruct struct was defined to represent the schema of the employees table in the database.
  + CRUD operations (CreateEmployee, ReadEmployees, UpdateEmployee, DeleteEmployee) were implemented in functions, using SQL queries to interact with the database.
  + The database/sql package was imported to manage database interactions.
* **CRUD Operations:**
  + **Create:** Insert a new employee record into the database.
  + **Read:** Fetch one or all employee records.
  + **Update:** Modify an existing employee record.
  + **Delete:** Remove an employee record from the database.

**Step 3: Handling HTTP Requests with Controllers**

**Folder: controller**

* **Objective:** Handle incoming HTTP requests and map them to the corresponding CRUD operations.
* **Implementation:**
  + The controller functions were created to manage the HTTP request and response cycle.
  + The gorilla/mux package was used to route HTTP requests to the appropriate handler functions.
  + Functions such as GetEmployees, CreateOneEmployee, UpdateOneEmployee, and DeleteOneEmployee were implemented to handle GET, POST, PUT, and DELETE requests, respectively.

**Step 4: Routing HTTP Requests**

**Folder: router**

* **Objective:** Define and handle different types of HTTP requests (GET, POST, PUT, DELETE).
* **Implementation:**
  + The router.go file was created to define routes for different API endpoints.
  + The gorilla/mux router was configured to map each endpoint to the appropriate controller function.
  + The router was designed to handle the various HTTP methods and route them to the correct handler functions.

**Step 5: Main Application Setup**

**File: main.go**

* **Objective:** Set up the application to run on a specific port and initialize the router.
* **Implementation:**
  + The main.go file was created to serve as the entry point of the application.
  + The HTTP server was set to listen on port 8000.
  + The router was initialized, and the Connect function from the connection package was called to ensure that the database connection was established before handling any requests.

**Conclusion**

This project successfully demonstrates how to build a RESTful API using Go and PostgreSQL. The API allows for the creation, retrieval, updating, and deletion of employee records in a PostgreSQL database. Each step in the project—from setting up the database connection, defining data models, handling HTTP requests, routing, and setting up the main application—was crucial in building a fully functional API.