EACOMM CORPORATION

TESDA Web Development (Backend) NC III

Review Session-1

Course Objectives

Course Objectives

By the end of this 10-day review, participants will be able to:
Install, configure, and utilize server software applications (LAMP, Node.js, etc.).
Design and implement databases using SQL, normalization, and ERDs.
Develop and deploy RESTful APIs integrated with frontend applications.
Prepare confidently for the **TESDA NC III Certification Exam**.

Learner Expectations

Participants are expected to:

- Attend and participate actively in our sessions
- Complete **asynchronous tasks and quizzes** on time.
- Maintain **academic integrity and professionalism** throughout the course.
- Submit a **fully functional backend project** as a final output.

Each day will be divided into Face to Face Classes (Morning) and Asynchornous Tasks (Afternoon)

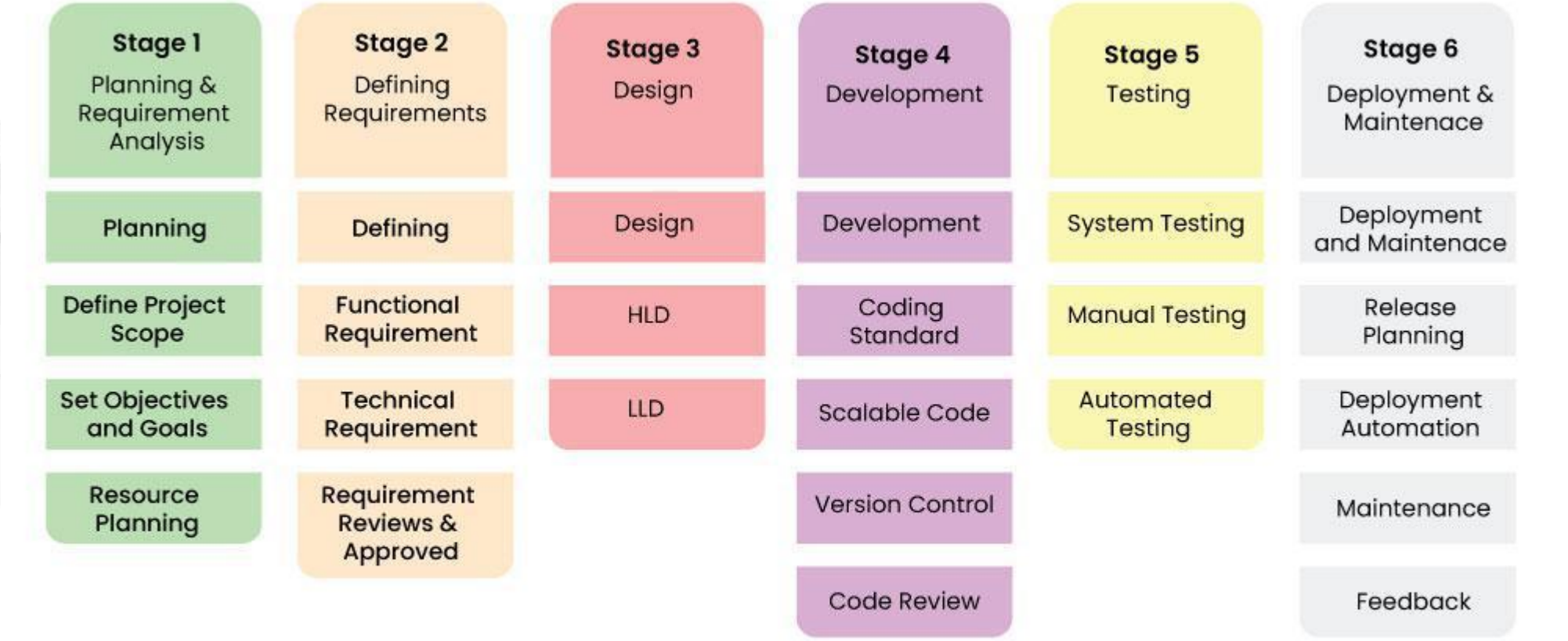
Course Outline

Day	Mode	Core Competency Reference	Synchronous Activity (Lecture / Lab)	Asynchronous Task (Independent Work)
Day 1	Lecture	**Utilize Software Methodologies for Back-End Systems** **Utilize Server Software Applications (Overview)**	• Course orientation and NC III overview • Review of SDLC models (Waterfall, Agile, RAD) • Choosing appropriate methodologies for backend development • Overview of common backend stacks (LAMP, Node.js, Python)	**Essay:** "Create a Project Plan"
Day 2	Lab	**Utilize Server Software Applications (Installation)**	• Install and configure local server stack (XAMPP/LAMP/WAMP or Node.js) • Setup database (MySQL / PostgreSQL) • Install Git and initialize repository **Exercise:** "Hello Server" script returning current time + user agent	**20-item Quiz** on Server-side Scripting Language of choice (PHP or Node)
Day 3	Lecture	**Develop Website Backend Systems (3.1–3.4)**	• Backend project planning (requirements, flowchart, Gantt/Kanban) • Review server + cloud deployment concepts • Review DB connectivity (PDO/ORM) • SQL CRUD / Normalization	**Worksheet:** Create an ERD for a sample system **Review** uploaded resources on MySQL databases
Day 4	Lab	**Database Design & Implementation**	• Create schema + relationships • Implement CRUD tables and data seeding • Indexing and joining tables • Test DB connections **Exercise:** "Company DB" (companies, employees, departments)	**20-item Quiz** on SQL queries, functions, etc.

Course Outline

Day 5	Lab	**Backend Data Processing**	• Write scripts for data computation & report generation • Add validation and error handling **Exercise:** "Department Summary" – compute employee counts and averages	**Review** upload resources on APIs
Day 6	Lecture	**Develop APIs (3.5–3.6)**	• Overview of MVC architecture • Create RESTful API endpoints (CRUD)	**20-item** Quiz on APIs
Day 7	Lab	**API Data Retrieval & Integration**	• Extend API with search/filter endpoints • Test APIs via Postman • Apply authentication (API keys/tokens) **Exercise:** Create API endpoints for writing and reading to DB based on previous exercises.	**Review** Capstone Project and **start working** on it.
Day 8	Lab	**Backend System Integration (Capstone)**	• Integrate backend with a given frontend Implement complete data flow (frontend → API → DB → output)	**Complete** Capstone Project
Day 9	Presentation + Critique Session	**Integrative Competency Review**	• Students present their completed backend projects • Live demo: frontend↔API↔database interaction • Peer and instructor feedback on code structure, security, and performance	**Review** for Institutional Assessment
Day 10	Lecture + Exam	**Review & Institutional Assessment**	• Review of all core competencies • 50 item written practice exam	

The Software Development Life Cycle



6 Stages of Software Development Life Cycle

https://www.geeksforgeeks.org/software-engineering/software-development-life-cycle-sdlc/

Software Development Methodologies

EACOMM CORPORATION

Methodology	Description	Pros	Cons	Best Use Case
Waterfall	Sequential model with clear phases: requirements → design → implementation → testing → deployment.	Simple, easy to manage; clear documentation; works for fixed requirements.	Hard to change once a phase is done; issues found late in process.	Best for stable, well-defined projects (e.g., government systems, static sites).
Rapid Application Development (RAD)	Emphasizes fast prototyping and user feedback through iterative cycles.	Fast development; early user feedback; flexible.	Needs skilled team; not ideal for large or complex projects.	Quick prototypes or MVPs (e.g., startup apps, internal tools).
Spiral	Combines iterative development with strong risk analysis at every phase.	Good risk management; flexible; early error detection.	Complex; costly; requires experienced developers.	Large, high-risk projects (e.g., enterprise or defense systems).
Agile	Iterative, flexible, and collaborative approach with short sprints and user feedback.	Adaptable; encourages teamwork; delivers usable software quickly.	Requires ongoing client input; unpredictable cost/timeline.	Dynamic, evolving projects (e.g., web platforms, SaaS applications).

Choosing a Methodology

EACOMM CORPORATION

Factor	What to Evaluate	Suggested Methodologies
Project Size & Complexity	Is the system small and simple or large and multi-layered?	Small: RAD, Agile Large: Spiral, Waterfall
Requirements Stability	Are requirements fixed or changing frequently?	Fixed: Waterfall Changing: Agile, RAD
Timeline & Delivery Speed	Is there a strict deadline or need for quick prototyping?	Tight deadline: RAD Flexible timeline: Agile
User Involvement	Will clients/users provide frequent feedback?	High: Agile, RAD Low: Waterfall
Risk Level	Does the project involve high uncertainty or innovation?	High: Spiral Low: Waterfall
Team Expertise	Is the team experienced in collaboration and iteration?	Experienced: Agile, Spiral New team: Waterfall

Guideline Summary:

- Waterfall → Best for clear, stable projects.
- RAD → Best for rapid prototypes and MVPs.
- Agile → Best for collaborative, evolving projects.
- Spiral → Best for large, high-risk projects needing analysis.

Creating a Project Schedule

EACOMM CORPORATION

Step	Description
1. Define Project Tasks	Break down the project into smaller, manageable activities. Taking into account **SDLC** and **Software Methodology** used.
2. Estimate Time and Resources	Assign realistic durations and required manpower or tools.
3. Identify Task Dependencies	Determine which tasks must be completed before others can start.
4. Sequence the Tasks	Arrange tasks logically according to dependencies and workflow.
5. Assign Responsibilities	Assign tasks to team members or departments.
6. Set Milestones	Identify major deliverables or checkpoints (e.g., “Prototype Complete”).
7. Create a Timeline or Gantt Chart	Plot tasks visually to monitor progress and deadlines.
8. Review and Adjust Regularly	Track progress, update schedules, and resolve delays promptly.

Example Waterfall Project Schedule

EACOMM CORPORATION

Phase	Duration	Objectives	Key Tasks / Deliverables
1. Requirements Analysis	Week 1–2	Understand project goals and user needs	• Gather and document requirements • Define scope and system specifications • Approve Software Requirement Specification (SRS)
2. System & Database Design	Week 3–4	Create technical design of the system	• Design architecture and data flow diagrams • Create Entity Relationship Diagram (ERD) • Prepare database schema and UI mockups
3. Implementation (Coding)	Week 5–8	Develop and integrate modules	• Develop backend logic and database connections • Code APIs and server scripts • Conduct internal testing per module
4. Testing & Debugging	Week 9–10	Validate system functionality and quality	• Perform unit and integration testing • Fix bugs and performance issues • Conduct quality assurance review
5. Deployment	Week 11	Deliver final system to client	• Set up production environment • Upload database and system files • Conduct user acceptance testing (UAT)
6. Maintenance & Support	Ongoing	Sustain performance and resolve issues	• Monitor system behavior • Apply patches and updates • Provide technical support

Example RAD Project Schedule

EACOMM CORPORATION

Phase	Duration	Objectives	Key Tasks / Deliverables
1. Requirements Planning	Week 1	Define broad goals and user requirements	• Conduct quick client interviews • Identify system scope and objectives • Draft preliminary project plan and timeline
2. User Design (Prototyping)	Week 2–3	Build and refine prototypes with user feedback	• Create wireframes and UI mockups • Develop functional prototypes (frontend + sample backend) • Gather feedback and revise quickly
3. Construction (Rapid Development)	Week 4–5	Convert prototypes into working applications	• Develop full backend logic and database • Integrate APIs and user authentication • Conduct iterative testing with client feedback
4. Cutover (Implementation & Delivery)	Week 6	Deploy and finalize system	• Migrate data and deploy live version • Conduct user training • Collect post-deployment feedback and finalize documentation

Example Agile Project Schedule

EACOMM CORPORATION

Sprint	Duration	Main Goals	Key Tasks / Deliverables
Sprint 1 – Planning & Setup	Week 1–2	Establish foundations	• Project kickoff meeting • Define user stories • Set up development environment (Git, server, DB) • Draft initial backlog
Sprint 2 – Database & API Development	Week 3–4	Build backend core	• Design database schema • Implement CRUD operations • Create REST API endpoints • Unit testing
Sprint 3 – Frontend Integration	Week 5–6	Connect frontend to backend	• Integrate API with frontend • Add authentication • Test data flow (frontend ↔ backend ↔ DB)
Sprint 4 – Refinement & Testing	Week 7–8	Improve quality and stability	• Fix bugs • Improve UI/UX • Conduct user acceptance testing (UAT) • Prepare documentation
Sprint 5 – Deployment & Review	Week 9–10	Final delivery and evaluation	• Deploy system to server • Conduct final review and feedback • Sprint retrospective and reporting

Laws, Policies, and Regulations Governing Software Development

EACOMM CORPORATION

Philippines	Description / Coverage
Data Privacy Act of 2012 (RA 10173)	Protects personal data and ensures confidentiality, integrity, and consent in data processing.
Cybercrime Prevention Act of 2012 (RA 10175)	Penalizes hacking, illegal access, and online offenses. Promotes secure coding.
Intellectual Property Code (RA 8293)	Protects software copyrights and patents. Developers must respect licensing and ownership.
E-Commerce Act of 2000 (RA 8792)	Recognizes electronic documents and digital signatures as legally valid.
National ICT Policy (DICT)	Promotes ICT innovation and cybersecurity compliance among developers.

International	Description / Coverage
GDPR (EU General Data Protection Regulation)	Regulates data privacy and user consent for EU citizens. Affects systems handling EU data.
ISO/IEC 27001	International standard for Information Security Management Systems (ISMS). Ensures data protection and risk control.
WIPO (World Intellectual Property Organization)	Protects intellectual property rights globally for software and digital works.
COPPA (Children's Online Privacy Protection Act, U.S.)	Protects personal data of users under 13 years old — important for educational or child-oriented sites.
DMCA (Digital Millennium Copyright Act, U.S.)	Protects digital copyrights and provides mechanisms for removing infringing content online.

Code Versioning

EACOMM CORPORATION

Code versioning is the practice of **tracking and managing changes to source code** over time. It allows developers to **collaborate**, **revert mistakes**, and **maintain multiple versions** of a project efficiently.

Collaboration	Multiple developers can work on the same project without overwriting each other's work.
Change Tracking	Every edit, addition, or deletion is recorded with who made it and when.
Backup and Recovery	Easy rollback to a previous stable version when errors occur.
Branching and Merging	Developers can experiment on new features safely before merging into the main code.
Release Management	Enables tagging of specific versions for testing, staging, or production releases.

Code Versioning

EACOMM CORPORATION

Tool	Type	Key Features
Git	Distributed	Most popular VCS; works offline; supports branching, merging, and GitHub/GitLab integration.
Subversion (SVN)	Centralized	Stores all files on a central server; simpler for small teams.
Mercurial	Distributed	Similar to Git but with simpler commands and better Windows support.
Perforce (Helix Core)	Centralized	Used by large enterprises and game studios for big repositories.

Web Servers

Deployment Types

Type	Description	Example Providers
On-premise hosting	Host system/website on computer hardware within the office/data center	Dell, IBM, HP, etc.
Shared Hosting	Multiple sites on one server (low cost, limited control).	GoDaddy, Hostinger
VPS (Virtual Private Server)	Shared hardware, isolated virtual environments.	DigitalOcean, Linode
Dedicated Server	Full control over hardware and OS.	IBM, Rackspace, Equinix
Cloud Hosting	Scalable virtual servers on-demand.	AWS, Google Cloud, Azure

Web Servers

Server Specifications

Key specifications to consider when creating a server.

Specification	Description
CPU (Processor)	Determines performance for handling multiple requests.
RAM (Memory)	Affects speed and ability to run concurrent processes.
Storage	SSDs for speed, HDDs for capacity.
Bandwidth	Impacts data transfer and site performance.
GPU	Determine if a GPU is needed to run AI or data analytics or other math-heavy applications
Uptime Guarantee	Indicates server reliability (e.g., 99.9%).
Geographic Location	Location of server affects the latency (speed) of access. Legal and compliance issues may also dictate server location.

Web Servers

Software needed

Category	Examples	Purpose / Description
Operating System (OS)	Linux (Ubuntu, CentOS, Debian), Windows Server, macOS Server	Foundation of the web server; manages hardware, processes, users, and networking.
Web Server Software	Apache HTTP Server, Nginx, Microsoft IIS	Delivers website content to browsers via HTTP/HTTPS. Handles client requests and responses.
Database Management System (DBMS)	MySQL, MariaDB, PostgreSQL, MongoDB	Stores, retrieves, and manages dynamic data used by websites and applications.
Server-Side Scripting / Runtime	PHP, Node.js, Python, .NET Core	Executes backend logic and connects web pages to databases and APIs.
Security & Access Tools	OpenSSL, Let's Encrypt, Firewall (UFW), Fail2Ban	Provides encryption, firewall protection, and access control to secure the server.
Version Control & Deployment	Git, GitHub, GitLab, Docker, Jenkins	Tracks code changes, enables collaboration, and automates deployment (CI/CD).
Monitoring & Maintenance Tools	htop, Nagios, Zabbix, Grafana, Logwatch	Monitors system performance, uptime, and logs for troubleshooting and optimization.

Development Stacks

EACOMM CORPORATION

Stack Name	Components	Description / Use Case
LAMP	**L**inux (OS), **A**pache (Web Server), **M**ySQL (Database), **P**HP (Language)	Classic open-source stack for dynamic websites. Common in PHP apps like WordPress, Joomla, and Drupal.
LEMP	**L**inux, **E**ngine-X (Nginx), **M**ySQL / **M**ariaDB, **P**HP / **Python**	Similar to LAMP but uses **Nginx** for better performance and scalability. Ideal for high-traffic web apps.
MEAN	**M**ongoDB, **E**xpress.js, **A**ngular, **N**ode.js	Full JavaScript stack — from database to frontend. Great for single-page applications (SPAs). MongoDB can be replaced with MySQL/other RDBMS
MERN	**M**ongoDB, **E**xpress.js, **R**eact, **N**ode.js	Uses **React** instead of Angular. Popular for modern web apps and APIs.
Django Stack	**Python**, **Django**, **PostgreSQL / MySQL**, **Nginx / Apache**	Python-based backend framework. Excellent for rapid development, security, and data-driven web apps.
.NET Stack	**Windows Server**, **IIS**, **SQL Server**, **ASP.NET / C#**	Enterprise-grade stack used for business applications and intranet systems.
Serverless Stack	**AWS Lambda**, **Firebase**, **Vercel**, **Supabase**	Backend logic runs on demand — no need to manage servers. Ideal for modern scalable apps.

Asynchronous Exercise

CASE STUDY: “CampusConnect – Student Services Portal”

Background Scenario

The **City Technical Institute (CTI)** is a mid-sized educational institution that wants to streamline its student services by creating a **“CampusConnect” web-based portal**. This system will allow students to:

- View their grades and schedules,
- Request documents online (transcripts, certifications),
- Communicate with their department, and
- Receive campus announcements.

Currently, all these services are done manually through paper forms and physical visits to the registrar’s office.

The **school administration** has asked your team to propose a **project plan** that includes:

- A defined **scope of work**, taking into account the features needed and **legal and ethical compliance** (data privacy, copyright, security etc)
- A **software development methodology** – including justification on your selected approach.
- A **project timeframe and task breakdown –** taking into account SDLC and methodology selected

EACOMM CORPORATION

TESDA Web Development (Backend) NC III

Review Session-2

Setting Up Your Development Environment

EACOMM CORPORATION

Selecting Your Visual Editor

- VS Code
- Sublime
- Notepad++
- Etc.

Selecting Your Development Stack

- What OS are you running?
- What Database will you use? (MySQL, MariaDB, PostGre, etc.)
- What Scripting Language will you use? (NodeJS, PHP, Python, C#.net, etc.)
- What Web Server will you use? (Nginx, Apache, IIS, etc)

Install your preferred Development Stack

EACOMM CORPORATION

SOME LINKS FOR DOWNLOADING:

- VSCode: https://code.visualstudio.com/
- XAMPP (Apache + MariaDB + PHP + Perl): https://www.apachefriends.org/
- NodeJS: https://nodejs.org/en/download
- Github: https://desktop.github.com/download/
- Postman: https://www.postman.com/downloads/

Programming Exercise

- Create a script that when executed will display:
  - IP Address
  - User Agent
  - Operating System
  - Server Time

```
$ip = $_SERVER['REMOTE_ADDR'];
$userAgent = $_SERVER['HTTP_USER_AGENT'];
```

- Depending on the Time of Day customize a Greeting to be displayed:
  - E.g. if 11AM below: “Good Morning, have you eaten breakfast yet?”, if 11AM-2PM: “It’s lunchtime, let’s take a break!”, etc…
- Example output:

```
Current Time: 6:45 PM
Have you eaten dinner?
Your IP Address: 192.168.1.25
You’re visiting using: Mozilla/5.0
Your Operating System is: Windows NT 10.0; Win64; x64
```

Programming Exercise 2

EACOMM CORPORATION

Exercise: Recursive File Lister

Objective: Write a script that lists all files in a given folder **recursively**, including files in all subfolders.

Requirements

- Input: A folder path (either from the command line or hardcoded for simplicity).
- Output:
  - Print all files with their **relative paths**.
  - Optionally show **folder indentation** to visualize structure.
- Must work even if subdirectories contain more folders.

```
FUNCTION listFiles(directory, prefix = "")
   files ← getListOfFilesIn(directory) //gets list of files as an array

   FOR EACH file IN files DO //scrolls through array list
      IF file IS "." OR file IS ".." THEN //exclude . And .. In display
         CONTINUE TO NEXT file
      END IF

      path ← directory + "/" + file

      IF path IS a directory THEN //if folder, call function again
         PRINT prefix + file
         CALL listFiles(path, prefix + "   ")   // recursive call with extra
indent
      ELSE
         PRINT prefix + file //display file
      END IF
   END FOR
END FUNCTION
```

Bonus Exercise

EACOMM CORPORATION

- Create a PHP, Node.js, or Python script that uses **FFmpeg** to:
  - Extract a thumbnail (screenshot) from a given video file.
  - Resize the thumbnail to a fixed width (e.g., 320px).
  - Overlay a watermark image (e.g., watermark.png) onto the thumbnail.
  - Save the output to a new file (e.g., output/thumbnail_watermarked.jpg).

FFMPEG Command Example

Take a 320px screenshot at 2 second mark:

```
ffmpeg -ss 00:00:02 -i input.mp4 -vframes 1 -vf "scale=320:-1" output_thumbnail_resized.jpg
```

Overlay a watermark image:

```
ffmpeg -ss 00:00:02 -i input.mp4 -i watermark.png -vframes 1 -filter_complex "[0:v]scale=320:-1[vid];[vid][1:v]overlay=10:10" output_thumbnail_watermarked.jpg
```

https://www.ffmpeg.org/

EACOMM CORPORATION

TESDA Web Development (Backend) NC III

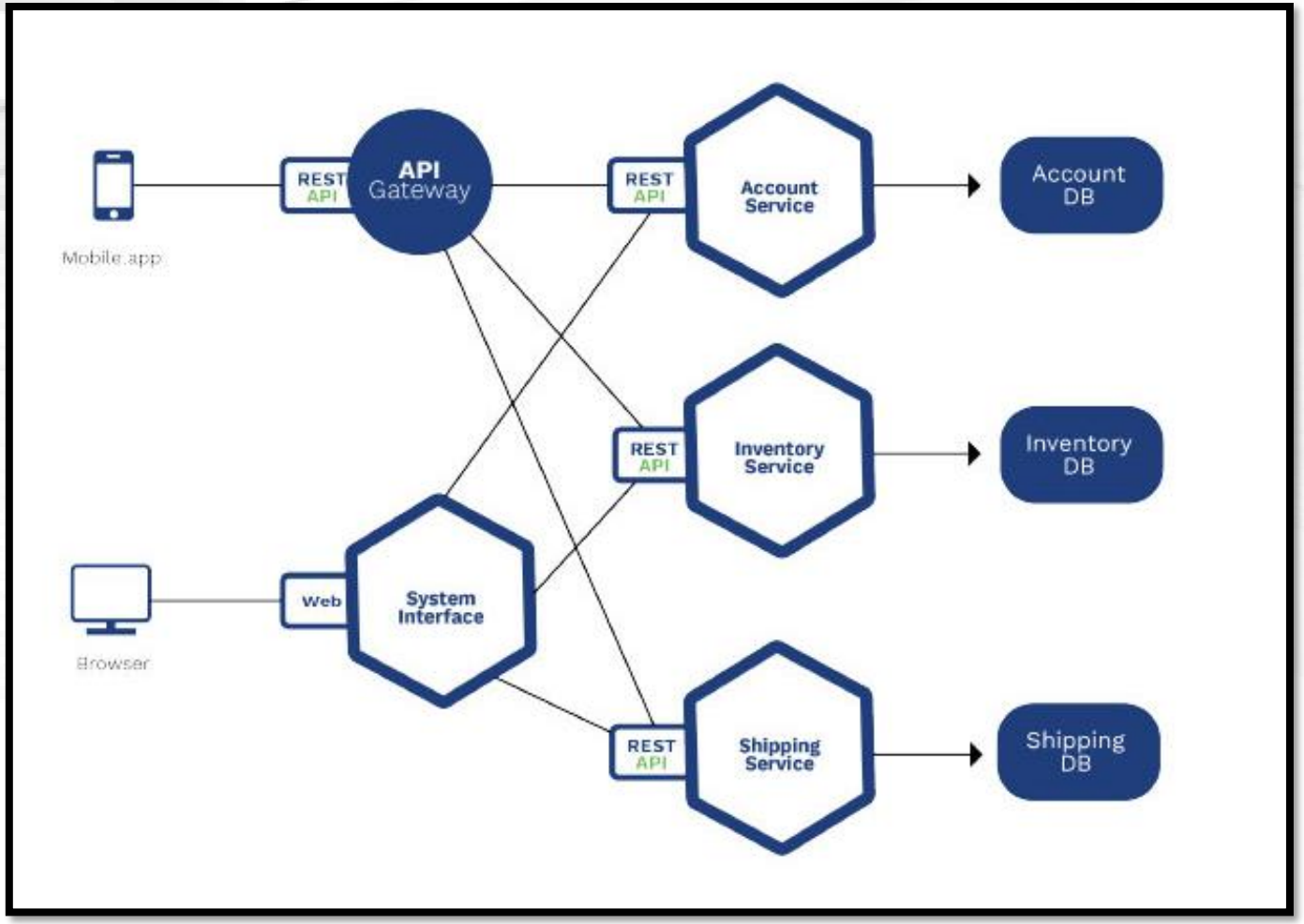
Review Session-3

Developing Website Backend Systems

EACOMM CORPORATION

Microservice Architecture
Software design approach where an application is built as a **collection of small, independent services**, each focusing on a **specific function**.

- **Independent Components**
- **API Communication**
- **Technology Flexibility**
- **Scalability & Resilience**
- **Continuous Deployment**



https://eacomm.com/blog/microservice-architecture/

Web Server User Access

EACOMM CORPORATION

Access Type	Description	Typical Tools / Methods
Public Access	Open to everyone on the internet. Usually for viewing website content (front-end).	Web browsers (HTTP/HTTPS)
Authenticated User Access	Requires login credentials. Users can access personalized or restricted content.	Login pages, session tokens, cookies
Admin / Superuser Access	Full control over the website or server. Used for configuration, deployment, backups, and security management.	Control Panel, SSH, FTP, Web Admin GUI
Developer Access	Used by developers to update, test, and debug code. Limited to specific environments (e.g., dev/staging).	SSH, Git, IDE integrations
Database Access	For managing or querying data stored on the server.	phpMyAdmin, MySQL CLI, pgAdmin, SQL Workbench
API Access	Access via application programming interface, usually restricted by API keys or tokens.	Postman, Curl, Mobile/Web Apps

Web Server Security

EACOMM CORPORATION

1. Keep Software Updated

- Regularly update the operating system, web server (Apache, Nginx, IIS), database, and frameworks.
- Apply security patches as soon as they are available.

2. Secure Configuration

- Disable directory listing and remove unused modules or services.
- Restrict file and folder permissions (e.g., 644 for files, 755 for folders).
- Protect configuration files (e.g., .env, config.php) from public access.

3. Use HTTPS (TLS/SSL)

- Install a valid SSL certificate and redirect all HTTP requests to HTTPS.
- Avoid mixed content by ensuring all resources load securely.

4. Firewalls and Network Protection

- Set up a **Web Application Firewall (WAF)** to filter malicious traffic.
- Use a **Network Firewall** to block unused ports and unauthorized IP addresses.
- Fail2Ban, UFW, or Cloudflare WAF can help prevent brute-force and DDoS attacks.

Web Server Security

EACOMM CORPORATION

5. Authentication and Access Control

- Enforce strong passwords and enable multi-factor authentication.
- Limit access to administrative panels by IP address or VPN.

6. Input Validation and Sanitization

- Validate and sanitize all user input to prevent SQL injection, XSS, and CSRF attacks.
- Use prepared statements and parameterized queries.

7. Monitoring and Logging

- Enable detailed access and error logs for the web server and applications.
- Regularly review logs and set up alerts for suspicious activity.

8. Backup and Recovery

- Schedule automatic backups of files and databases.
- Store backups securely and test restoration procedures periodically.

Server Uptime, SLA, and Management

EACOMM CORPORATION

1. Uptime Requirements

- **Uptime** measures how long a server stays operational without interruption.
- Common targets:
  - **99% uptime** → ~3.65 days downtime/year
  - **99.9% uptime** → ~8.76 hours/year
  - **99.99% uptime** → ~52 minutes/year
- The higher the uptime requirement, the greater the need for redundancy and monitoring.

2. Service Level Agreement (SLA)

- Defines **performance commitments** between service provider and client.
- Typically includes:
  - **Guaranteed uptime** percentage
  - **Response and resolution times** for issues
  - **Penalties or credits** for missed targets
- Ensures accountability and measurable service quality.

3. Managing and Maintaining Uptime

- **Monitoring:** Use tools like Nagios, UptimeRobot, or AWS CloudWatch to detect outages early.
- **Redundancy:** Deploy load balancers, failover servers, and backup power systems, RAID storage, etc.
- **Regular Maintenance:** Schedule updates and patches during low-traffic periods.
- **Scaling:** Use cloud auto-scaling or container orchestration (e.g., Kubernetes) to handle traffic spikes.
- **Incident Response:** Maintain clear escalation procedures and communication channels.
- **Horizontal/Vertical Scaling:** Consider adding more servers/instances (Horizontal) or increasing server specifications (Vertical) if performance is declining.
- **Containerization:** Isolate processes/applications using containers such as Docker to optimize server resource utilization.

Database Design

EACOMM CORPORATION

Purpose of Databases

- Organize data efficiently to support application functionality, integrity, and scalability.
- Ensure that data can be stored, retrieved, and updated accurately and efficiently.

Key Steps in Database Design

1. **Requirements Analysis** - Identify what data needs to be stored and how it will be used.
2. **Conceptual Design -** Create an **Entity-Relationship Diagram (ERD)** to define entities and relationships.
3. **Logical Design -** Define tables, columns, data types, and constraints based on the ERD.
4. **Normalization -** Eliminate redundancy and ensure data integrity (1NF → 3NF or higher).
5. **Physical Design -** Optimize indexes, storage, and partitioning for performance and scalability.

Best Practices

- Use **primary and foreign keys** to enforce relationships.
- Apply **indexes** judiciously to improve query performance.
- Implement **constraints** (e.g., NOT NULL, UNIQUE, CHECK) for data validation.
- Document the schema and relationships for future maintenance.

Database Requirements Analysis

EACOMM CORPORATION

1. Understand System Objectives

- Determine what the database should accomplish (e.g., track students, manage sales, store inventory).
- Align with organizational or project goals.

2. Identify Stakeholders and Data Users

- Consult end-users, managers, and developers.
- Understand how each will input, update, and retrieve data.

3. Gather Data Requirements

- List all **entities** (e.g., Customer, Order, Product).
- Identify **attributes** for each entity (e.g., Name, Date, Price).
- Note **relationships** among entities (e.g., one-to-many, many-to-many).

4. Determine Data Usage

- Define what **queries, reports, or transactions** must be supported.
- Estimate **data volume** and **growth rate** for performance planning.

5. Define Constraints and Business Rules

- Identify validation rules (e.g., unique email, required age > 18).
- Note security, access, and backup requirements.

Database Entity-Relationship Diagram

EACOMM CORPORATION

Key Components

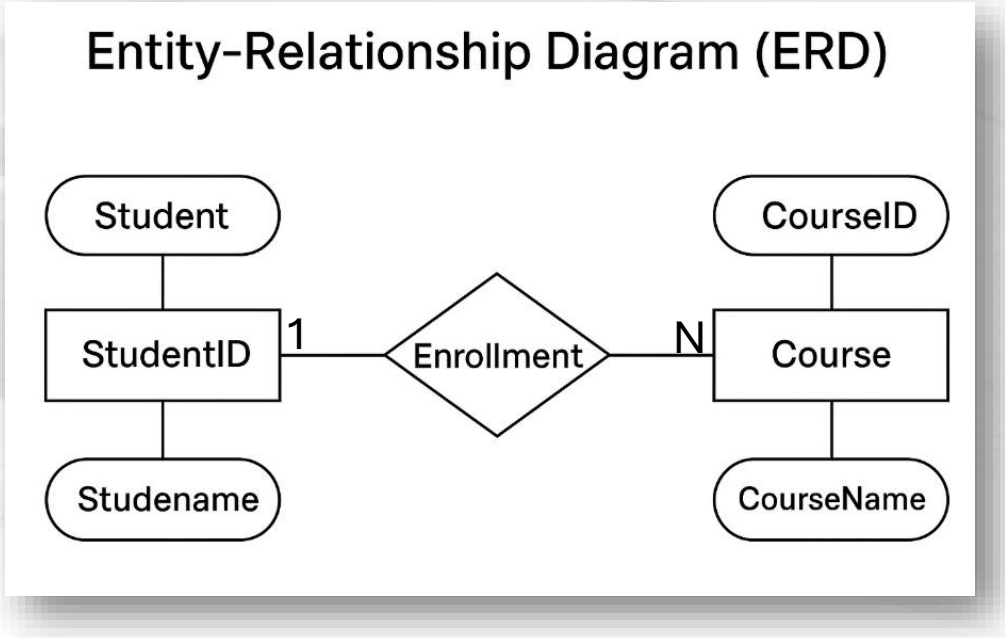
- **Entities** – Real-world objects or concepts stored in the database
  (e.g., Student, Course, Employee)
- **Attributes** – Properties or details about an entity
  (e.g., StudentName, CourseCode, HireDate)
- **Relationships** – Associations between entities
  *(e.g., A Student **enrolls in** a Course)*

Types of Relationships

- **One-to-One (1:1)** – e.g., Each employee has one ID card
- **One-to-Many (1:N)** – e.g., One teacher teaches many students
- **Many-to-Many (M:N)** – e.g., Students enrol in many courses

Common ERD Symbols

- **Rectangle** → Entity
- **Oval** → Attribute
- **Diamond** → Relationship
- **Lines** → Connect entities and relationships



Database Logical Design

Transform the **conceptual ERD** into a **logical schema** that defines tables, fields, and relationships.

Key Steps:

- **Map Entities to Tables -** Each entity becomes a database table.
- **Map Attributes to Columns -** Define data types, constraints, and defaults.
- **Define Primary and Foreign Keys -** Establish relationships and enforce referential integrity.

Column Name	Data Type	Constraints / Notes
customer_id	INT	**PRIMARY KEY**, AUTO_INCREMENT
full_name	VARCHAR (100)	NOT NULL
email	VARCHAR (100)	UNIQUE, NOT NULL
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

Column Name	Data Type	Constraints / Notes
order_id	INT	**PRIMARY KEY**, AUTO_INCREMENT
customer_id	INT	**FOREIGN KEY** → Customers(customer_id)
order_date	DATE	DEFAULT CURRENT_DATE
status	ENUM('Pending','Shipped','Delivered', 'Cancelled')	DEFAULT 'Pending'

Column Name	Data Type	Constraints / Notes
item_id	INT	**PRIMARY KEY**, AUTO_INCREMENT
order_id	INT	**FOREIGN KEY** → Orders(order_id)
product_name	VARCHAR (100)	NOT NULL
quantity	INT	DEFAULT 1 CHECK (quantity > 0)
price	DECIMAL(10,2)	NOT NULL CHECK (price >= 0)

Database Normalization

EACOMM CORPORATION

Initial Table

StudentID	StudentName	Courses
1	Ana Cruz	Math, English
2	Ben Santos	Science, English

First Normal Form (1NF)

- Each column holds atomic (indivisible) values.
- No repeating groups or arrays.

StudentID	StudentName	Course
1	Ana Cruz	Math
1	Ana Cruz	English
2	Ben Santos	Science
2	Ben Santos	English

Second Normal Form (2NF)

- Must first satisfy 1NF.
- Remove partial dependencies — all non-key columns depend on the entire primary key.

Student Table

StudentID	StudentName
1	Ana Cruz
2	Ben Santos

Enrollments Table

StudentID	Course
1	Math
1	English
2	Science
2	English

Database Normalization

EACOMM CORPORATION

Initial Table

StudentID	StudentName	Courses
1	Ana Cruz	Math, English
2	Ben Santos	Science, English

1NF

StudentID	StudentName	Course
1	Ana Cruz	Math
1	Ana Cruz	English
2	Ben Santos	Science
2	Ben Santos	English

2NF

Student Table

StudentID	StudentName
1	Ana Cruz
2	Ben Santos

Enrollments Table

StudentID	Course
1	Math
1	English
2	Science
2	English

Third Normal Form (3NF)

- Must satisfy 2NF.
- Remove transitive dependencies — non-key columns depend only on the primary key.

Students Table

StudentID	StudentName
1	Ana Cruz
2	Ben Santos

Courses Table

CourseID	CourseName
1	Math
2	English
3	Science

Enrollments Table

StudentID	CourseID
1	1
1	2
2	3
2	2

Database Physical Design

Translate the **logical design** into a **physical implementation** — defining how data will be **stored, indexed, and accessed** in the database.

- **Define Storage Structures**
  - Choose **data types** suited for the DBMS (e.g., VARCHAR(100) for names, DECIMAL(10,2) for prices).
  - Store large files separately using **BLOBs** or cloud storage (e.g., Amazon S3).
  - Example: Customer records stored in tbl_customers table under the USERDATA tablespace.
- **Indexing and Performance**
  - Create **indexes** to improve search speed on frequently queried fields.
  - Example: CREATE INDEX idx_lastname ON employees(last_name);
  - Use **clustered indexes** for primary keys and **non-clustered** for secondary search fields.
- **Partitioning and Archiving**
  - Split large tables for performance and maintenance.
  - Example: Sales data partitioned by **month or year** (sales_2024, sales_2025).
  - Archive old data to a separate table (archive_sales) or storage.
- **Security and Access Control**
  - Assign roles and privileges:
    - Example: GRANT SELECT ON employees TO hr_staff;
  - Encrypt confidential data like passwords or salary using AES encryption.
  - Use **role-based access control (RBAC)** in systems like PostgreSQL or SQL Server.
- **Backup and Recovery Planning**
  - Schedule **nightly backups** (e.g., using mysqldump or SQL Server Maintenance Plans).
  - Example: Daily incremental and weekly full backup strategy.
  - Use **replication** (e.g., MySQL Master-Slave setup) for disaster recovery.

EACOMM CORPORATION

TESDA Web Development (Backend) NC III

Review Session-4

Setup the Tables (using SQL, PHPMyAdmin, or other tools)

EACOMM CORPORATION

tbdepartments

Column Name	Type	Description
department_id	INT (PK)	Unique department ID
name	VARCHAR(50)	Department name
location	VARCHAR(50)	Office location

tbemployees

Column Name	Type	Description
employee_id	INT (PK)	Unique employee ID
name	VARCHAR(50)	Employee full name
email	VARCHAR(50)	Employee email address
department_id	INT (FK)	References departments table
salary	DECIMAL(10,2)	Employee salary

tbprojects

Column Name	Type	Description
project_id	INT (PK)	Unique project ID
project_name	VARCHAR(50)	Name of the project
department_id	INT (FK)	References departments table
budget	DECIMAL(10, 2)	Project budget

```
CREATE DATABASE company_db;
USE company_db;

CREATE TABLE departments (
 department_id INT PRIMARY KEY,
 name VARCHAR(50),
 location VARCHAR(50)
);
```

Populate the Tables

EACOMM CORPORATION

OPTIONS:

Use LOAD DATA using MYSQL to upload the contents of dbcontent.zip

```
LOAD DATA INFILE '/path/to/employees.csv'
INTO TABLE employees
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(name, email, department_id, salary);
```

Upload CSVs via PHPMyAdmin or MySQL Workbench

Database Query Exercises

EACOMM CORPORATION

Basic SELECT Queries

- Display all records from the employees table.
- Show only employee names and emails.
- List employees who belong to the “IT” department. *(Hint: Use JOIN)*
- Display all departments located in “Makati”.
- Show all projects with budgets greater than 70,000.

UPDATE Operations

- Increase the salary of all IT employees by 10%.
- Update the location of the “Marketing” department to “Pasig”.
- Change the project budget of “Website Redesign” to 200,000.

INSERT Operations

- Add a new department: department_id: 4, name: Finance, location: BGC
- Add a new employee assigned to Finance: employee_id: 105, name: Carlo Tan, email: carlo@company.com, department_id: 4, salary: 40000.00
- Add a new project for Finance with a budget of 90,000.

DELETE Operations

- Delete the project “Recruitment Drive”.
- Remove an employee with employee_id = 103.
- Delete the “Finance” department **only if it has no employees**. *(Hint: Use a subquery or check employee count first.)*

Server-side Scripting Exercise

EACOMM CORPORATION

- Create a Back-End that accepts the “Department” as a GET variable and it will return the employees of that department in CSV format with header information.
- For example: http://localhost/?department=Marketing returns:

employee_id	name	email	department_id	salary
103	Ana Reyes	ana@company.com	3	42000

- Make sure to use **secure** practices that avoids SQL injection and other potential threats.

EACOMM CORPORATION

TESDA Web Development (Backend) NC III

Review Session-5

Backend Data Processing

EACOMM CORPORATION

Aspect	MySQL (Database-Level Calculations)	Server-Side Scripts (PHP, Python, Node.js, etc.)
Performance	Faster for large datasets (uses indexes, runs close to data)	Slower if data must be fetched first before computing
Data Transfer	Reduces data sent over network (processes inside DB)	Requires pulling raw data, increasing transfer time
Complex Logic	Limited to SQL functions and expressions	Ideal for multi-step, conditional, or business logic
Maintainability	Harder to debug and modify complex SQL	Easier to maintain, version, and test in code
Scalability	Efficient for bulk, set-based operations	May strain application servers if handling large data
Security	Data stays within DB (less exposure)	More handling points, possible data leakage if not secured
Reusability	SQL logic tied to DB engine (less portable)	Code reusable across systems or databases
Example Use Case	Summing total sales per branch	Computing tax, commissions, or dynamic pricing

Backend Data Processing

EACOMM CORPORATION

- Use MySQL when the computation is data-intensive, aggregate, or query-based.
- Use Server-Side Scripts when the logic is complex, dynamic, or user-specific.
- The best systems combine both — letting SQL handle raw aggregation, and scripts handle business rules and presentation.

MySQL-based Data Processing

EACOMM CORPORATION

Data Aggregation

- **Compute the average salary across all employees.**
  → Use the AVG() function and round the result using ROUND().
- **Find the highest and lowest salary in the company.**
  → Use both MAX() and MIN() functions.
- **Show the average salary per department, sorted by highest average.**
  → Combine JOIN, AVG(), GROUP BY, and ORDER BY.
- **List departments with total project budgets exceeding 70,000.**
  → Use SUM() with GROUP BY and a HAVING clause to filter totals.

Data Filtering & Transformation

- **Show employees earning above the company average salary.**
  → Use a **subquery** with AVG() inside a WHERE condition.
- **Add a column showing each employee's annual salary (monthly × 12).**
  → Use a calculated field with multiplication and an alias.
- **Combine employee names with their department names in one column.**
  → Use CONCAT() to create a formatted text like "Maria Santos - HR Department".
- **Display the total payroll cost for each office location.**
  → Use SUM() on salary, and group by location from the departments table.

Summary Reports

- **Generate a Department Summary Report.**
  → Show each department's total number of employees, average salary, and total project budget using COUNT(), AVG(), and SUM() in a single query.
- **Identify the top earner in each department.**
  → Use a **subquery** that finds the MAX(salary) per department and match it to employee records.
- **Compare total project budgets vs. total salaries per department.**
  → Use SUM() and arithmetic operations to calculate the "budget balance" (budget – total salary) for each department.

Server-Side Data Processing with Error-Handling

EACOMM CORPORATION

Company Performance Evaluator

The company's internal system needs a backend script to **process employee performance submissions** sent from another service.
Each request includes:

- Employee name
- Department
- Hours worked
- Number of tasks completed

The script must:

- Validate the incoming data.
- Compute performance metrics (efficiency score, rating).
- Return a JSON response.
- Handle missing or invalid inputs gracefully.

Part 1 — Input Handling and Validation

- **Retrieve input data** via GET:
  Required fields:
  - name (string)
  - department (string)
  - hours (numeric)
  - tasks (numeric)

  http://localhost/process_performance.php?name=Maria&department=IT&hours=40&tasks=50
- **Validate**:
  - All fields must be present.
  - **Employee/Department must exist in your database.**
  - Hours, tasks, and errors must be positive numbers.
  - If any field is invalid, output a JSON error message with a clear description (e.g., "Missing required field: hours").
- **Implement basic sanitization** using trim() and htmlspecialchars() to clean input values.

Server-Side Data Processing with Error-Handling

EACOMM CORPORATION

Part 2 — Processing the Data

- **Compute performance efficiency** as:
  efficiency = (tasks / hours) * 100
- **Generate a performance rating** based on efficiency:
  - 90–100: Excellent
  - 75–89: Good
  - 50–74: Average
  - <50: Needs Improvement
- **Use conditional statements** (if, elseif, else) to determine the rating.

- **Package the results** into an array:

```
$result = [
 "name" => $name,
 "department" => $department,
 "efficiency" => round($efficiency, 2),
 "rating" => $rating
];
```

- Convert it to JSON and output it.

```
{
 "status": "success",
 "data": {
  "name": "Maria",
  "department": "IT",
  "efficiency": 95.5,
  "rating": "Excellent"
 }
}
```

```
{
 "status": "error",
 "message":
"Invalid input:
hours must be
greater than 0"
}
```

EACOMM CORPORATION

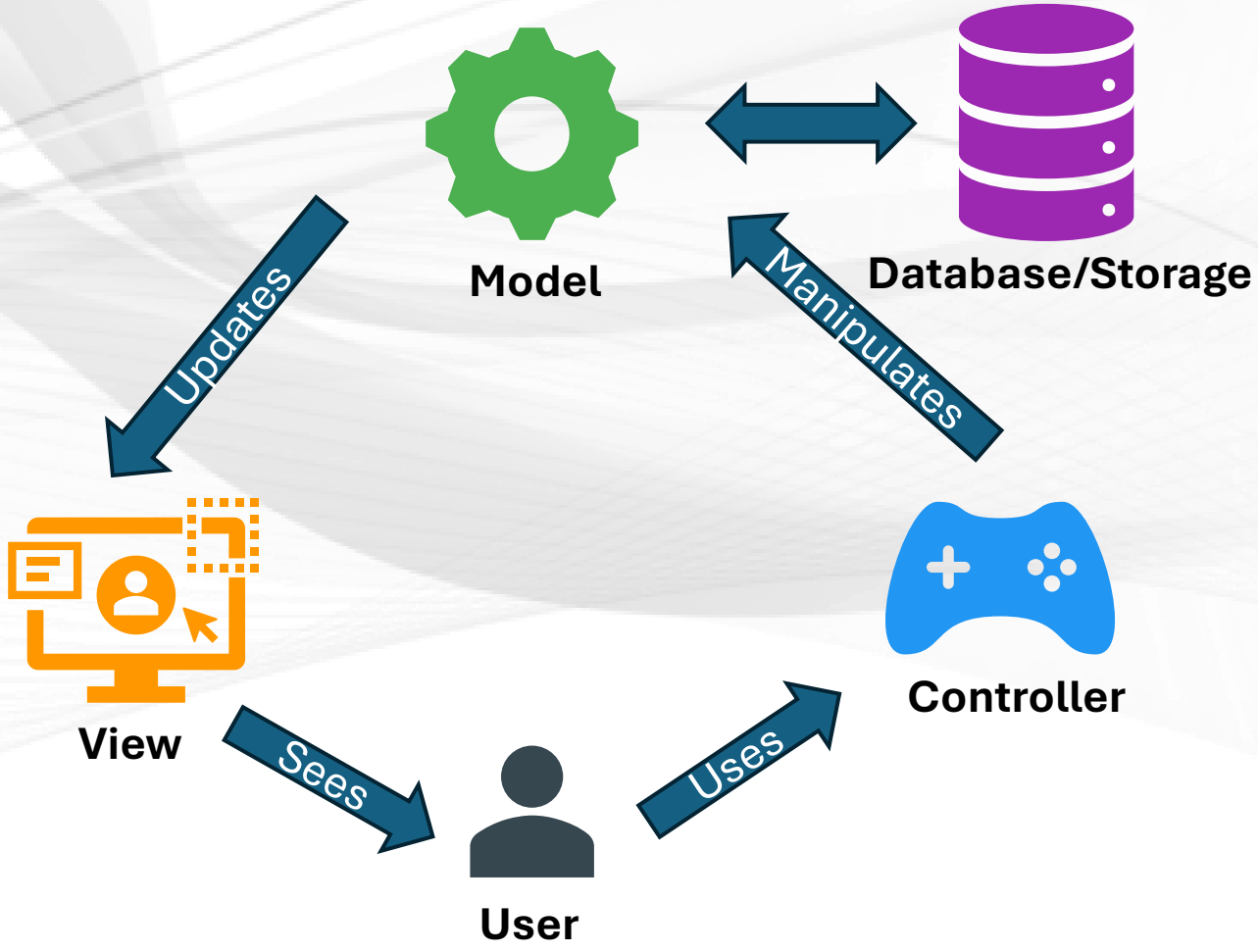
TESDA Web Development (Backend) NC III

Review Session-6

MVC Framework

MVC stands for **Model–View–Controller**. It's a design pattern used to organize code in web applications. It separates logic, UI, and user input for easier maintenance and scalability.

- **Model:** Handles data and business logic (e.g., database queries, data validation).
- **View:** Manages user interface and presentation (HTML, CSS, JS).
- **Controller:** Acts as the bridge that processes user input, updates the model, and refreshes the view.



https://eacomm.com/blog/microservice-architecture/

Examples of MVC Frameworks

EACOMM CORPORATION

Language	Popular MVC Frameworks	Notes / Highlights
	**Laravel, CodeIgniter, Symfony, CakePHP**	Easy setup, great for web apps and CMS development
	**ASP.NET MVC, ASP.NET Core**	Enterprise-ready, integrated with Microsoft ecosystem
	**Django, Flask (MVC-structured), Pyramid**	Django follows MTV (Model-Template-View), similar to MVC
	**Express.js (custom MVC), NestJS, AdonisJS**	Modern frameworks; NestJS enforces MVC structure
	**Spring MVC / Spring Boot, Struts, JSF**	Common in large-scale enterprise and academic systems
	**Ruby on Rails**	Pioneer of modern MVC pattern; convention over configuration

Application Programming Interfaces (API)

EACOMM CORPORATION

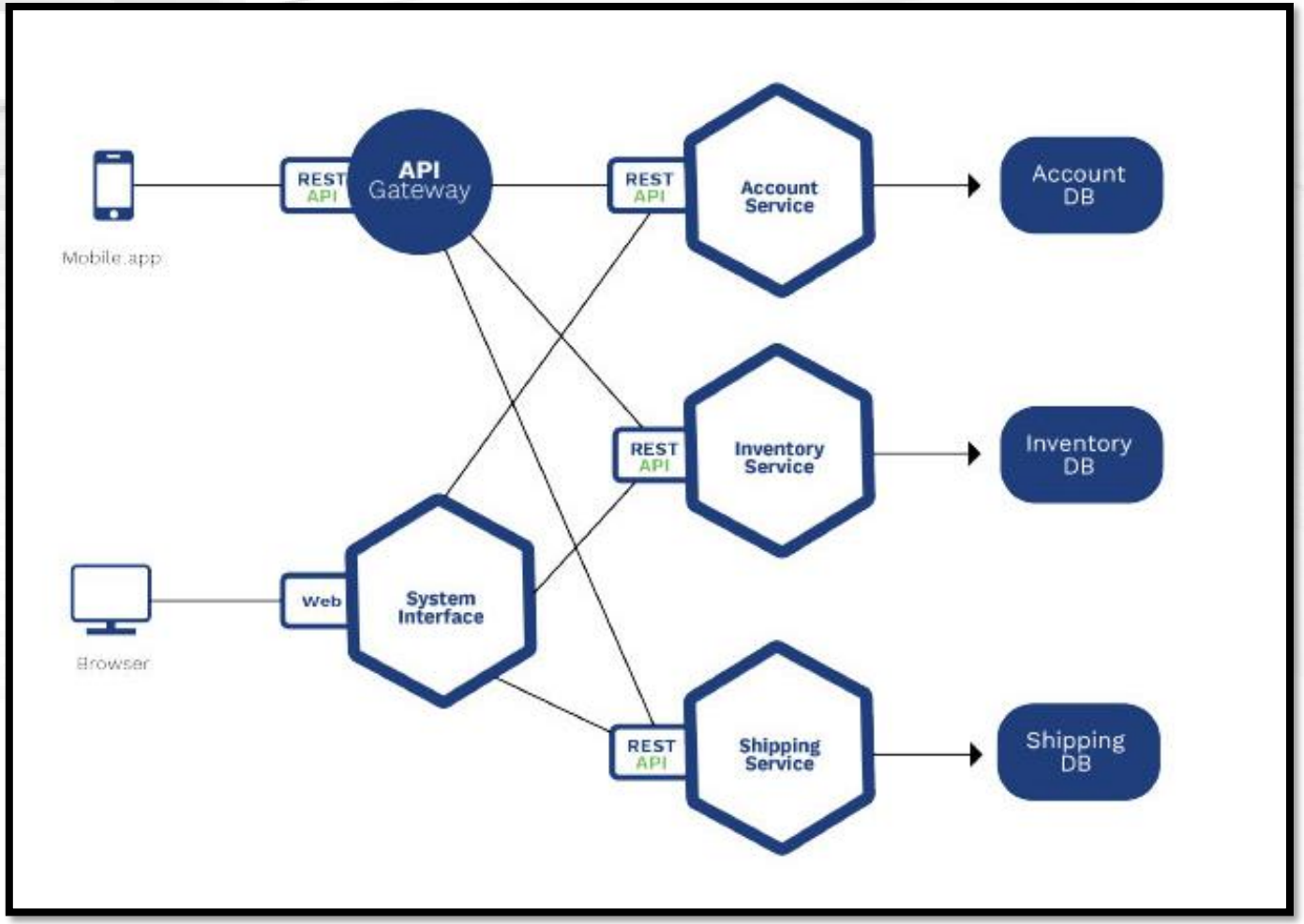
- **Definition:**
  An **API (Application Programming Interface)** is a set of rules that allows one software program to interact with another.
- **Analogy:**
  Think of an API as a **restaurant waiter** — it takes your order (request) to the kitchen (server) and brings back your food (response).
- **Why APIs Matter**
  - Allow **different systems** to talk to each other.
  - Enable **automation** and **integration**.
  - Power modern apps (social media, maps, payments, weather, etc.).
  - Save time by **reusing** existing services instead of rebuilding them.

Examples of APIs

Service	API Example	What It Does
Google Maps	Maps API	Embed maps or get coordinates
OpenWeather	Weather API	Get real-time weather data
PayPal	Payments API	Process online payments
Facebook	Graph API	Access user posts, pages, etc.

Developing Website Backend Systems

- **MVC** organizes *each service's internal code structure.*
- **APIs** expose *specific functions or data* from that service.
- **Microservices** use APIs to *interact* with other services, creating a flexible system.



https://eacomm.com/blog/microservice-architecture/

RESTful APIs

EACOMM CORPORATION

- **REST** stands for **Representational State Transfer** — it's an **architectural style** for designing **web services** that are:
  - **Stateless**
  - **Uniform**
  - **Client-server based**
  - **Cacheable**
- A **RESTful API** is simply a web API that follows these REST principles and communicates using **HTTP**.

- Most common type of web API
- Uses HTTP methods:
  - GET → Retrieve data
  - POST → Create data
  - PUT/PATCH → Update data
  - DELETE → Remove data
- Uses URLs to identify resources
- Uses JSON for data exchange (Older APIs would use XML)

EXAMPLE: https://jsonplaceholder.typicode.com/posts/1

HTTP Methods and Status Codes

EACOMM CORPORATION

Method	Action	Example	Success Codes	Common Error Codes
GET	Retrieve data	/users → fetch user list	200 OK – Request successful	404 Not Found, 500 Server Error, 401 Unauthorized
POST	Create new data	/users → add new user	201 Created – Resource created	400 Bad Request, 409 Conflict, 500 Server Error
PUT	Update/replace entire data	/users/1 → replace user info	200 OK or 204 No Content – Update successful	400 Bad Request, 404 Not Found, 500 Server Error
PATCH	Update part of a record	/users/1 → update email only	200 OK or 204 No Content – Partial update successful	400 Bad Request, 404 Not Found, 500 Server Error
DELETE	Remove data	/users/1 → delete record	200 OK or 204 No Content – Deleted successfully	404 Not Found, 403 Forbidden, 410 Gone, 500 Server Error

2xx codes = Success
4xx codes = Client error (your request is wrong)
5xx codes = Server error (API failed internally)

API Authentication

EACOMM CORPORATION

Method	How It Works	Example Use Case	Pros / Cons
API Key	A unique string sent in headers or URL	Weather, Maps APIs	Simple / Must be kept secret
Basic Auth	Sends username + password (Base64 encoded)	Internal/test APIs	Easy setup / Not secure without HTTPS
Bearer Token / JWT	Client sends a signed token proving identity	Modern web/mobile apps	Secure and stateless / Tokens can expire
OAuth 2.0	Delegated access via third-party (e.g., "Login with Google")	Social media, finance APIs	Industry standard / Complex setup

Example Bearer Token Authentication:

GET /user/profile

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6...

Server Checks for Token Validity, Expiration, and Permissions before responding.

Securing Your API

EACOMM CORPORATION

Common Security Threats to APIs

Threat	Description	Example
API Key exposure	Keys stored in frontend or GitHub	Public repo leak
Injection attacks	Unsafe input in queries	SQL Injection
Man-in-the-middle (MITM)	Data intercepted in transit	No HTTPS
Rate abuse / DDoS	Too many requests	Bot overload
CORS* misconfigurations	Wrong cross-domain setup	Data leaks

Best Practices for Securing APIs

- ✓ Use **HTTPS** for all endpoints
- ✓ Store **API keys/tokens** securely (never in code)
- ✓ Use **rate limiting** to prevent abuse
- ✓ Implement **CORS policies** carefully
- ✓ Rotate and **expire tokens** regularly
- ✓ Validate and **sanitize user input**
- ✓ Monitor and **log API activity**

**CORS, or Cross-Origin Resource Sharing, is a browser security mechanism that allows a web page from one "origin" (domain, protocol, or port) to request resources from a different origin.*

Accessing Third Party APIs

EACOMM CORPORATION

EXERCISE:

- Get the Current Weather of Metro Manila Using: https://api.open-meteo.com/v1/forecast?latitude=14.5995&longitude=120.9842&current_weather=true
- Sign-up for a FREE account at https://groq.com
- Once logged in create a new API Key at: https://console.groq.com/keys
- Create a Simple Weather App using server-side scripts by getting the current weather from open-meteo and asking groq.com to generate a summary in paragraph format.

  Example: https://eacomm.com/support/manilaweather.php

Create a Manila Weather App

EACOMM CORPORATION

WMO Weather Codes

Code	Description
0	Clear sky
1, 2, 3	Mainly clear, partly cloudy, and overcast
45, 48	Fog and depositing rime fog
51, 53, 55	Drizzle: Light, moderate, and dense intensity
56, 57	Freezing Drizzle: Light and dense intensity
61, 63, 65	Rain: Slight, moderate and heavy intensity
66, 67	Freezing Rain: Light and heavy intensity
71, 73, 75	Snow fall: Slight, moderate, and heavy intensity
77	Snow grains
80, 81, 82	Rain showers: Slight, moderate, and violent
85, 86	Snow showers slight and heavy
95	Thunderstorm: Slight or moderate
96, 99	Thunderstorm with slight and heavy hail

Simplified Groq Access:

Use: https://eacomm.com/support/groq.php

POST Payload Accepted:

```
{
    "api_key": "YOUR_GROQ_API_KEY",
    "prompt": "Explain the theory of relativity simply"
}
```

EACOMM CORPORATION

TESDA Web Development (Backend) NC III

Review Session-7

Create a Central API Router

EACOMM CORPORATION

Create a file like api.php that accepts requests like:

- api.php?endpoint=employees&action=read
- api.php?endpoint=departments&action=create
- api.php?endpoint=project&name="web design"&action=update
- api.php?endpoint=employess&id=1&action=delete

- Retrieve query parameters using $_GET['endpoint'] and $_GET['action']. Actions accepted are: **C**reate, **R**ead, **U**pdate, or **D**elete or use GET (for Read), POST (for Create), PUT (for Update), DELETE (for Delete)
- Return a 401 JSON error if the token is missing or invalid.

API CRUD Operations

EACOMM CORPORATION

Create scripts to process various types of CRUD operations

- **R**ead (GET) – Return all rows of given table (department/employees/projects)
  - Check endpoint = table name.
  - Use SELECT * FROM table.
  - Return data as JSON array.
  - Handle invalid table name or SQL errors.
- **C**reate (POST) – Insert a new record to provided endpoint.
  - Accept POST data (e.g., for employees: name, email, department_id, salary).
  - Validate all required fields are present.
  - Execute an INSERT INTO query.
  - Provide success/error JSON response.
- **U**pdate an existing record identified by id.
  - Accept id and other fields via POST.
  - Use UPDATE table SET … WHERE id=?.
  - Return a message confirming update or error if record not found.
- **D**elete a record from the given table.
  - Accept id via POST.
  - Validate ID exists before deletion.
  - Return JSON with status: success or status: error.

Centralized Error Response by sending http response code 400.

```
function sendError($message, $code = 400) {
    http_response_code($code);
    echo json_encode(["status" => "error", "message" => $message]);
    exit;
}
```

Creating Secure APIs

EACOMM CORPORATION

Create a new table for authorized API users:

```
CREATE TABLE users (
  user_id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) UNIQUE,
  password VARCHAR(255)
);

INSERT INTO users (username, password)
VALUES ('admin', SHA2('admin123', 256));
```

Creating Secure APIs

EACOMM CORPORATION

Create an Authentication End-Point e.g. api_auth.php:

- Accept POST requests with username and password.
- Validate credentials against the users table.
- If valid, generate and return a simple auth token (e.g., base64_encode(username:timestamp)).
- Return JSON:
  - SUCCESS: { "status": "success", "token": "xyz123" }
  - INVALID: { "status": "error", "message": "Invalid username or password" }
- **Store the token temporarily** (for example, in a PHP session variable or a text file).
  - This will simulate session-based access for the next API endpoints.
- **Include error handling** for:
  - Missing POST fields.
  - Empty username/password.
  - Database connection failure.