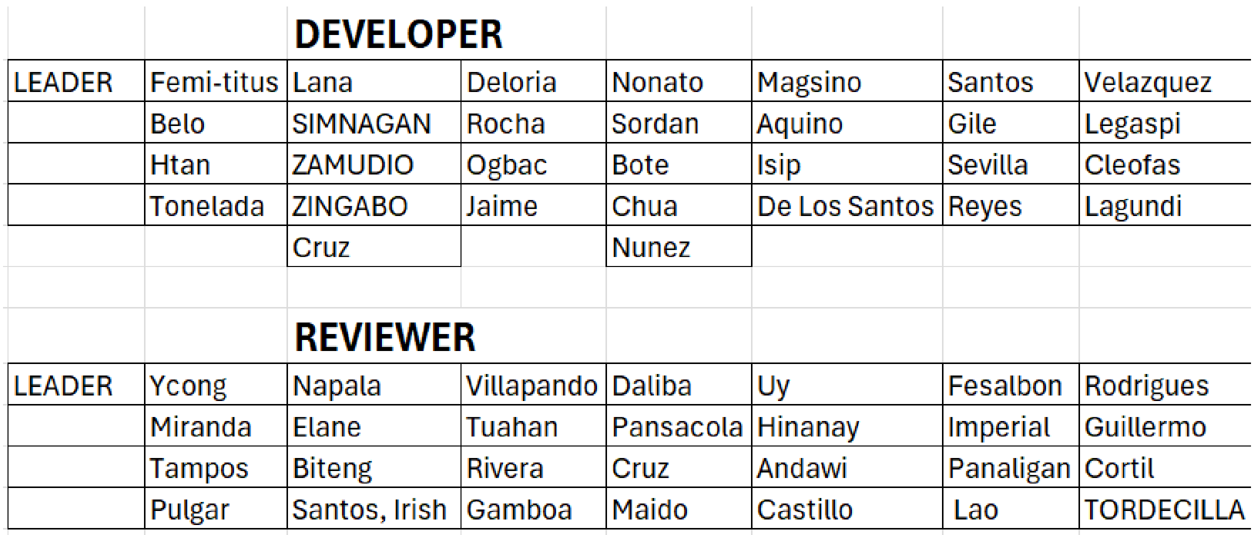
NEW GROUPINGS:



		DEVELOPER					
LEADER	Femi-titus	Lana	Deloria	Nonato	Magsino	Santos	Velazquez
	Belo	SIMNAGAN	Rocha	Sordan	Aquino	Gile	Legaspi
	Htan	ZAMUDIO	Ogbac	Bote	Isip	Sevilla	Cleofas
	Tonelada	ZINGABO	Jaime	Chua	De Los Santos	Reyes	Lagundi
		Cruz		Nunez			
		REVIEWER					
LEADER	Ycong	Napala	Villapando	Daliba	Uy	Fesalbon	Rodrigues
	Miranda	Elane	Tuahan	Pansacola	Hinanay	Imperial	Guillermo
	Tampos	Biteng	Rivera	Cruz	Andawi	Panaligan	Cortil
	Pulgar	Santos, Irish	Gamboa	Maido	Castillo	Lao	TORDECILLA

📌 Project Instructions: Secure File Sharing / Document Vault App

Project Title: Secure File Sharing / Document Vault App
Technology Stack: Laravel 12 (Backend), ReactJS (Frontend), Docker (Deployment), MySQL (Database)
Schedule:

- **Development period (async): Sept. 16 – Sept. 19, 2025**
- **Presentation dates: Sept. 24 and Sept. 25, 2025**
- **Mode: Team-based**

🎯 Project Objectives

This project aims to help you apply information assurance and security concepts in a real-world application by developing a Document Management System (DMS) with the following features:

1. **Confidentiality – Files are stored encrypted using AES, and keys are managed using RSA.**

2. **Access Control – Secure login and Role-Based Access Control (RBAC) (Admin, Staff, Regular User).**

3. **Integrity – Uploaded documents must pass validation (MIME check, size limits, virus scan optional).**

4. **Accountability – Every action is logged (upload, delete, share, download).**

5. **Availability – Data backup and integrity verification mechanisms must be demonstrated.**

🧑‍🤝‍🧑 Team Structure

Each team will have two key roles:

1. **App Developers – Responsible for implementing features and coding.**

2. **App Reviewers – Responsible for testing, auditing security measures, and validating features.**

All members are expected to understand the concepts, whether they are reviewers or developers.

📚 Required Concepts to Learn

Before coding, every student should study and understand the following security concepts:

- **Encryption: AES (file storage), RSA (key sharing).**
- **RBAC: Role-based access control (Admin, Staff, User).**
- **File Security: MIME type validation, file size limit, and optional virus scanning.**
- **Audit Logging: Capture actions like login, upload, delete, and share.**

- **Backup & Data Integrity: Methods to backup files and verify their integrity (checksums/hashes).**

🛠️ Project Requirements

1. Authentication & RBAC

- **Implement user authentication (login/logout).**
- **Define roles: Admin, Staff, Regular User.**
    - **Admin: Manage users, view logs, full access.**
    - **Staff: Upload, share, manage department files.**
    - **User: Upload and view their own files only.**

---

2. File Handling & Encryption

- **Upload and download files must be encrypted with AES.**
- **Use RSA key pair for securely sharing AES keys between users.**
- **Validate file uploads:**
    - **Only allow safe MIME types (PDF, DOCX, TXT, etc.).**
    - **Set maximum file size limit.**
    - **(Optional bonus) Add virus scanning.**

---

3. Audit Logging

- **Every user action must be recorded in the system:**

  - **Login/Logout**
  - **File Upload/Download/Delete**
  - **Sharing/Accessing files**
- **Logs should include timestamp, user, and action performed.**

---

4. Backup & Integrity

- **Demonstrate at least one backup method (e.g., scheduled database dump, file backup).**
- **Implement data integrity check using hashes (e.g., SHA256) to ensure files are not tampered with.**

📅 Timeline & Deliverables

- **Sept 16 – Sept 19 (Async): Project development & preparation. Equivalent to your async class time.**
- **Sept 24 – Sept 25: Team presentation.**

Each presentation must include:

1. **Live demo of the system.**
2. **Explanation of implemented security features (AES, RSA, RBAC, logging, backup).**
3. **Reviewer's validation report (how they tested, what issues they found, if requirements were met).**

✅ Evaluation Criteria

- **Security Features (40%) – Correct application of encryption, RBAC, validation, logging, backup.**
- **Functionality (30%) – The app works as intended (upload, share, search, restrict, etc.).**
- **Presentation & Documentation (20%) – Clarity of explanation, demo, and reviewer's feedback.**
- **Team Collaboration (10%) – Evidence of teamwork and shared understanding.**

🔑 **Reminder: This project is not just about coding but about understanding and applying security principles. Both developers and reviewers must be able to explain how the system protects confidentiality, integrity, and availability.**

🔒 Project Instructions: Secure File Sharing / Document Vault App

---

1. 🎯 Learning Goals

By completing this project, students will:

- Understand **encryption** (AES for file storage, RSA for key sharing).
- Apply **role-based access control (RBAC)** with user roles (Admin, Staff, Regular User).
- Implement **secure file handling** (MIME type check, size limit, virus scan optional).
- Build **audit logging** for accountability.
- Demonstrate **backup and data integrity** verification.
- Deploy in **Dockerized environment** for reproducibility.

---

2. 🏗️ High-Level Architecture

```
ReactJS (Frontend)
  ↓ (REST API / JWT)
Laravel 12 API (Backend)
  ↓
File Storage (encrypted files)
Database (users, roles, logs, metadata)
  ↓
Dockerized Environment (frontend, backend, db, nginx/reverse proxy)
```

---

3. 🛠️ Step-by-Step Development Instructions

A. Setup & Scaffolding

1. **Docker Environment**
   - Create containers for:
     - Laravel backend (PHP, Composer, Nginx/Apache)
     - React frontend (Node)
     - MySQL/PostgreSQL database
     - (Optional: Redis for sessions)
   - Define them in `docker-compose.yml`.
2. **Laravel Scaffold**
   - Install Laravel 12 inside Docker container.
   - Scaffold basic **authentication** (JWT or Sanctum).
   - Setup models:
     - `User` (with roles: admin, staff, user)
     - `File` (metadata, owner, path, hash)
     - `AuditLog` (user_id, action, timestamp, IP).
3. **React Scaffold**

    - Create a React app (frontend container).
    - Setup routes: Login, Dashboard, Upload, My Files, Shared Files, Admin Panel.
    - Implement JWT-based login/logout (connect to Laravel API).

B. Core Features

🔑 1. Authentication & RBAC

- Laravel: implement roles via middleware (e.g., `isAdmin`, `isStaff`).
- Admin: can manage users, view logs.
- Staff: can upload, share, manage departmental docs.
- User: can upload/view own files, access shared files.

📂 2. Secure File Upload & Download

- On **upload**:
    - Validate file size & MIME type.
    - Encrypt file contents (AES-256).
    - Store encryption key per file, encrypted with RSA (admin/staff keys).
    - Save file metadata in DB.
- On **download**:
    - Check user permissions.
    - Decrypt key with RSA, then file with AES.
    - Serve securely (force download, never inline for sensitive docs).

🔍 3. Powerful Search

- Store metadata + optional OCR/text extraction (basic text index).
- Allow search by: filename, uploader, labels, date, extracted text.

📝 4. Logging & Auditing

- Every upload, download, delete, share action → record in `AuditLog`.
- Include: `user_id`, `action`, `file_id`, `timestamp`, `IP`.
- Admin can view logs in a dashboard.

💾 5. Backup & Data Integrity

- Database backup script (cronjob inside Docker).
- File backup with versioning.
- Integrity: generate SHA-256 hash for each file at upload; verify on access.

C. Frontend Features (React)

- **Dashboard:** quick view of files uploaded, recent activity.
- **Upload Page:** drag & drop upload with progress bar.
- **My Files:** list + search uploaded files.
- **Shared Files:** view files shared with me.
- **Admin Panel:**
  - Manage users & roles.
  - View audit logs.

    - Backup/restore options.

D. Security Hardening

- Use HTTPS (self-signed cert in Docker for dev).
- Use CSRF tokens in React forms.
- Input validation (no raw SQL → Eloquent ORM only).
- Enforce strong password policies.
- Set storage outside webroot (so files aren’t directly accessible).
- Rate-limit login attempts.

---

4. 📦 Deliverables

1. **Source Code** (Laravel + React + Docker configs).
2. **Database Schema** (ERD + migrations).
3. **Demo Video** (walkthrough of features + security).
4. **Documentation** (README, setup steps, security features explained).

---

5. 📊 Grading Rubric

- **Core Functionality (40%)** → working upload, download, RBAC.
- **Security Features (30%)** → encryption, logging, validation.
- **Deployment (10%)** → Docker setup working.
- **Documentation (10%)** → clear instructions & code comments.
- **Presentation (10%)** → demo clarity.

📦 Starter Scaffold for Secure File Sharing / Document Vault

---

1. 🐳 Docker Setup (docker-compose.yml)

This defines 3 services: Laravel backend, React frontend, and MySQL DB.

```yaml
version: "3.8"
services:
  app:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: laravel_app
    restart: unless-stopped
    working_dir: /var/www
    volumes:
      - ./backend:/var/www
    depends_on:
      - db
    networks:
      - dms_net

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: react_app
    restart: unless-stopped
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/usr/src/app
    networks:
      - dms_net

  db:
    image: mysql:8
    container_name: dms_db
    restart: unless-stopped
    environment:
      MYSQL_DATABASE: dms
      MYSQL_ROOT_PASSWORD: root
      MYSQL_USER: dms_user
      MYSQL_PASSWORD: secret
    ports:
      - "3306:3306"
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - dms_net

networks:
  dms_net:

volumes:
  db_data:
```

---

2. 📂 Laravel Backend Scaffold (`/backend`)

Dockerfile (Laravel)

```
FROM php:8.3-fpm

# System deps
RUN apt-get update && apt-get install -y \
    build-essential \
    libpng-dev \
    libjpeg-dev \
    libfreetype6-dev \
    zip \
    git \
    unzip \
    curl \
    libonig-dev \
    libxml2-dev

# PHP extensions
RUN docker-php-ext-install pdo pdo_mysql mbstring exif pcntl bcmath gd

# Composer
COPY --from=composer:2.7 /usr/bin/composer /usr/bin/composer

WORKDIR /var/www

COPY . /var/www
```

```
RUN composer install --no-scripts --no-autoloader

RUN php artisan key:generate

CMD ["php-fpm"]
```

Initial Laravel setup:

- Install Laravel 12 in `/backend`.
- Add **JWT Authentication** (`tymon/jwt-auth`).
- Scaffold users & roles:

```
// database/migrations/xxxx_create_users_table.php
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->string('password');
    $table->enum('role', ['admin', 'staff', 'user'])->default('user');
    $table->timestamps();
});
```

Auth Controller (JWT Login/Register)

```
public function login(Request $request) {
    $credentials = $request->only('email', 'password');
    if (!$token = auth()->attempt($credentials)) {
        return response()->json(['error' => 'Unauthorized'], 401);
    }
    return response()->json(['token' => $token]);
}
```

---

3. ⚛️ React Frontend Scaffold (`/frontend`)

Dockerfile (React)

```
FROM node:20

WORKDIR /usr/src/app
```

```
COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000
CMD ["npm", "start"]
```

Basic React Setup

- Pages: Login.js, Dashboard.js, MyFiles.js, SharedFiles.js, AdminPanel.js.
- React Router configured.
- API client (axios) with JWT header.
- Simple login flow:

```
// services/api.js
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost/api", // Laravel backend
});

api.interceptors.request.use((config) => {
  const token = localStorage.getItem("token");
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

export default api;
```

Login Component (simplified)

```
import React, { useState } from "react";
import api from "../services/api";

function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const res = await api.post("/login", { email, password });
      localStorage.setItem("token", res.data.token);
      window.location.href = "/dashboard";
    } catch (err) {
      alert("Invalid credentials");
    }
  };

  return (
    <form onSubmit={handleLogin}>
      <input type="email" onChange={(e) => setEmail(e.target.value)} />
      <input type="password" onChange={(e) => setPassword(e.target.value)} />
      <button type="submit">Login</button>
    </form>
  );
}

export default Login;
```

---

4. 🚀 Next Steps for Students

- **Implement Encryption** (AES + RSA key wrapping in Laravel service).
- **File Upload/Download API** with secure storage (not in `/public`).
- **RBAC Middleware** (only admins can view logs, staff can share files, etc).
- **Audit Logging** (create `AuditLog` model + DB table).
- **Search Engine** (use Laravel Scout or LIKE queries on metadata).
- **Backup Scripts** (DB + file volume backup).