24-8-23 => Always used the Wrapper class in the Model class.

14 MVC ARCHITECTURE

=> The model-view-controller (MVC) is well-known design pattern in the web development field. It is way to organize our code. It specifies that a program or application shall consist of data model, presentation information and control information.

=> The MVC pattern architecture consists of the three layers →

- Model → It represents the business layer of applicatio It is an object to carry the data that can also contain the logic to update controller if data is changed.

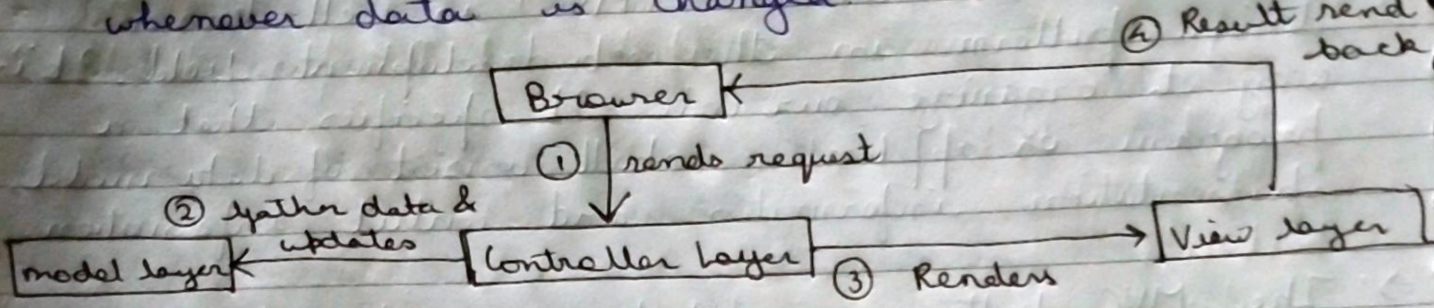
=) The Model class is simple POJO, represents the table of the database. The ~~clas~~ name of the model class and the table in the database is the same, but the name of the table is in the plural.

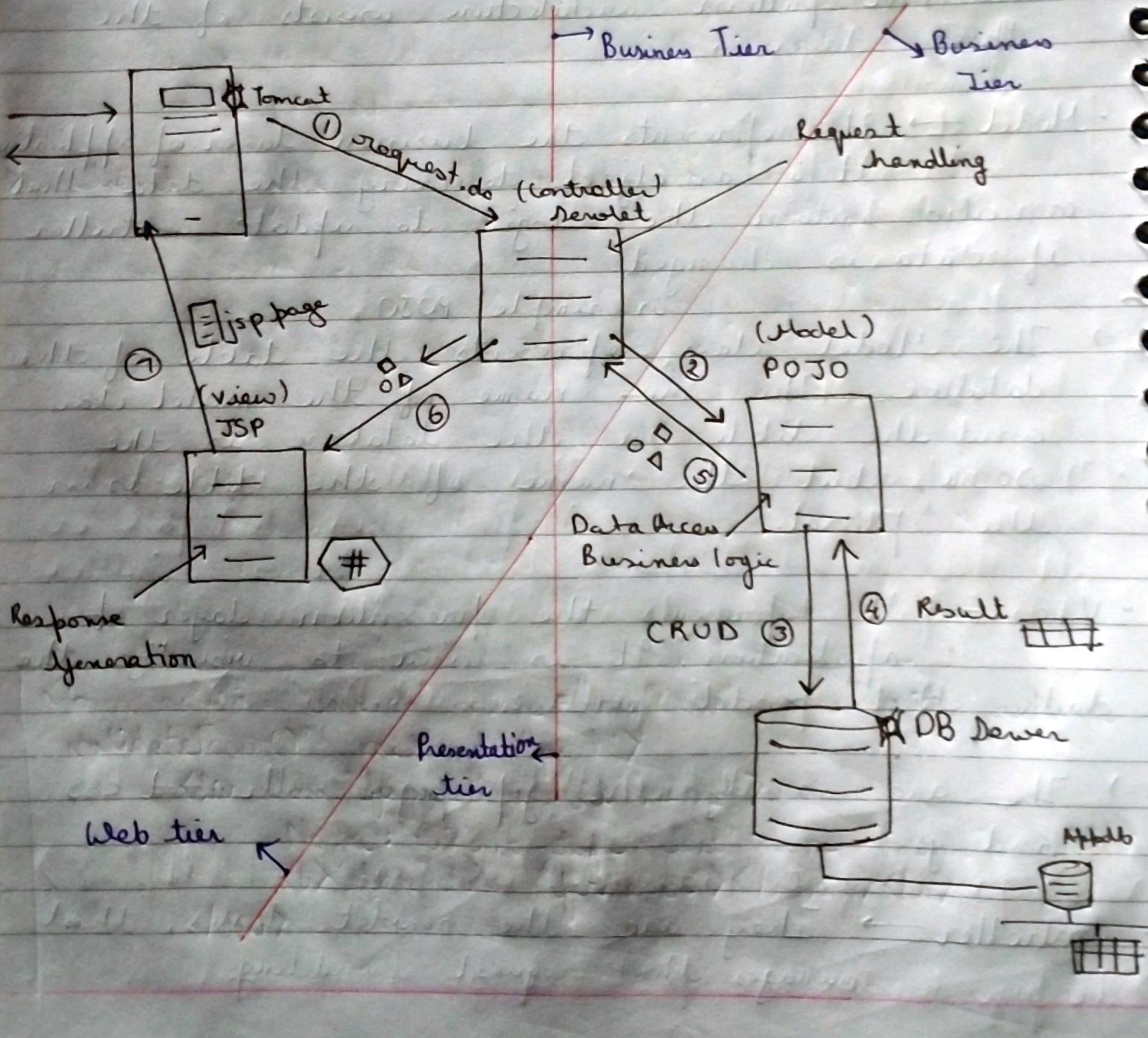
- View → It represents the presentation layer of application. It is used to visualize the data that model contains.

=) The are usually the jsp pages with ~~and~~ css and js.

- Controller → They are the servlet classes, that manages the request handeling.

⇒ It works on both the model and view. It is used to manage the flow of application, i.e. data flow in the model object and to update the view whenever data is changed.





=) From ~~never~~ this time, we will never write the path of html or jsp in the <a> tag.

=) We will only write the fake URL in the <a> tag.

• About the vertical line ⟶ The vertical lines denote that the (Controller) Servlet differentiate betn the Presentation layer & the business layer.

=) The request from the client, get directed to the servlet and it decides to whether transfer the flow to the prentation ~~layer~~ tier or to the ~~view layer~~ business tier.

=) The Servlet (Controller) separates the presentatio ~~layer~~ tier and the ~~view~~ ~~view tier~~ business tier.

• About the slant line ⟶ The slant line denotes the separation of business tier & ~~tt~~ web tier.

=) The business tier donot contain a single element of web component ~~line~~ like Servlet or JSP.

=) The idea is that for different devices (like desktop, TV, mobile, ATM), the business tier will remain same, we only have to design the presentation tier.

=) This will save the money & energy of writing the code of business tier again & again.

VALIDATION OF THE FORM →

=) When we enter data, the browser and/or the web server will check to see that the data is in the correct format and within the constraints set by the application.

=) Validation done in the browser is called client-side validation, while validation done ~~in~~ on the server is called server-side validation.

———————— × ———————— × ————————

★ Client-Side Validation ⟶ There is two options are available for the client side validation;

(1) In-built form Validation
(2) Javascript Validation.

=) We will be focusing mostly on the Javascript validation. We will ~~expl~~ understand it using an example.

```
<form action="save.do" name="myForm" onsubmit="return
validateForm()" method="post">
        <div class="formdesign" id="name">
                Name: <input type="text" name="fname" required>
                <b><span class="formerror"></span></b>
        </div>
```

```
<div class="formdesign" id="email">
        Email: <input type="~~text~~ email" name="femail"
        required><b><span class="formerror">
        </span></b>
</div>

<div class="formdesign" id="phone">
        Phone: <input type="phone" name="fphone"
        required><b><span class="formerror"></span></b>
</div>

<div class="formdesign" id="pass">
        Password: <input type="password" name="fpass"
        required><b><span class="formerror"></span></b>
</div>

<div class="formdesign" id="cpass">
        Confirm Password: <input type="password" name "fcpass
        required><b><span class="formerror"></span></b>
</div>

<input class="but" type="submit" value="Submit">
</form>
```

to index.js

```
function clearErrors() {
        errors = document.getElementsByClassName("formerror");
        for(let item of errors)
        {
            item.innerHTML = "";
}       }
```

```
function setError(id, error){
    var element = document.getElementById(id);
    element.getElementsByClassName('formerror')[0].innerHTML
    = error;
}

function validateForm(){
    var returnVal = true;
    clearErrors();

    var name = document.forms['myForm']["fname"].value;
    var email = document.forms['myForm']["femail"].value;
    var phone = document.forms['myForm']["fphone"].value;
    var password = document — " — " — ["fpass"].value;
    var cpassword = — " — " — " — ["fcpass"].value;

    var regEmail = /^\w+([\.-]?\w+)*@\w+([\.-]?
    \w+)*(\.\w{2,3})+$/g;
    var regPhone = /^\d{10}$/;
    var regName = /\d+$/g;

    if(name == "" || regName.test(name)) {
        setError("name", "Enter proper name");
        name.focus();
        returnVal = false;
    }

    if(email == "" || regEmail.test(email)){
        setError("email", "Enter proper email");
        email.focus();
        returnVal = false;
    }
```

/* similarly we make if condition for */
all the input fields

———— x ———— x ————

* Server side Validation ⟶ ~~The~~ We also have to validate the data entered by the client at the server side

=) Because sometime, the potential client can ditch the javascript validation, and can directly access the server. ~~to~~ ~~if~~ To handle this, we have to validate the inputs

=) To validate the input ~~for~~ values, in the server side (JAVA), we will use the Pattern Matcher.

=) We will get the parameter in the servlet and then validate it using the Regex Pattern Matcher

=) If Any of the Matching fails, we will set the error message, as attribute, and we can display the error message in the next jsp ~~page~~ page

———— x ———— x ————

10-9-23

16_FOREIGN KEY IN MODAL CLASS→

∴ Suppose, we have two tables in ~~the~~ our database; i.e. table User and table Product.

=> The primary key of user table is a foreign in the product table.

[user_id   user_name   user_email   user_password
└→ ~~User~~ user table

[product_id   user_id   product_name   product_price.
└→ product table.

• How we will define foreign key in modal class→

=> In the modal class, the foreign key of a particular class is defined by the object of that modal class.

Product.java

```
package models;
~~import mod~~
public class Product {

    private int productId;
    private User user;
    private String productName;
private int productPrice;

public void setUser (User user){
        this.user = user;
}

public User getUser(){
        return user;
}
```

:) We don't write a simple primitive variable of or the Wrapper class reference variable for the foreign key in the model class.