420-DW3-AS Group 07278 H24

Integration Project

Table of Contents

Introduction

Read carefully the entirety of this document. It lists very specific requirements, which the failure to respect can cost you dearly during grading. If you have any questions or if you are uncertain of any part of the assignment, do not hesitate to contact me for clarifications.

This integration project is designed to represent a part or a module of what would be a larger dynamic, server-side web application. Due to the scale and complexity of modern web applications, completing an entire functional web application of any scope would be too large an assignment; I have therefore opted to limit the task to a single functionality/module, but to require this functionality to be implemented up to professional-quality levels. In simple terms, you will not do a complete web application, only a part of it, but this part will be detailed up to professional standards.

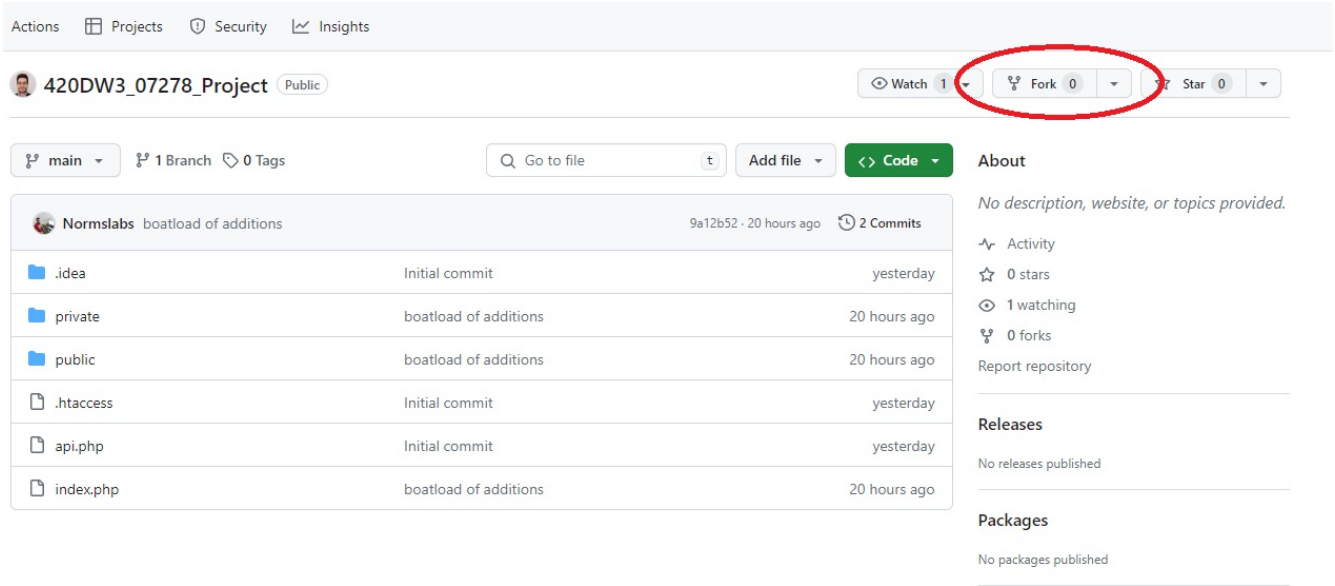
Regardless of this limitation, the scope of the functionality will encompass all the requirements of the course. This means that even though we will limit ourselves to what amounts to a single module of a larger web application, its requirements and the logic of its functionalities are equivalent to those of a complete web application; they are simply limited in scope.

Initial project

In order to simplify your work and to give you access to pre-built utility code (debug functions, class autoloader, internal routing system, ...), demonstrations and examples, an initial project over which you can build your own is available on Github at :

Initial project github repository

Please use the 'fork' button on the repository's main page to create your own copy of it on your Github account that you can then work with (see the image below).



Then you can clone your own fork on your local computer to work on your own version of the project.

I will update the original project with more examples and more documentation as we continue seeing the relevant course material in the coming weeks. You will be able to synchronize and obtain those updates in your own version.

Context

You are a full-stack developer at *{CodeBloc} Inc.* (fictitious company, ed. note) working on a new modular ERP (Enterprise Resource Planning) software project as a platform for e-commerce transactional web applications. You are tasked with developing the users, user groups and permissions subsystem, its management web pages and a login page to access such administrative functionalities.

Other than respecting the specifications and requirements listed below, you are left free to design your own UI and your own internal style of coding, but the company enforces basic coding standards for all its developers that you must follow.

Specifications

Users

Users represent individual user accounts that can log-in to the management functionalities of the application. Their data representation consists of the following attributes :

- an internal unique identifier,
- a unique username,
- a password,
- an e-mail address,
- temporal management date and time fields (creation, modification), and
- any other attribute that is appropriate.

The password of the users must not be stored in an un-encrypted manner; instead, a one-way hash function using the *Blowfish* algorithm must be used to generate hashes that can then be stored and used to compare password values.

Users can be part of zero, one, or many user groups.

Individual users can possess individual permissions.

User groups

User groups represent groups of users to which specific permissions are granted. When a user is part of a certain user group, it inherits all the permissions assigned to that group. Their data representation consists of the following attributes :

- an internal identifier,
- a group name,
- a group description,
- temporal management date and time fields (creation, modification), and
- any other attribute that is appropriate.

A user can be a part of multiple user groups.

Permissions

Permissions represent access rights to certain parts of the application; they can be assigned to either individual users or to user groups. They are used to restrict access to certain functionalities only to users who possess certain permissions or who are part of a user group that possess them. Their data representation consists of the following attributes :

- an internal identifier,
- a unique permission identifying string,
- a permission name,
- a permission description,
- temporal management date and time fields (creation, modification), and
- any other attribute that is appropriate.

Here are examples of permissions that are expected to exist for the current application project :

- LOGIN_ALLOWED (allows users to log-in)
- MANAGE_PERMISSIONS (allows access to management of permission entities, including CRUD operations)
- MANAGE_USERGROUPS (allows access to management of user group entities, including CRUD operations)
- MANAGE_USERS (allows access to management of user entities, including CRUD operations)

Suggested Calendar

Course calendar week #	General work timeline
9	Project directory structure, DTO classes.
10	Database structures, DAO classes.
11	Base service classes, controller classes.
12	Web pages, HTML and forms.
13	Front-end AJAX coding.
14	Data validation, final business logic, completion of features.
Final exams weeks	Completion of features, documentation, code quality checks.

Requirements

Absolute requirements

- The application's server-side code MUST be **written in PHP** with a language level **NOT LOWER THAN 8.0 AND NOT NEWER than 8.2**.
- The application MUST make use of **a MySQL or MariaDB database**.
- The application MUST run **on an Apache web server**.
- The application MUST make use of **the PDO extension** of PHP to connect to and interact with the database.
- **PHP Sessions** MUST be used to persist stateful information between requests (which user is logged in, if any, as an example).
- The application MUST be **written using the object oriented programming paradigm**.
- ***You MUST write the subsystem yourself and are forbidden from reusing previous code, copying code from outside sources or any other form of plagiarism.***
- **No external code libraries are allowed**, with the exception of the following :
  - Bootstrap front-end CSS and JS libraries
  - JQuery JS & AJAX library
  - PHP built-in extensions

Secondary requirements

- Your project must **be linked to a git repository on Github** and **updates must be pushed regularly**.
- The application MUST **work properly without major errors.**
- Database queries must **use parameterized statements**.
- With the exception of navigating between pages, interactions with the server-side of the application MUST be implemented **using asynchronous Javascript (AJAX) requests** (strong suggestion to use the provided JQuery library for this).
- **All file paths** used inside your PHP code **must be relative and not absolute**.
- **All URLs** used inside your HTML web pages **must be relative and not absolute**.

Modelization requirements

You MUST modelize (create classes for) the following elements :

- At least 3 DTO-type classes to represent the conceptual objects used by your subsystem (users, user groups, permissions),
- At least 3 DAO-type classes to implement data access functionalities (classes designed to implement operations with a data source) for the 3 DTO classes; each implementing at least the 4 basic CRUD operations,
- At least 3 Service-type classes to implement complete business logic behaviours regarding the 3 DTO classes,
- At least 3 Controller-type classes to implement API request-response functionalities for operations regarding the 3 DTO classes,

You CAN also define as many classes as you want or need for your application (examples : standalone services, router, base application class, ...).

Database

- Your database will require at least 5 tables : 3 tables for the representation of the users, user groups and permissions entities, and two 'pivot' tables for the representation of user group ↔ permission and user ↔ permission associations.
- Your database (and the script that sets it up) will need to have initial test data that includes at least 3 permissions, 2 user groups and 3 users; the users and user groups having varied associated permissions.

Web pages

Your subsystem/application MUST have at least 4 web pages :

- A login page that allows users to be authenticated and authorized.
- A users management page, that requires a logged-in user with a specific user management permission to access (redirect to login page if no user logged in, error display if permission not granted). This page MUST allow a valid logged-in user to do the following operations :
  - List the existing users (direct listing or selector).
  - Create a new user
  - Display the complete details (ALL attributes) of a certain user
  - Modify an existing user
  - Delete an existing user
  - Assign a permission to an individual user
  - Remove a permission that is assigned to an individual user
- A user groups management page, that requires a logged-in user with a specific user group management permission to access (redirect to login page if no user logged in, error display if permission not granted). This page MUST allow a valid logged-in user to do the following operations :
  - List the existing user groups (direct listing or selector).
  - Create a new user group
  - Display the complete details (ALL attributes) of a certain user group
  - Modify an existing user group
  - Delete an existing user group
  - Assign a permission to a user group
  - Remove a permission that is assigned to a user group
- A permissions management page, that requires a logged-in user with a specific permissions management permission to access (redirect to login page if no user logged in, error display if permission not granted). This page MUST allow a valid logged-in user to do the following operations :
  - List the existing permissions (direct listing or selector).

- Create a new permission
- Display the complete details (ALL attributes) of a certain permission
- Modify an existing permission
- Delete an existing permission

The 3 management pages must have a navigation bar or similar system allowing a user to navigate between them.

Code quality

Standards of code quality are enforced for the project :

- ALL functions, constants classes and methods must be documented (documentation blocks)
- Correct separation of code between public and private directory structures
- Validation is made on input data (5%)
- Object security (visibility modifiers) and encapsulation (2.5%)
- Strong typing of object properties, functions, methods, parameters, ... (2.5%)
- Functions are functionally atomic (do a single task) (5%)
- Function argument validation (validate that the values received as parameters are correct and handle if not) (5%)
- Exception handling (catch exceptions and handle them gracefully) (5%)
- Naming conventions (5%)

UI/UX

- HTML pages and their elements **are viewport-size reactive** (adapt to user screen size).
- The user interface (look and feel) should be easy to use and clear.
- The user experience (interactions between user and user interface) should be straightforward and simple enough to use the application without having prior knowledge of its workings.

Project submission

When completed, the entirety of the project's directory, including the exported database script, must be zipped and submitted on LEA. The database script MUST be exported with the following options :

- Format-specific options:
  - ***"Disable foreign key checks"*** is **ON**
- Object creation options:
  - ***"Add CREATE DATABASE / USE statement"*** is **ON**
  - ***"Add DROP TABLE / VIEW / PROCEDURE / FUNCTION / EVENT / TRIGGER statement"*** is **ON**
  - ***"IF NOT EXISTS (less efficient as indexes will be generated during table creation)"*** is **ON**

Grading

Grading approach

My main approach when grading integration projects revolves around respecting the requirements first. The project is designed to represent a task given by an employer in a context of a larger project where multiple developers could be working on various functionalities, the project itself being only one of them, and thus, it is important that the work produced respects these requirements.

Second, contrary to examinations where you are under time pressure, you have more time to do the project, and thus I do not only look at functionality; I also look at quality. The project has to work, but it also has to be done well. Code quality and user experience count for a large percentage of the grade.

Nevertheless, pay attention not to prioritize fanciness over functionality. A project that is not very beautiful UI-wise, but that is easy to use, intuitive and fully functional will get a better grade that a project with a very good looking UI but that is un-intuitive to use, half-broken and a mess. It happened in previous semesters that students made the wrong choice and spent a lot of time and efforts to do something that looked very good, but that was a mess. They were not happy with their grade...

Correction Grid

Graded elements	%
Structural requirements *2.1 Proper installation of the Web development platform and the development database management system,* *2.2 Proper installation of software and libraries* • Project built in PHP 8.0 – 8.2 • Project uses a MySQL / MariaDB database • Project runs on an Apache web server • Application is coded in the object-oriented paradigm • Use of PHP PDO extension • PHP Sessions are used • Use of jQuery to implement AJAX requests • Application runs without major errors	Absolute (failure or heavy penalties if not respected)
Respect of required application functionalities *1.1 Accurate analysis of design documents* *1.2 Proper identification of the tasks to be carried out* *3.3 Compliance with the data model* *7.5 Compliance with design documents* • All the required web pages are implemented • All the required web pages implement their required functionalities • All required object types are modelized with the required and realistic attributes • All required models (DTOs) have their database representations (tables)	20
VCS and documentation *2.3 Appropriate configuration of the version control system* *7.4 Compliance with issue tracking and version control procedures* *9.1 Proper identification of the information to be written up* *9.2 Clear record of the work carried out*	10

• Project is linked to a GitHub repository • Git commits must have a clear message representing the committed work • Functions, constants, classes and methods should be documented	
Database *3.1 Suitable creation or adaptation of the database* *3.2 Proper insertion of initial or test data* *5.3 Appropriate choice of clauses, operators, commands or parameters in database queries* *5.4 Correct handling of database data* *5.5 Appropriate use of data exchange services* • Database script is fully functional • Database script includes initial data: at least 3 permissions, 2 user groups and 3 users having various permissions are included • Database queries use the PHP PDO extension • Database queries are parameterized and use prepared statements • Database queries are fully functional	20
HTML and front-end *4.1 Appropriate use of markup language* *4.2 Suitable creation and use of style sheets* *4.4 Suitable creation of Web forms* *4.5 Adaptation of the interface based on the display format and resolution* *5.2 Proper programming of interactions between the Web interface and the user* *6.5 Web forms in compliance with usability requirements* *6.1 Correct manipulation of DOM objects* *6.2 Proper programming of asynchronous calls* • HTML web forms are used to collect user inputs • HTML web forms are fully functional and collect all the required input data for their tasks • HTML web forms use various and appropriate input elements • User experience (UX) is simple, easy and intuitive • HTML page sections should have viewport-relative sizes • Use AJAX asynchronous requests (calls) with web forms • Use AJAX asynchronous requests to modify the content of pages • AJAX calls sending data to the server use the POST HTTP method • AJAX calls are fully functional • Use of PHP sessions	20
Security and code quality *5.1 Proper programming or integration of authentication and authorization mechanisms* *5.7 Precise application of secure programming techniques* *6.4 Systematic use of Web form data validation techniques* *7.2 Thorough reviews of code and security* *7.3 Relevance of the corrective actions* • Placing an order requires customer account log-in • Customer account passwords are stored as hashed encrypted values • Correct separation of code between public and private directory structures • Validation is made on input data (form-level) • Object security (visibility modifiers) • Strong typing of object properties, functions and methods • Functions are functionally atomic • Naming conventions are respected • Validation is made on input data • Function argument validation • Exception handling	30
TOTAL	**100**