1. What is the difference between TDD and BDD?

Test Driven Development (TDD) is used to develop features. You write a failing test for a feature, then code the feature to result in a passing test. Repeat until the code is feature complete. Behavior Driven Development (BDD) is used to incorporate features into executables. You write an executable that fails testing because the feature specification isn't present. Then you write the feature (i.e. add the behavior) to pass the test. TDD tends to be specific to the developer or a pair of developers whereas BDD tends to be more team, tester, and user focused.

"For small, co-located, developer-centric teams, TDD and BDD are effectively the same."

- https://www.pluralsight.com/blog/software-development/tdd-vs-bdd#:~:text=TDD%20is%20a%20development%20practice,BDD%20are%20effectively%20the%20same.

2. What does mocking a class allow you to do?

Mocking allows you to test code units that have external dependencies. Through stubs, fakes, and mocking, the mocking class replaces external dependencies with static values so that the unit of code under test can be observed under tightly controlled cases.

- https://www.telerik.com/products/mocking/unit-testing.aspx#:~:text=Mocking%20is%20a%20process%20used,or%20state%20of%20external%20dependencies.

3. What is the value in separating your code into controller, service, and data access layers rather than keeping it all in the same files?

If you try to have your features implemented in one layer (i.e. your controller layer) you can run into the following problems:

- " Controllers that have lots of code in them, doing lots of things - AKA "fat controllers".
- Closely related to the previous one, your code looks cluttered. With controllers making 4 or 5 or more database/model calls, handling the errors that could come with that, etc., that code probably looks pretty ugly.
- You have no idea where to even begin writing tests.

- Requirements change, or you need to add a new feature and it becomes really difficult to refactor.
- Code re-use becomes pretty much *non-existent*."

By separating out the web/HTTP actions from the business rules and data model, you greatly relieve the above problems.

- https://www.coreycleary.me/why-should-you-separate-controllers-from-services-in-node-rest-apis

4. Why would you want to avoid putting credentials in plaintext in your code?

Any (un)authorized person can read and exploit credentials (e.g. passwords) when they're left as plaintext in the code.

- https://www.passcamp.com/blog/dangers-of-storing-and-sharing-passwords-in-plaintext/

5. What is one method that can be used to avoid putting plaintext database usernames and passwords into your code?

Create a hash of the credential and store that. This also has the advantage of not involving encryption since encryption keys can be stolen.

- https://snyk.io/learn/password-storage-best-practices/#:~:text=2.-,Hash%20all%20passwords,hash%20can't%20be%20reversed.

6. What is your favorite thing you learned this week?

This week could have been made a lot better by having assignments with basic code examples - not completed code – that we could code along to the videos with. *Then* having completed code to verify our work when done.