**Backend Engineer Candidate (Jerod Gawne)**

**What .Net framework would you use to develop a RESTful API?**

* I wouldn’t; I would use FastAPI w/ Python because I’ve been working primarily in Python for the last seven years.

**What is the difference between docker and docker-compose?**

* The docker cli is used when managing individual containers on a docker engine. It is the client command line to access the docker daemon api.
* The docker-compose cli can be used to manage a multi-container application. It also moves many of the options you would enter on the docker run cli into the docker-compose.yml file for easier reuse. It works as a front end "script" on top of the same docker api used by docker, so you can do everything docker-compose does with docker commands and a lot of shell scripting.

**What unit testing frameworks have you used for writing unit tests?**

* + If given a choice I would use PyTest, but there is also the built-in unit testing framework.

**What are SOLID design principles? Explain In brief each of them.**

* S: Single-responsibility Principle
  + A class should have one and only one reason to change, meaning that a class should have only one job.
* O: Open-closed Principle
  + Objects or entities should be open for extension but closed for modification.
* L: Liskov Substitution Principle
  + Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.
* I: Interface Segregation Principle
  + A client should never be forced to implement an interface that it doesn’t use, or clients shouldn’t be forced to depend on methods they do not use.
* D: Dependency Inversion Principle
  + Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.

**How can we communicate with another service asynchronously? Explain different ways.**

* Asynchronous code utilizing micro-services
* Message queues or similar technologies

**When making changes, explain how the code is branched from GIT repo, and then merged back into the master branch.**

* The way this question is worded it doesn’t give me a clear idea of what you’re really asking for.
* Myself along with many other people follow the git-flow pattern. One example:
* Two primary branches (main & dev), developers create a new branch based on dev and make their code changes. The developer then creates a pull request which allows a code-review and after approval the code is merged into the development branch and the developers branch is deleted. If the development branch passes all of the tests a pull request is then created to merge the development branch into the main branch.
* There are many different ways to work within git, it all really depends upon what the team has decided to use.

**How do you prevent SQL injection attacks?**

* Follow guidance from the O.W.A.S.P.
  + Use of Prepared Statements (with Parameterized Queries)
  + Use of Stored Procedures
  + Allow-list Input Validation
  + Escaping All User Supplied Input
  + Enforcing Least Privilege
  + Performing Allow-List Input Validation

**How do you prevent XSS attacks?**

* Follow guidance from the O.W.A.S.P.
  + Understand how your framework prevents XSS attacks and where it has gaps.
  + Protect each variable within a web page
  + Output encoding
  + HTML sanitization

**What steps should we take to prevent application internals getting leaked to the client, when exception happens in the backend services.**

* Don’t output logs to the console; Use a logging sink instead (external application that collects and provides access to the logs/errors).
* Ensure you have sufficient exception handling to prevent console output.

**What is the role of validation in hardening application security?**

* Prevents code injection/execution.

**How should a password be stored in the database?**

* Using a one-way hashing algorithm

**What is an ORM and what ORMs have you worked with?**

* Object relational mapper; maps programming objects to database objects (tables, rows, etc).
* I normally use SQLAlchemy along with Pydantic for data validation.

**What is the difference between inner join, left outer join and right outer join?**

* Inner join returns only rows that match criteria in both tables
* Left outer join returns all rows from the first table and any matching rows from the 2nd table
* Right outer join is just the opposite of a left outer join

**What are some of the ways to improve query performance?**

* This depends entirely on the type of database and how it operates under the hood.
* Don’t over-normalize the database; joins are very expensive operations.
  + Ensure data that is normally used at the same time exists in the same row to reduce the number of queries the db performs.
* Create a(n) index(s) based on common query keys
* Partition the db based on related data and spread out to different nodes. The partition should make sense according to the data and create reasonably even nodes.
  + E.g. for names db partition a-e, f-j…

**What are some of the micro-service best practices?**

* Have a dedicated infrastructure
* Have a dedicated database
* Ensure you are following the single responsibility principle
* Document the infrastructure so anyone not familiar can understand the flow

**What complexities does a micro-service architecture introduce?**

* The more you have the harder it is to keep track of what everything is doing, where it’s running, and what part it plays in the flow.
* Problem tracing
* Communication (between services)

**Explain how the OIDC workflow works.**

* Client requests access token but authenticating with server
* The server validates the credentials and if valid issues an access token and optionally a refresh token
* The client makes requests with the server providing the token each time
* The server responds if the token is valid
  + If token is not valid server returns an error
  + Client requests new token by presenting credentials along with refresh token