DIY – Peer Review
David Chan
3 March 2023
Anh Huynh
8 March 2023

Major Positives and Negatives

Positives	Negatives

• UX efficient
  ○ Visually, the DIY interface looks simple and understandable, making it easy to use
  ○ Design also appears to be easy to program, too
• Input validation and sanitation
  ○ User input is validated so SQL injection attacks are prevented
• Minimal trips to data storage layer
  ○ Little trips to the data storage layer means better runtime
	• Has room for concurrency/parallelism
	  ○ Most Entry layer calls wait for other Entry layer calls to finish before starting, when it could be concurrently started on the former's return trip to the user
	• Some methods do not follow async naming convention
	  ○ Async methods should be suffixed with "Async"
	  ○ Await keyword missing from all async method calls
	• Some syntax errors
	  ○ > var result = public async Task<Bool> AddDIY(string dIYID, string email)
	• No use of dependency injections
	  ○ Classes from other namespaces are being instantiated instead of being passed as a dependency injection through a constructor
	• Search feature
	  ○ Feature should make use of pagination instead of making API calls to fetch next series of items
	  ○ HTTPPost is not necessary when only fetching a list of items
	• Returning void
	  ○ Returning void does not exist and syntactically incorrect
	• Design inconsistent with return type from Entry Point to Presentation

	○ Some returns are HTTP (200 OK) and some are type values • Passing video file format through code ○ Passing video files through code doesn't seem possible, and if it were, seems incredibly inefficient

Unmet Requirements

- Are no unmet requirements

Design Recommendations

- Leverage a design which accounts for concurrency instead of going back and forth between Entry and Data Storages.
- UI can use a slight revamp; don't want it to be too boring.
- Using await keywords cohere with the async methods and should be used or else there will be a compiler error
- Consider using dependency injection when wanting to instance classes from other namespaces. Dependency injection has the option to control the lifetime of the instance object and that can serve as a spatial optimization.
- When fetching multiple items, its result to the presentation layer should be in a pagination format, so that the handler in the presentation layer can easily query for the next set of items, instead of writing an entirely new controller endpoint.
- Instead of returning void, return a nullable value (i.e. public string? Name) and check for a null result to validate the response.
- If the Entry layer is an API interface, it should always return HTTP status codes and a response body, not type values from C#. Consider different status codes for different resulting cases (200 for success, 404 for not found items, 403 for forbidden authentication, etc.).
- Video files shouldn't be passed in the program, there should be a URI which references to the video file that is stored on some cloud service.

Test Recommendations

Majority of designs in LLD are very linear; they start at the User then continues narrowly throughout the system to the Datastore then back to the User. Because there is very little concurrency or parallelism, the best type of test to deploy are end-to-end tests. A program like Postman is perfect for these type of tests and automating them.