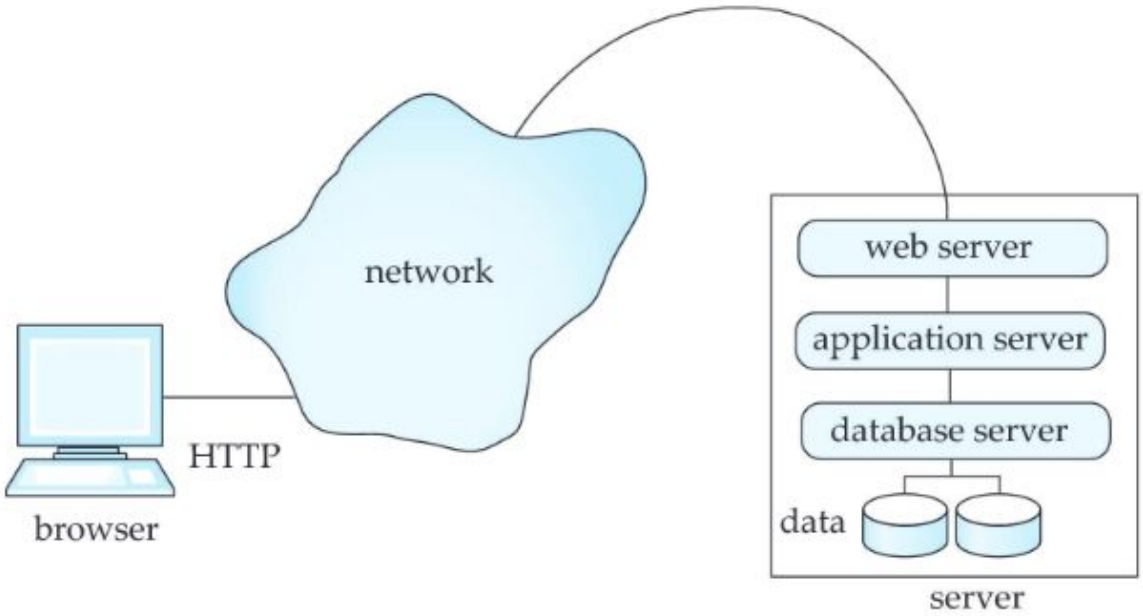
Web Application Development

Application Program Design

- Most database users do *not* use a query language like SQL
- An application program acts as the intermediary between users and the database
  - Applications split into
    - front-end
    - middle layer
    - backend
- Front-end: user interface
  - Forms
  - Graphical user interfaces
  - Many interfaces are Web-based

Three Layer Web Architecture



Server Side Scripting

- Server-side scripting simplifies the task of connecting a database to the Web
  - Define an HTML document with embedded executable code/SQL queries.
  - Input values from HTML forms can be used directly in the embedded code/SQL queries.
  - When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document.
- Numerous server-side scripting languages
  - JSP, PHP
  - General purpose scripting languages: VBScript, Perl, Python

Java Server Pages (JSP)

- A JSP page with embedded Java code

```
<html>
<head> <title> Hello </title> </head>
<body>
<% if (request.getParameter("name") == null)
{ out.println("Hello World"); }
else { out.println("Hello, " + request.getParameter("name")); }
%>
</body>
</html>
```

- JSP is compiled into Java + Servlets
- JSP allows new tags to be defined, in tag libraries
  - Such tags are like library functions, can are used for example to build rich user interfaces such as paginated display of large datasets

PHP

- PHP is widely used for Web server scripting
- Extensive libaries including for database access using ODBC

```
<html>
  <head> <title> Hello </title> </head>
  <body>
  <?php if (!isset($_REQUEST[ 'name' ]))
  { echo "Hello World"; }
  else { echo "Hello, " + $_REQUEST[ 'name' ]; }
  ?>
  </body>
</html>
```

Javascript

- Javascript very widely used
  - Forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- Javascript functions can
  - Check input for validity
  - Modify the displayed Web page, by altering the underling **document object model (DOM)** tree representation of the displayed HTML text
  - Communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
    - Forms basis of AJAX technology used widely in Web 2.0 applications
    - E.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

Javascript

- Example of Javascript used to validate form input

```
<html> <head>
    <script type="text/javascript">
        function validate() {
            var credits=document.getElementById("credits").value;
            if (isNaN(credits)|| credits<=0 || credits>=16) {
                alert("Credits must be a number greater than 0 and less than 16");
                return false
            }
        }
    </script>
</head> <body>
    <form action="createCourse" onsubmit="return validate()">
        Title: <input type="text" id="title" size="20"><br />
        Credits: <input type="text" id="credits" size="2"><br />
        <Input type="submit" value="Submit">
    </form>
</body> </html>
```

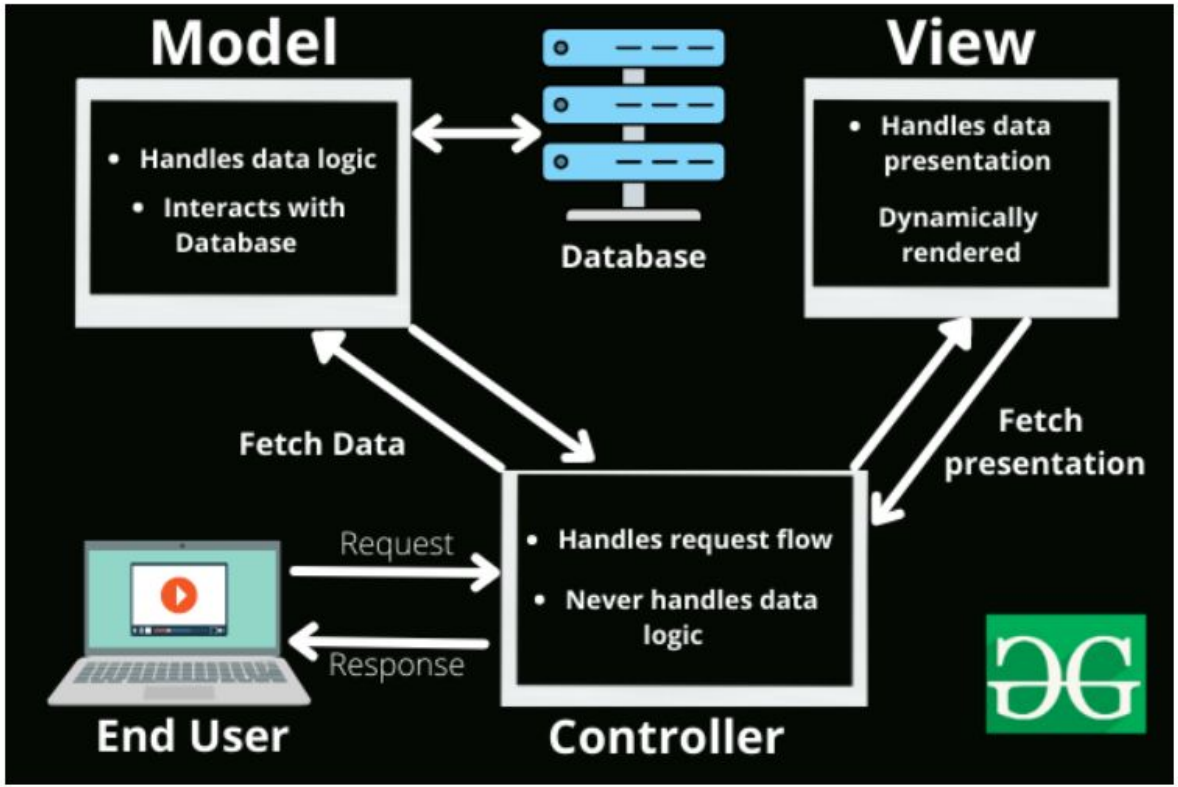
Application Architecture

- Application layers
  - Presentation or user interface
    - **model-view-controller (MVC)** architecture
      - **model**: business logic
      - **view**: presentation of data, depends on display device
      - **controller**: receives events, executes actions, and returns a view to the user
  - **business-logic** layer
    - provides high level view of data and actions on data
      - often using an object data model
    - hides details of data storage schema
  - **data access** layer
    - interfaces between business logic layer and the underlying database
    - provides mapping from object model of business layer to relational model of database

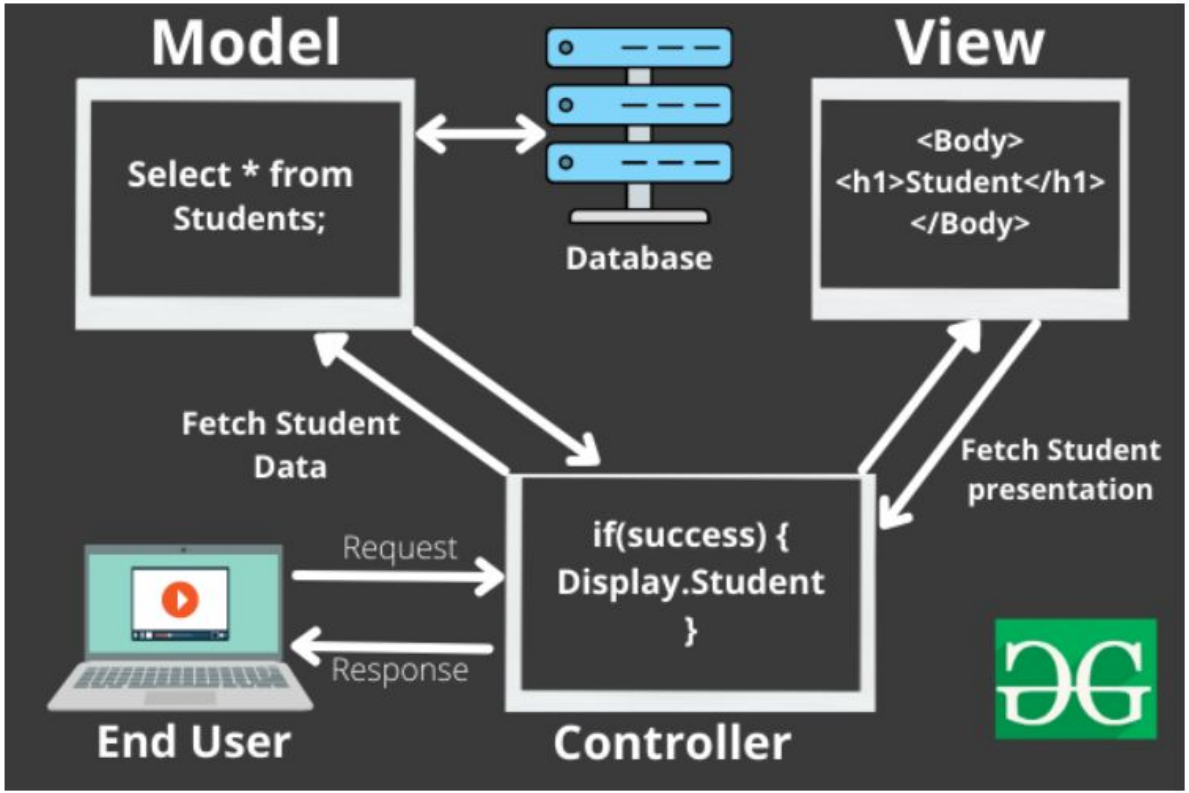
Business Logic Layer

- Provides abstractions of entities
  - E.g., students, instructors, courses, etc
- Enforces **business rules** for carrying out actions
  - E.g., student can enroll in a class only if she has completed prerequsites, and has paid her tuition fees
- Supports **workflows** which define how a task involving multiple participants is to be carried out
  - E.g., how to process application by a student applying to a university
  - Sequence of steps to carry out task
  - Error handling
    - E.g. what to do if recommendation letters not received on time

MVC


Model
Handles data logic
Interacts with Database
Database
View
Handles data presentation
Dynamically rendered
Fetch Data
Fetch presentation
Request
Response
Handles request flow
Never handles data logic
End User
Controller

MVC Example


Model
Select * from Students;
Database
View
<Body>
<h1>Student</h1>
</Body>
Fetch Student Data
Fetch Student presentation
Request
Response
if(success) {
Display.Student
}
End User
Controller

Popular MVC Platforms

- Ruby on Rails
- Django
- CherryPy
- Spring MVC
- Catalyst
- Rails
- Zend Framework
- Fuel PHP
- Laravel
- Symphony

Improving Server Performance

- Performance is an issue for popular Web sites
  - May be accessed by millions of users every day, thousands of requests per second at peak time
- Caching techniques used to reduce cost of serving pages by exploiting commonalities between requests
  - At the server site:
    - Caching of JDBC connections between servlet requests
      - a.k.a. **connection pooling**
    - Caching results of database queries
      - Cached results must be updated if underlying database changes
    - Caching of generated HTML
  - At the client's network
    - Caching of pages by Web proxy

Authorization

- We may assign a user several forms of authorizations on parts of the database.
  - **Read** - allows reading, but not modification of data.
  - **Insert** - allows insertion of new data, but not modification of existing data.
  - **Update** - allows modification, but not deletion of data.
  - **Delete** - allows deletion of data.
- Each of these types of authorizations is called a **privilege**. We may authorize the user all, none, or a combination of these types of privileges on specified parts of a database, such as a relation or a view.

Application Level Authorization

- Current SQL standard does not allow fine-grained authorization such as "students can see their own grades, but not other's grades"
  - Problem 1: Database has no idea who are application users
  - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

```
create view studentTakes as
select *
from takes
where takes.ID = syscontext.user_id()
```

  - where syscontext.user_id() provides end user identity
    - End user identity must be provided to the database by the application
  - Having multiple such views is cumbersome

Authorization in SQL

- The **grant** statement is used to confer authorization

  grant <privilege list> **on** <relation or view > **to** <user list>

- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Example:
  - **grant select on** *department* **to** Amit, Satoshi
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges

- **select**: allows read access to relation, or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

    grant select on *instructor* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

Roles

- A **role** is a way to distinguish among various users as far as what these users can access/update in the database.
- To create a role we use:

  create a role <name>
- Example:
  - **create role** instructor
- Once a role is created we can assign “users” to the role using:
  - **grant** <role> **to** <users>

End