

Pro Dev Session



Pro Dev Session

You may want to write this down.

Collaboration is an essential part of being a great teammate and developer. To be a good collaborator, we must master our collaboration tools, namely git and GitHub.

Today we will get more practice with git. We'll be building on this for the next several lessons.



Pro Dev Session

You may want to write this down.

Recall what we've learned so far ...





Check-in Time

- What git command do you use to temporarily save changes that you don't want a permanent record of?
- What git command is used to permanently save files exactly as they are in that moment?
- What git command tracks (or stages) all files in a project?
- What git command is used to create a safe isolated place to work on new features?
- What git command is used to see a list of all previous saved versions?
- What git command is used to see a specific older saved version of the project?



Git Demo

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

Pay careful attention to this demo of:

```
git clone
git add -A
git commit -m "<message here>"
git status
git stash
git branch <branchname>
git log
git checkout <branchname or SHA>
```

Pro Dev Session

You may want to write this down.

Today we will get more practice with these commands before we add new ones in the next class.



Git Practice

Work individually but with the support of your classmates to complete all of the goals below.

Goals:

- Create a new repository on GitHub
- Clone the repository to your local machine
- Create a new IntelliJ project with a main method that prints Hello World
- Add, commit, and push your changes to master
- Create a new branch called math and switch to this branch
- Use the `status` command to validate that you are on the correct branch.
- Create an add, subtract, multiply, divide, and exponentiate method.
- Overload at least two of the method.
- Add and commit these changes.
- Return to the master branch. Create a new branch called concat and switch to it.
- Add a method that concatenates multiple strings. Overload this method.
- Use `git pull origin master` to ensure your branch is up to date with the master on GitHub.
- Add, commit, and push your changes using `git push origin concat`.
- On github, create a pull request and merge it.
- Locally, switch to your master branch and use `pull` to update it. With the changes.
- Switch to your math branch. Determine the best way to create a pull request and merge it into master.

Stand Up!



Review



Stay Seated & Take 3 Deep Breaths.

RELAX.

Now take a short walk. Clear your head. After a few minutes break, quickly review your notes.
We'll start back shortly.



Lab Time

LabTime

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Work together but INDEPENDENTLY write your own code to complete all of the goals below.

Goals:

- Complete all the katas listed in the activity file.

If this is tough, great! You're getting practice.

If this is easy, great! Help your buddies.

We'll be circulating to provide individual help where it's needed.



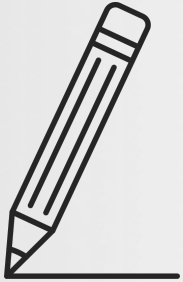
**40
minutes!**

lunch.

Service Layer

Service Layer

THE BRAINS OF THE OPERATION



Notebooks Ready? It's time for a brief lecture.



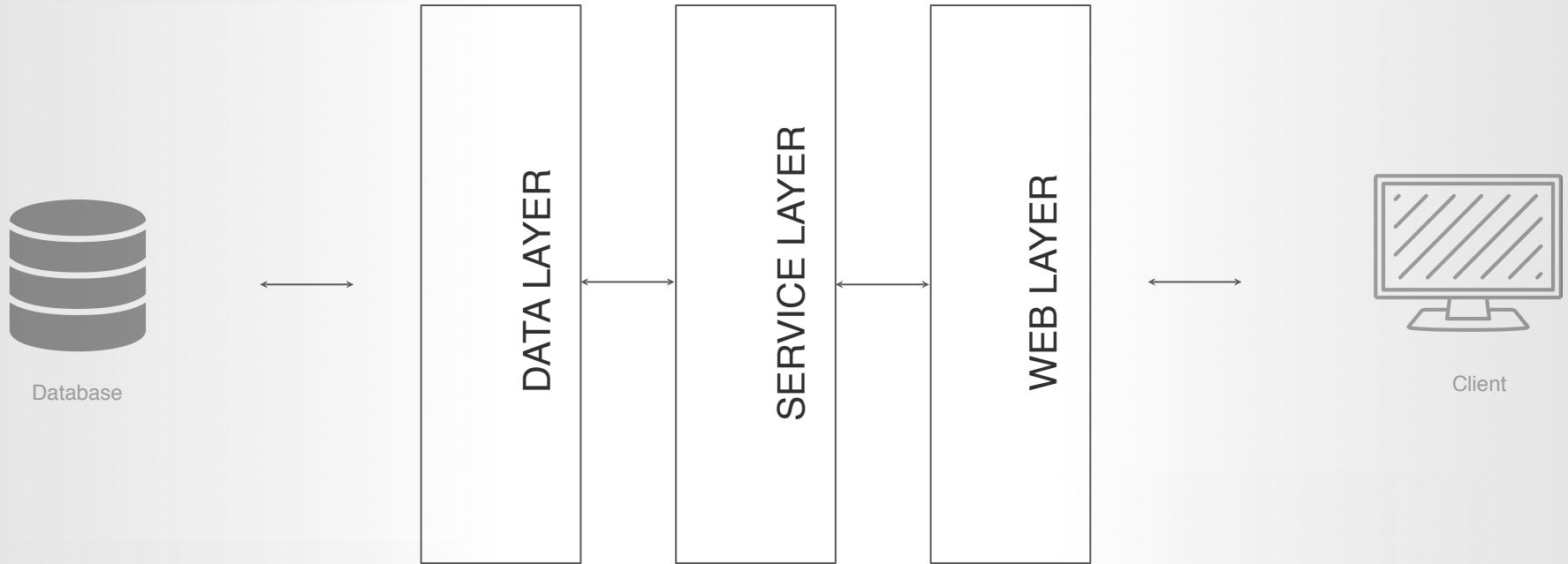
Let's take a moment to recall the layers of a server...



Server Structure

ANATOMY OF A SERVER

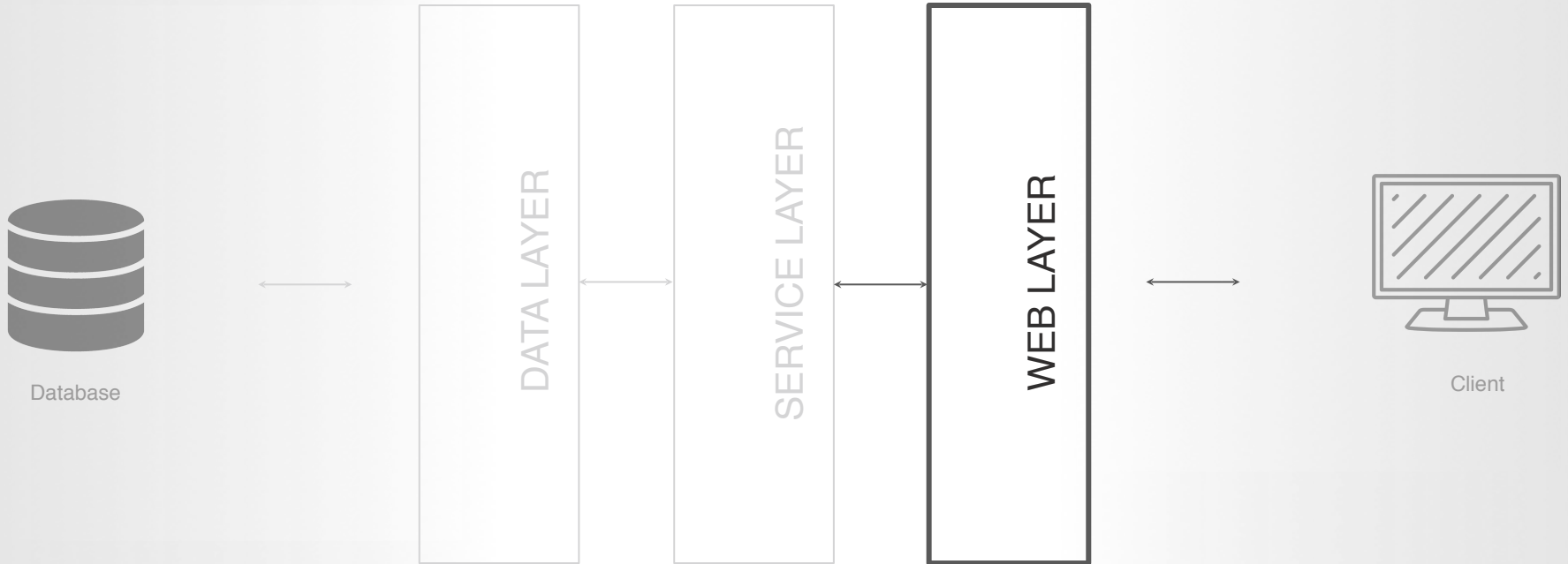
As Java developers, we're focussed on building servers. Spring servers have 3 parts:



Server Structure

WEB LAYER

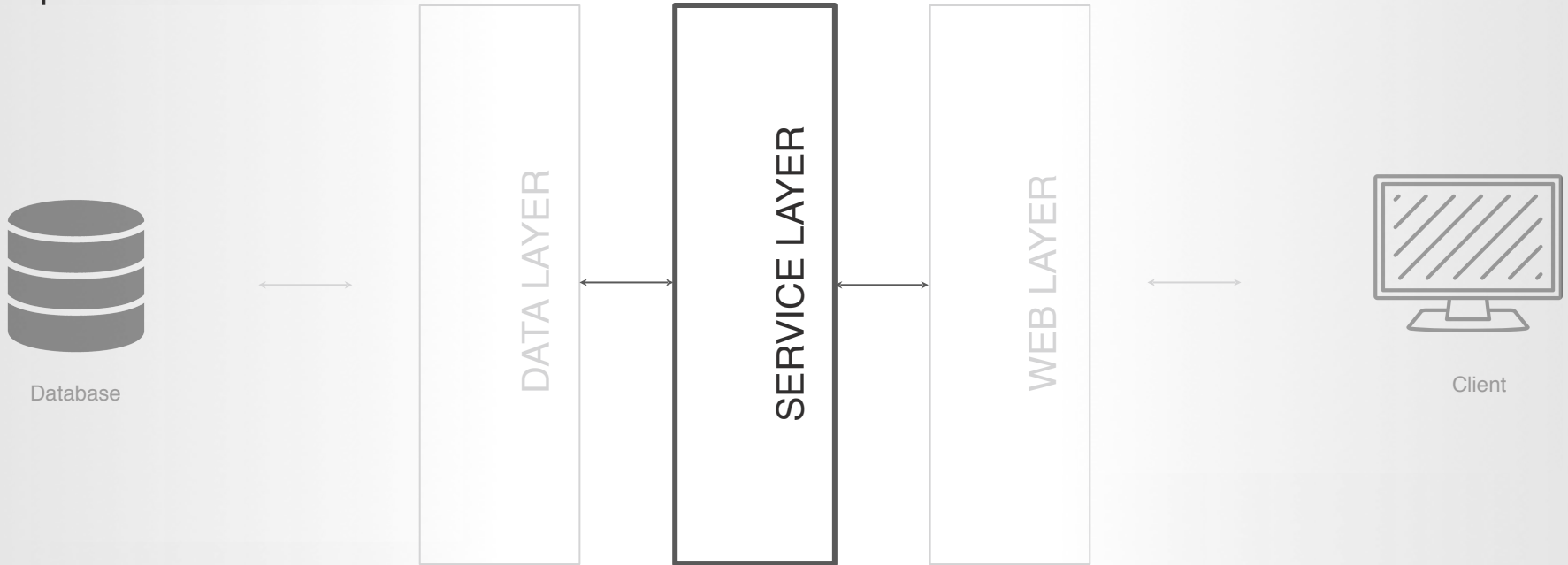
The web layer is responsible for processing requests from the client and sending responses.



Server Structure

SERVICE LAYER

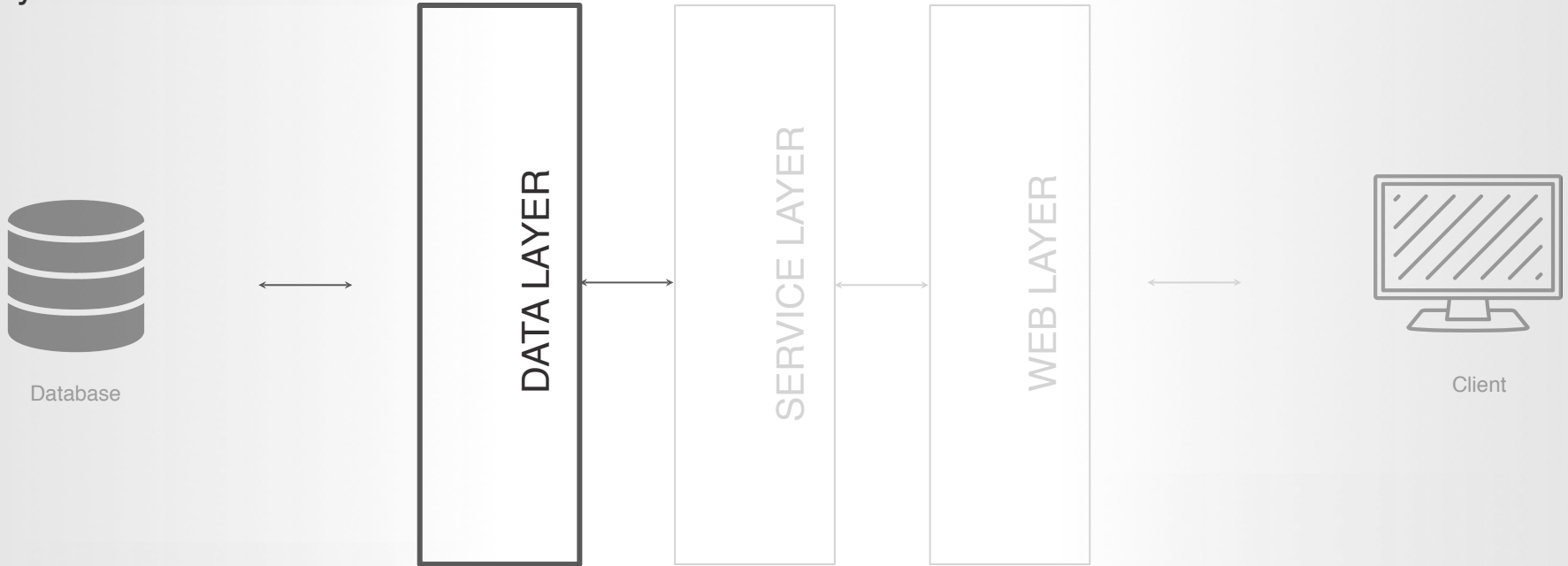
The service layer is responsible for any logic that needs to be performed on the request or the response.



Server Structure

DATA LAYER

The data layer is responsible for getting data from the database and sending it to the service layer.

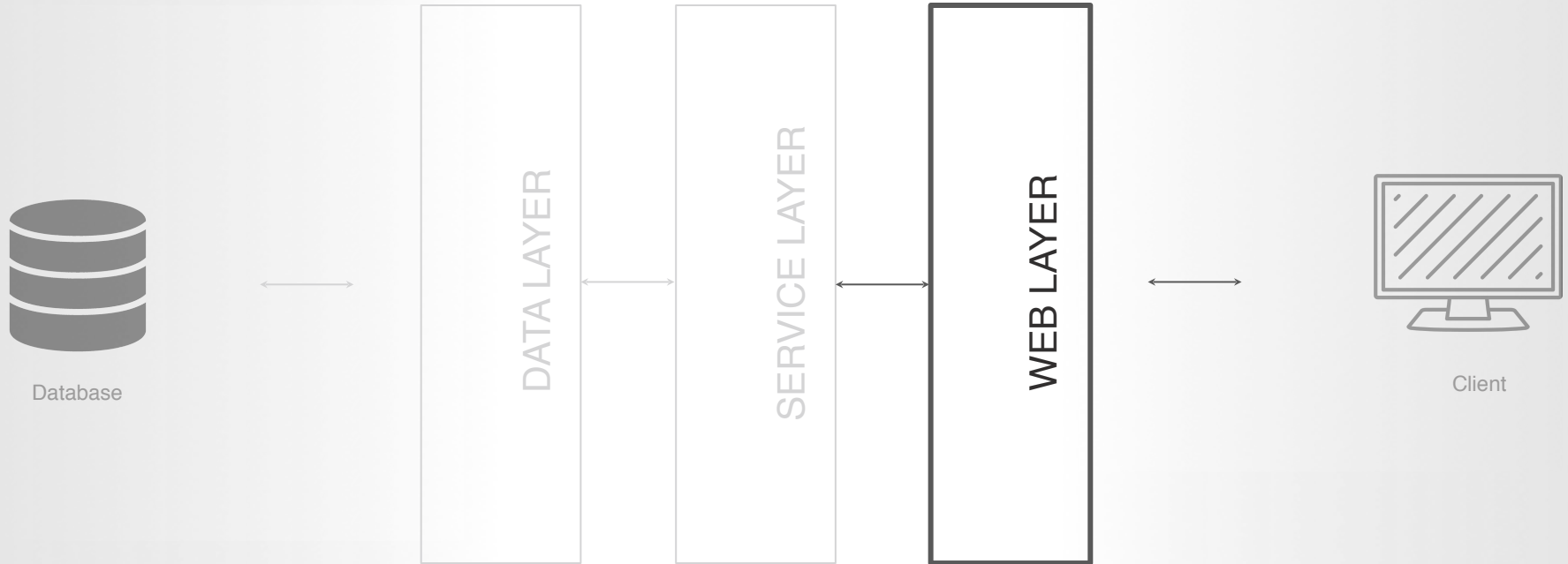


Let's look at a more concrete example. Suppose we are back in Netflix and Vickie clicks on the recommendations page



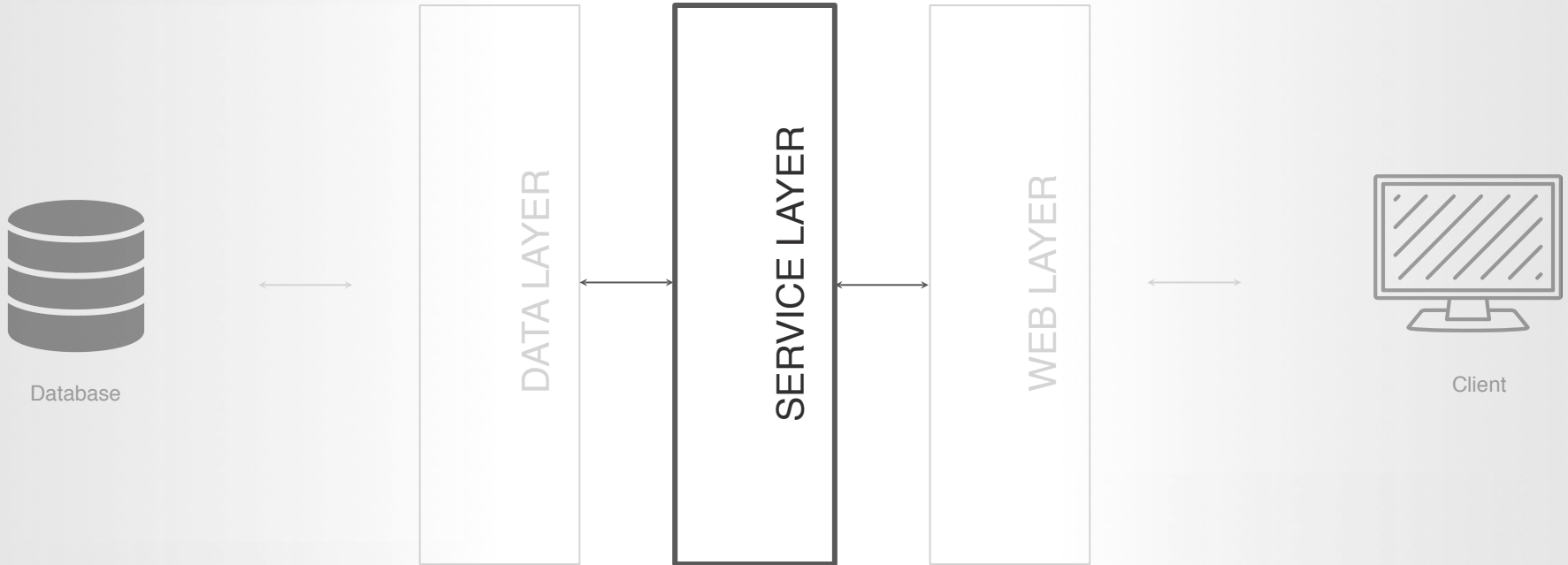
Server Structure

The web layer is responsible for receiving the request for recommended movies and calling the proper method in the service layer, then it waits to get a response from the service layer and sends this response back to Vickie's computer (client).



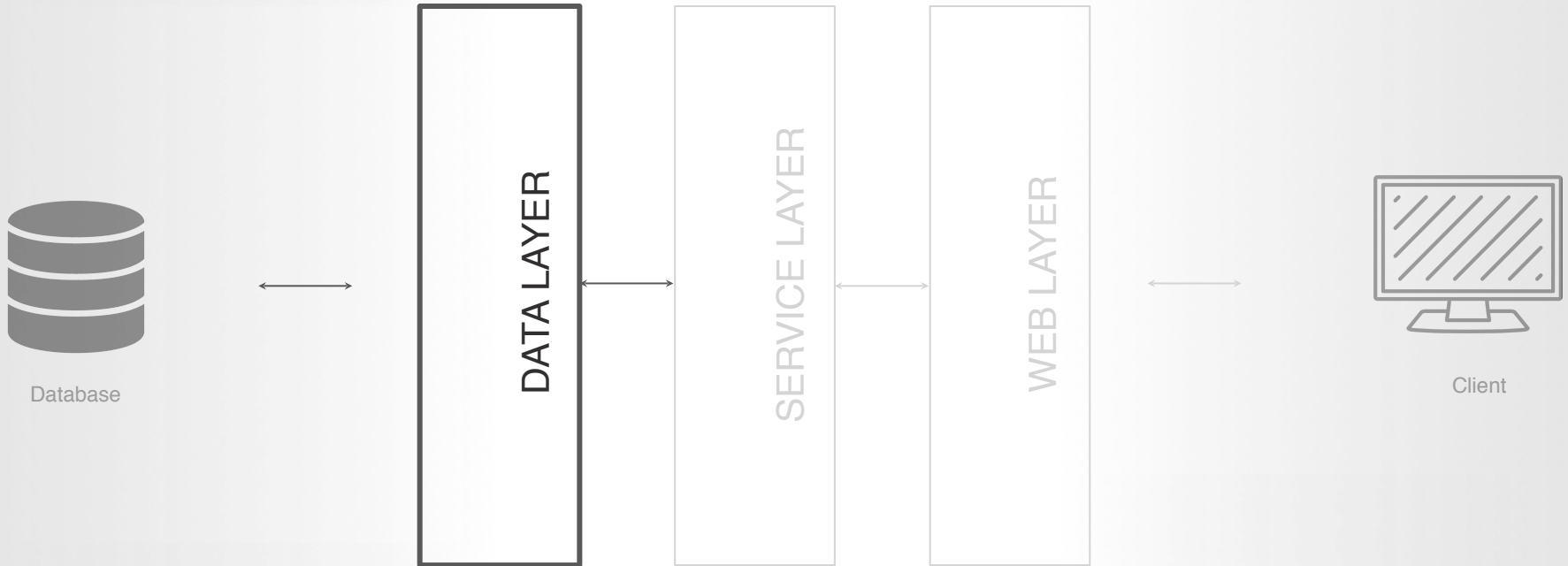
Server Structure

The service layer has a method that was called by the web layer. It's first job is to call a method from the data layer that gets all of Vickie's previous ratings.



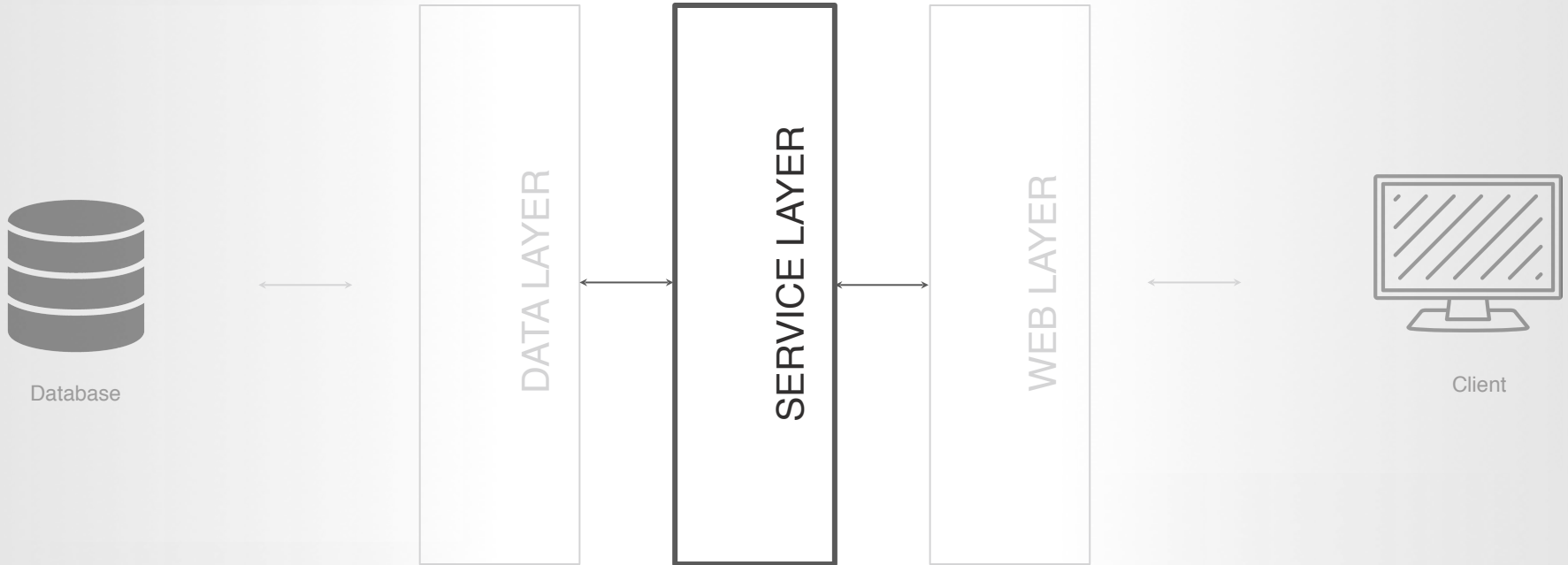
Server Structure

The data layer has a method that was called by the service layer. It's job is to ask the database for Vickie's data. It then returns this data to the service layer.



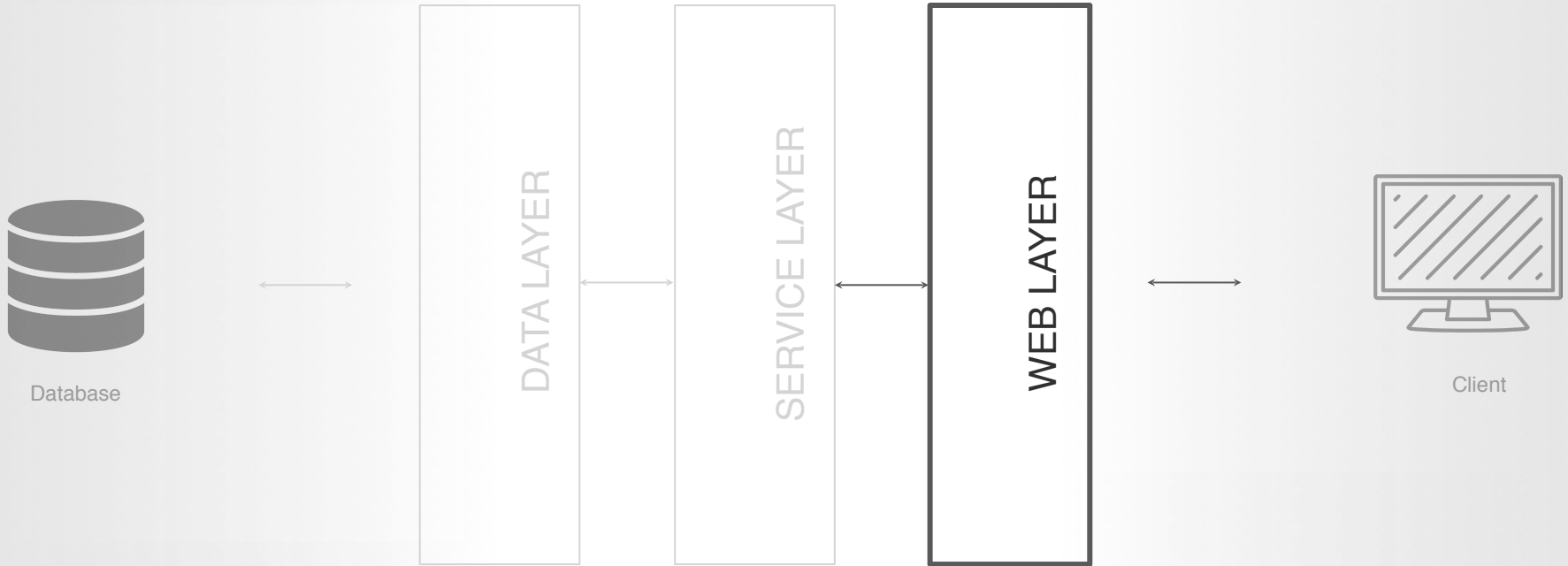
Server Structure

Now that the service layer has Vickie's data, it uses some complex algorithm to figure out which movie to recommend next. Once it knows the new movie list, it returns this list to the web layer. The service layer is kind of the brains of the operation.



Server Structure

The web layer now has the recommended movies and sends this list to Vickie's computer to render visually, so that Vickie can select a movie to watch.



Service Layer

THE BRAINS OF THE OPERATION


We keep talking about separate layers in a server. What do we mean by a layer?

A layer just means a piece of our application that functions with some level of independence. For a small project, just keeping these things (controllers, DTOs, logic, etc) in separate classes is sufficient.

Your data layer (DTO and associated repository) are in different directories from your controllers (web layer). This creates a higher degree of association between some classes than others and helps us divide our application conceptually into “layers”.

On a larger project, the web layer, service layer, and data layer may all exist on different physical or virtual machines.

“Layer” is really a conceptual term that means that each of these components (controllers, logic, data transfer) should be as autonomous as possible, interfacing with other layers only through interfaces.



Service Layer

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Turn to your neighbor

Goals:

- In 2 minutes, write down 4 reasons that we might want to include a separate layer for logic. Yep 2 minute, 4 reasons.... Ready? GO!



2 minutes!

Service Layer

THE BRAINS OF THE OPERATION

There are a lot of reasons to separate concerns like logic and data transfer into different layers. Here's a scenario that makes one benefit fairly obvious:

A few years ago Netflix decided to hold a competition - \$1M went to the group with the best algorithm for recommending videos. They were going to rip out the existing logic behind their recommendation feature and replace it with this new algorithm.

Netflix has a data layer that likely involves a lot of potential SQL queries, but we can safely assume that get videos by genre, get videos by title, get videos by director, and get videos by cast member are a few methods that exist. These are likely methods used by the recommendations algorithm.

They are also used by other features: the search feature, the “find titles similar to” link, the genre menu, and the “continue watching” section to name a few.



Service Layer

THE BRAINS OF THE OPERATION

There are a lot of reasons to separate concerns like logic and data transfer into different layers. Here's a scenario that makes one benefit fairly obvious:


Suppose Netflix did not separate out their logic layer but instead dropped this new algorithm into their data layer. It's amazing! Users are now 5 times more likely to click on recommended videos. Everything is smooth.

But suddenly, the search and continue watching feature don't work. Why?

Who knows??

We have fundamentally altered a class that many other features rely on and even though they aren't using this algorithm directly, it exists in the class with methods needed by other features and fundamentally alters the complexity of that class. Debugging is suddenly a nightmare.

Losing the search feature is too big of a loss. They are forced to roll back the new recommendations feature. Users are annoyed and there's a 1% subsequent attrition rate increase. All because no one separated the application concerns into separate layers.



Service Layer

THE BRAINS OF THE OPERATION

There are a lot of reasons to separate concerns like logic and data transfer into different layers. Here's a scenario that makes one benefit fairly obvious:

Fortunately, that's not how the real story ends. Netflix employs talented and experienced developers that heavily tested the new algorithm using a battery of integration, unit, and performance tests. They deployed it in its own layer to make it easy to test, monitor, and maintain and easy to mitigate any potential negative impact if any was found.

The new recommendations feature was a boon to Netflix and to computer science.





KeyPoint

The Data, Service, Web layer model works well for small to medium-sized enterprise applications. At a large enough scale, solutions necessarily become more complex and many factors need to be considered in choosing the right architectural approach.



Service Layer

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

```
@Component
public class ServiceLayer {

    @Autowired
    private CustomerRepository customerRepo
    @Autowired
    private NoteRepository noteRepo;

    public Customer addCustomer(Customer customer) {
        customerRepo.save(customer);
        return customer;
    }

    public void updateCustomer(Customer customer, int id) {
        if (customer.getId() != id) {
            throw new IllegalArgumentException("Customer id must match id provided");
        }

        customerRepo.save(customer);
    }
}
```

Service Layer

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

```
@RestController
public class CustomerController {

    @Autowired
    private ServiceLayer service;

    @RequestMapping(value="/customers", method = RequestMethod.GET)
    public List<Customer> getAllCustomers() {
        return service.getAllCustomers();
    }

    @RequestMapping(value="/customers", method = RequestMethod.POST)
    public Customer createCustomer(@RequestBody Customer customer) {
        service.addCustomer(customer);
        return customer;
    }
}
```

Service Layer

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Work in PAIRS to complete all of the goals below.

Goals:

- Add an additional method to the ServiceLayer called `getCustomersByStartingLetter`
- In this method, return a List of customers whose last name starts with the letter provided.
- Add an additional endpoint at `/customers/lastname/{letter}` which invokes this new method and returns the list to the client.



**20
minutes!**



Stay Seated & Take 3 Deep Breaths.

RELAX.

Now take a short walk. Clear your head. After a few minutes break, quickly review your notes.
We'll start back shortly.



Lab Time

LabTime

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Work together but INDEPENDENTLY write your own code to complete all of the goals below.

Goals:

- Complete all the katas listed in the activity file.

If this is tough, great! You're getting practice.

If this is easy, great! Help your buddies.


We'll be circulating to provide individual help where it's needed.



**40
minutes!**

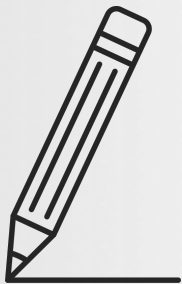
Sneak Peek

Note that next class will start with the sneak peek topic. You are NOT expected to master this today.



Lambdas & Streams

THE HOT NEW SYNTAX



Notebooks Ready? It's time for a brief lecture.



Lambdas & Streams

THE HOT NEW SYNTAX

Lambdas, streams, and aggregate operations are relatively new to Java. They were introduced in Java 8.

The real power for everyday tasks is when streams and aggregate operations are used with lambdas. This combination gives developers some great options for processing collections of data.



Lambdas & Streams

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

```
List<Television> tvList = tvRepo.findAll();

// All TVs with screens larger than 60 inches
tvList.stream()
    .filter(t -> t.getScreenSize() > 60)
    .forEach(tv -> {
        System.out.println("=====");
        System.out.println("Brand: " + tv.getBrand());
        System.out.println("Model: " + tv.getModel());
        System.out.println("Screen Size: " + tv.getScreenSize());
        System.out.println("Price: " + tv.getPrice());
    });
```

Wrap Up

Module 4 Lesson 3

HOMEWORK

You don't have to submit your nightly homework, but you are expected to complete it.

Complete any remaining katas.

