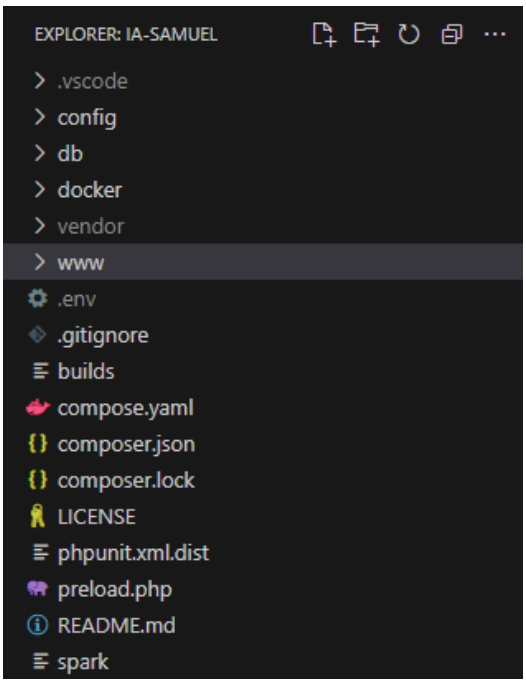
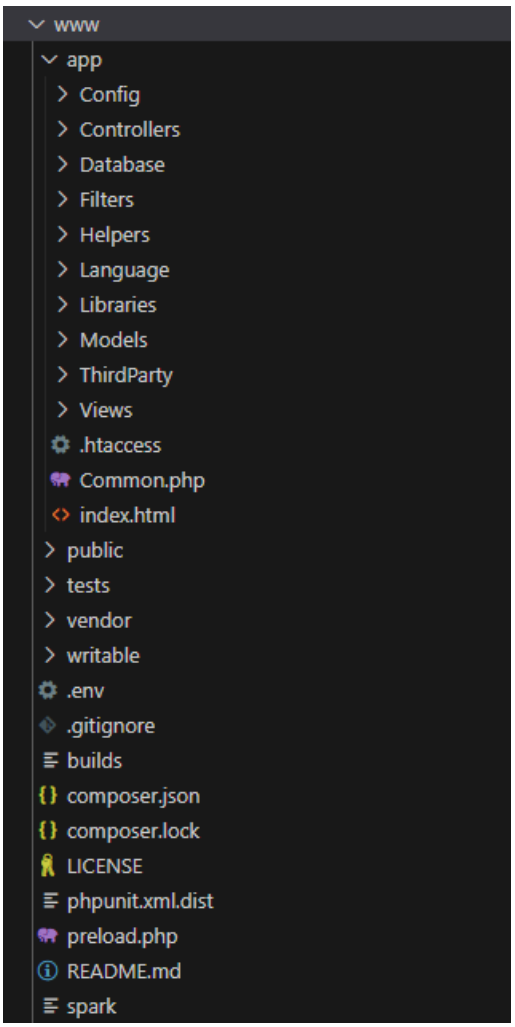
Criteria C – Development

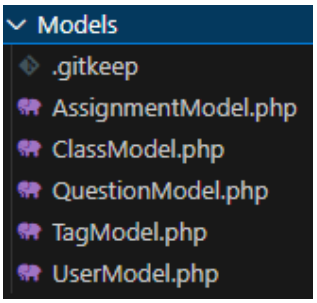
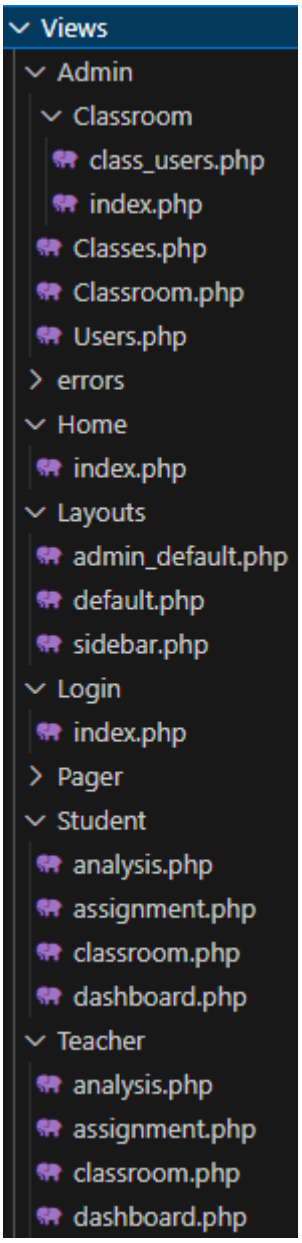
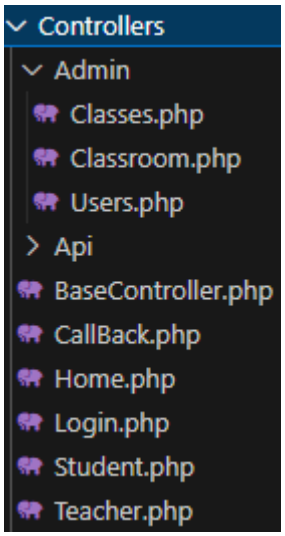
Contents

C1 – Program Structure

C1.1 – Overall Structure

Overall Project	CodeIgniter Core
	

C1.2 – MVC Structure

Model	View	Controller
		

C2 – Tools Used

	Name of Tools	Benefits
Programming Language	PHP (Back-End)	• PHP is open-source and widely used, providing a large community for support. • It is compatible with a wide range of databases, including MySQL. • PHP offers powerful library support for various functionalities, enhancing development.
	JavaScript (Front-End)	• Enables interactive web pages which improve user experience. • Widely supported by all modern web browsers without the need for any additional plugins. • Allows for the integration of additional libraries like jQuery for AJAX requests and Chart.js for data visualization.

	Name of Tools	Benefits
IDE	Visual Studio Code	• Offers a wide range of extensions and integrations that streamline the coding process. • Features like IntelliSense provide smart completions based on variable types, function definitions, and imported modules. • GitHub Copilot extension offers AI-powered code suggestions, improving coding efficiency.
Database	MySQL	• MySQL is reliable and scalable, handling large databases and high transaction volumes. • It is open-source with a strong community, providing a wealth of resources and support. • Offers comprehensive security features to protect data integrity and prevent unauthorized access.

	Name of Tools	Benefits
Framework	CodeIgniter 4 (Back-End)	• Provides a simple and elegant toolkit to create full-featured web applications. • Has excellent documentation and a straightforward MVC architecture, making it easy to learn and use. • Lightweight framework that provides high performance and fast loading times for applications.
	Bootstrap 5 (Front-End)	• Facilitates responsive design, making the website mobile-friendly and accessible on various devices. • Contains pre-designed components, which speeds up the UI development process. • It's lightweight and customizable, allowing for efficient loading times and tailored styling.

	Name of Tools	Benefits
Library	CodeIgniter Shield OAuth CodeIgniter Shield	• Provides a secure and standardized way to implement user authentication via OAuth providers like Google, and user access control. • Integrates seamlessly with CodeIgniter, maintaining consistency within the framework's ecosystem. • Simplifies the OAuth flow, reducing the complexity of adding social sign-in to the application.
	jQuery	• Simplifies the process of writing JavaScript, especially for AJAX requests and DOM manipulation. • Large ecosystem of plugins available for extended functionality. • Well-documented and widely adopted, ensuring community support and resources.

	Chart.js	• Simplifies the process of creating interactive and animated charts for data representation. • Responsive and HTML5 Canvas-based, ensuring compatibility across various devices and platforms. • Extensive documentation and community support make it easy to implement complex visualizations.

C3 – Tutorials Followed

	Description	Source
W3Schools	To learn the basic syntax and structure in PHP and JavaScript	https://www.w3schools.com/
CodeIgniter 4 documentation	To learn how does CodeIgniter 4 work in general, and how to use it efficiently	https://www.codeigniter.com/user_guide/intro/index.html
Bootstrap 5 documentation	To learn how does Bootstrap 5 work in general, and how to use it efficiently	https://getbootstrap.com/docs/5.3/getting-started/introduction/
Codeigniter Shield OAuth documentation	To learn how to implement Google OAuth in the application	https://www.shield-oauth.codeigniter4.ir/
Codeigniter Shield documentation	To learn how to implement user access control	https://shield.codeigniter.com/
jQuery documentation	To learn how to implement AJAX request in the application	https://api.jquery.com/category/ajax/
Chart.js documentation	To learn how to implement charts in the applcation	https://www.chartjs.org/docs/latest/
Stack Overflow	To find answers to specific problems encountered in the development	https://stackoverflow.com/
ChatGPT	To find more comprehensible explanations from above sources	https://chat.openai.com/

C4 – Techniques Used

- Single Sign-On (by CodeIgniter Shield OAuth) [P. 12]
    - To authenticate users with their school Gmail account
- Controller Filters (by CodeIgniter Shield) [P. 12-13]
    - To control user access to webpages
- Process Session User Data [P. 25]
    - To identify user by the ID stored in the session
- AJAX Requests (by jQuery) and Database Queries [P. 14-19, ...]
    - To create, read, update and delete records in the database without reloading the webpages
- Charts and Diagrams (by Chart.js) [P. 33-36, 57-60]
    - To represent data in an intuitive manner
- JavaScript Object Notation [P. 14-19, ...]
    - To enable lightweight and human-readable communications between frontend and backend
- Modal Boxes (by Bootstrap 5) [P. 17, 21, 54]
    - To display pop-up windows on the webpages, often enable the user to confirm their actions
- Multi-layer collections, associated arrays [P. 25-26, ...]
    - To store the records from the database, so that data can be efficiently processed and retrieved
- Tag Clouds [P. 31, 40, 43]
    - To enable user to filter contents in the webpages intuitively
- Templates [P. 13, 23-31]
    - To organize webpages in similar layout; to display navigation bar and sidebars

C5 – Code Explanation

C5.1 – Route

```
// Home
$routes->get('/', 'Home::index');

// Login
$routes->get('/login', 'Login::index');
$routes->get('/Login', 'Login::index');

// Access Denied (for outsiders)
$routes->get('/accessDenied', 'AccessDenied::index');

// Student (Ajax) and Teacher (Ajax) are seperated so that access right can be controlled

// Student
$routes->get('/student/dashboard', 'Student::index');
$routes->get('/student/classroom/(:num)', 'Student::classroom/$1');
$routes->get('/student/assignment/(:num)', 'Student::assignment/$1');
$routes->get('/student/analysis', 'Student::analysis');

// Student Ajax
$routes->post('/student/questionUpdate', 'Student::questionUpdate');
$routes->post('/student/tagAdd', 'Student::tagAdd');
$routes->post('/student/tagDelete', 'Student::tagDelete');

// Teacher
$routes->get('/teacher/dashboard', 'Teacher::index');
$routes->get('/teacher/classroom/(:num)', 'Teacher::classroom/$1');
$routes->get('/teacher/assignment/(:num)', 'Teacher::assignment/$1');
$routes->get('/teacher/analysis', 'Teacher::analysis');

// Teacher Ajax
$routes->post('/teacher/assignmentAdd', 'Teacher::assignmentAdd');
$routes->post('/teacher/assignmentDelete', 'Teacher::assignmentDelete');
$routes->post('/teacher/assignmentEdit', 'Teacher::assignmentEdit');
$routes->post('/teacher/assignmentTopicUpdate', 'Teacher::assignmentTopicUpdate');
$routes->post('/teacher/questionAdd', 'Teacher::questionAdd');
$routes->post('/teacher/questionDelete', 'Teacher::questionDelete');
$routes->post('/teacher/questionUpdate', 'Teacher::questionUpdate');
$routes->post('/teacher/tagAdd', 'Teacher::tagAdd');
$routes->post('/teacher/tagDelete', 'Teacher::tagDelete');

// Admin
$routes->get('/admin/dashboard', 'Admin\Users::index');
$routes->get('/admin/classes', 'Admin\Classes::index');
$routes->get('/admin/classroom', 'Admin\Classroom::index');

// Admin ajax
$routes->get('/admin/users/loadUsers', 'Admin\Users::loadUsers');
$routes->post('/admin/users/addUser', 'Admin\Users::addUser');
$routes->post('/admin/users/deleteUser', 'Admin\Users::deleteUser');
$routes->post('/admin/users/editUser', 'Admin\Users::editUser');

$routes->get('/admin/classes/loadClasses', 'Admin\Classes::loadClasses');
$routes->post('/admin/classes/addClass', 'Admin\Classes::addClass');
$routes->post('/admin/classes/deleteClass', 'Admin\Classes::deleteClass');
$routes->post('/admin/classes/editClass', 'Admin\Classes::editClass');

$routes->get('/admin/classroom/loadUsers', 'Admin\Classroom::loadUsers');
$routes->get('/admin/classroom/getClassID', 'Admin\Classroom::getClassID');
$routes->get('/admin/classroom/getClassName', 'Admin\Classroom::getClassName');
$routes->post('/admin/classroom/addUser', 'Admin\Classroom::addUser');
$routes->post('/admin/classroom/deleteUser', 'Admin\Classroom::deleteUser');
```

Path

Controller

Controller's method

Get view

Post to the controller method

Code segment 1 – Routes of All Webpages

Above are the route definitions of the application. Whenever users ‘get’ the paths shown above, corresponding controllers will return the correct view; whenever users ‘post’ to the path shown above, corresponding controllers will be executed with the posted variables.

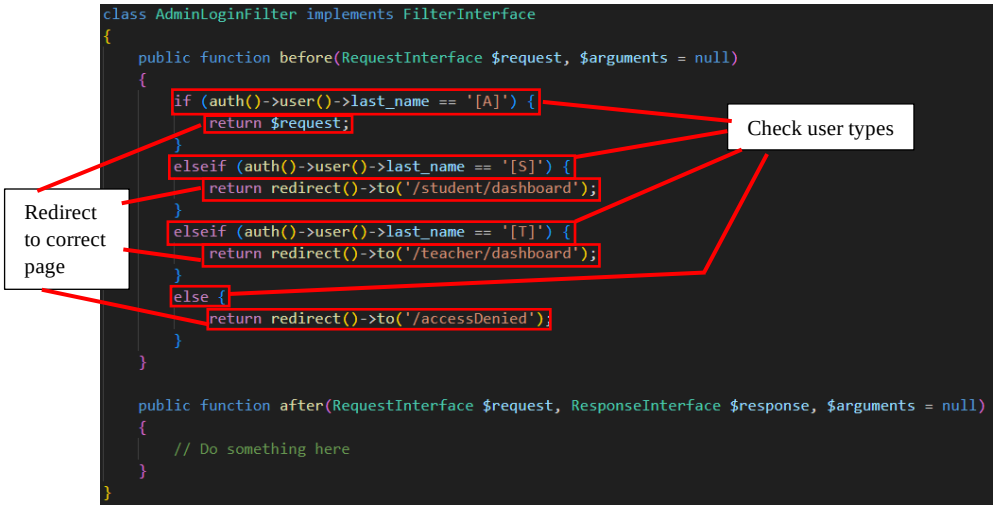
C5.2 – Login

In the application, users will be first directed to the login page (i.e., ‘/login’) if they did not.

It is expected that all users will login with their school Google Account. Therefore, Google's OAuth 2.0 is currently the only authentication method in the application. To authenticate successfully, Shield OAuth, a CodeIgniter 4 library, is used. After installing it according to the documentation, users can now login with their school Google Account, and their ‘id’, ‘avatar’, ‘unsername’, ‘first_name’, ‘last_name’, ‘created_at’ will be recorded in the ‘users’ table in the database.

In user’s School Google Account, ‘last_name’ is set according to the role of the user by our school IT administrators (i.e., it will be set to be ‘[S]’, ‘[T]’ or ‘[A]’, indicating that if the user is student, teacher or admin). Since it cannot be changed by the user, it is used for the access right control of the application.

By creating `AdminLoginFilter.php`, `StudentLoginFilter.php` and `TeacherLoginFilter.php` in `www/app/Filters/` in a similar format as shown:



Code segment 2: Format of Login Filters

Then by adding these lines in `www/app/Config/Filters.php`:

```
public array $filters = [
    'AdminLoginFilter' => [
        'before' => ['admin/*']
    ],
    'StudentLoginFilter' => [
        'before' => ['student/*']
    ],
    'TeacherLoginFilter' => [
        'before' => ['teacher/*']
    ]
];
```

Webpage directories that are allowed to access after passing the filter

Code segment 3: Filter Config

Now, pages with URL start with '/admin/' can only accessed by authorized admins; pages with URL start with '/student/' can only accessed by authorized students or admins; pages with URL start with '/teacher/' can only accessed by authorized teachers or admins. Also, students and teachers will be redirected to the correct page if they accidentally input the page path wrongly. Finally, unauthorized access from outsiders will be redirected to a dedicated page.

C5.3 – Admins' pages

Admins are responsible for CRUDing the 'users' and 'classes' tables in the database. To achieve this, there are 3 controllers and corresponding views:

Path	Controller
'/admin/dashboard'	'Admin\Users::index'
'/admin/classes'	'Admin\Classes::index'
'/admin/classroom'	'Admin\Classroom::index'

5.3.1 Templates

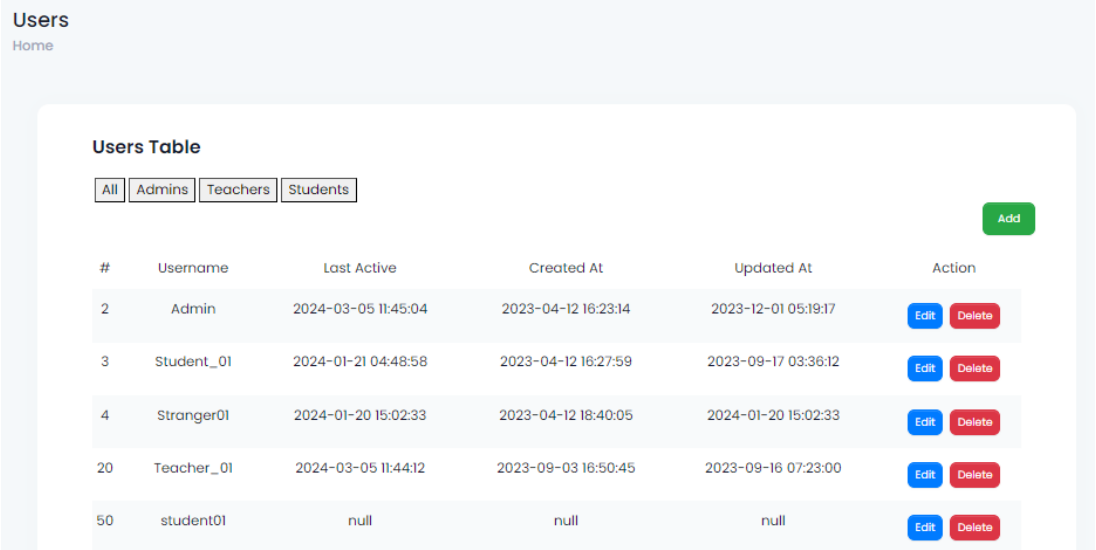
In all admin views, a default layout (template) is used: `www/app/Views/Layouts/admin_default.php`. In the template, the line:

```
<?= $this->renderSection("content") ?>
```

is used to render the content in different views (with `<?= $this->extend("Layouts/admin_default") ?>` `<?= $this->section("content") ?>` and `<?= $this->endSection() ?>`)

5.3.2 Admin\Users

In the view returned by this controller, the 'users' table in the database is shown:



Actual webpage 1: Admin's Users Page

It is done by following code segments:

Get user records from database (pager included) using CodeIgniter 4 built in database query methods

```php
public function __construct()
{
    $this->model = new \App\Models\UserModel;
}

public function index()
{
    $users = $this->model->orderBy('id')->paginate(10);
    $pager = $this->model->pager;

    $data = [
        'users' => $users,
        'pager' => $pager
    ];

    return view('Admin/Users', $data);
}
```

Return user records to the view

Code segment 4: Controller Method – Getting records

```
<script>
    // sh
    var showUserType = "";
    function changeShowUserType(userType) {
        showUserType = userType;
    }

    // savedID is                                  he user to be edited or deleted
    var savedID;
</script>
```

Initiate the variable for filtering users by userType

Initiate the function for changing the filter

Initiate the variable for editing/deleting user by userID

Code segment 5: View – Initialization of variables & function

```
<!--Load jQuery for ajax request-->
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script>
    // Ajax request to load users
    function loadUsers() {
        $.ajax({
            url: "/admin/users/loadUsers",
            type: "GET",
            data: {
                showUserType: showUserType
            },
            success: function(response) {
                $.each(response.users, function(key, value) {
                    var row = '<tr>';
                    row += '<th scope="row" class="text-center">' + value.id + '</th>';
                    row += '<td class="text-center">' + value.username + '</td>';
                    row += '<td class="text-center">' + value.last_active + '</td>';
                    row += '<td class="text-center">' + value.created_at + '</td>';
                    row += '<td class="text-center">' + value.updated_at + '</td>';
                    row += '<td class="text-center"> <div class="d-flex justify-content-center">';
                    row += '<button data-bs-toggle="modal" data-bs-target="#editUserModal" class="saveIdRequired verySmallButton m-1 btn-primary">Edit</button>';
                    row += '<button data-bs-toggle="modal" data-bs-target="#deleteUserModal" class="saveIdRequired verySmallButton m-1 btn-danger">Delete</button>';
                    row += '</td>';
                    row += '</tr>';
                    $('tbody').append(row);
                });
            }
        });
    }
```

jQuery is called

AJAX request to get user records from the controller

Pass data of current filter to the controller

If the request success, display each user record as a row in a table

Code segment 6: View – AJAX request of loading records

```
// Load users
public function loadUsers()
{
    // get showUserType from ajax request
    $showUserType = $this->request->getVar('showUserType');
    if ($showUserType != '') {
        $users = $this->model->where('last_name', $showUserType)->orderBy('id')->paginate(10);
    }
    else {
        $users = $this->model->orderBy('id')->paginate(10);
    }

    $pager = $this->model->pager;

    $data = [
        'users' => $users,
        'pager' => $pager
    ];

    // response to ajax request
    return $this->response->setJSON($data);

}
```

Get user records according to the filter from the model

Respond to the AJAX request in JSON

Code segment 7: Controller Method – Handling AJAX request of loading records

Buttons on the view that change the filter onclick

```
<div class="tagcloud">
    <button onclick="changeShowUserType('')" class="tag-cloud-link changeShowUserType">All</button>
    <button onclick="changeShowUserType('[A]')" class="tag-cloud-link changeShowUserType">Admins</button>
    <button onclick="changeShowUserType('[T]')" class="tag-cloud-link changeShowUserType">Teachers</button>
    <button onclick="changeShowUserType('[S]')" class="tag-cloud-link changeShowUserType">Students</button>
</div>
```

Code segment 8: View – Choices of the filter

```
// load users when the showUserType is changed
$('.changeShowUserType').click(function() {
    $('tbody').empty();
    loadUsers();
});
```

Whenever the buttons are clicked, the table is loaded again with the filter

Code segment 9: View – Reloading the table onClick

When the addButton is clicked...

```
$(document).on('click', '#addButton', function() {
    $.ajax({
        url: "/admin/users/addUser",
        type: "POST",
        data: {
            userType: showUserType
        },
        success: function(response) {
            $('tbody').empty();
            loadUsers();
        }
    });
});
```

Send s AJAX (post) request to the controller to add a user

Pass data of current filter to the controller

If the request success, reload the table

Code segment 10: View – AJAX request for inserting record

```
// Add user according to userType
public function addUser()
{
    // get userType from ajax request
    $userType = $this->request->getVar('userType');
    $user = [
        'last_name' => $userType,
        'created_at' => date('Y-m-d H:i:s'),
    ];

    // Insert model
    $this->model->insert($user);

    // Return a response indicating success
    $response = array('success' => true);
    return $this->response->setJSON($response);
}
```

Add user to the model, according to current datetime, and the current filter given in the request

Respond to the AJAX request

Code segment 11: Controller Method – Handling the AJAX request for inserting record

Bootstrap 5 are called through CDN

Bootstrap 5 modals are used

Button to confirm deletion of user's record

Input boxes for updating user's record

```
<!--begin::Bootstrap Scripts Bundle-->
<script defer src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm" crossorigin="anonymous"></script>
<!--end::Bootstrap Scripts Bundle-->
<!--begin::Modal-->

<!--begin::deleteUserModal-->
<div class="modal fade" id="deleteUserModal" tabindex="-1" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel"> Delete User</h1>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
                Confirm to delete this user?
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
                <button id="deleteButton" type="button" class="btn btn-danger" data-bs-dismiss="modal">Delete</button>
            </div>
        </div>
    </div>
</div>
<!--end::deleteUserModal-->

<!--begin::editUserModal-->
<div class="modal fade" id="editUserModal" tabindex="-1" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel"> Edit User</h1>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="username" id="username" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="first_name" id="first_name" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="last_name" id="last_name" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="classID_1" id="classID_1" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="classID_2" id="classID_2" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="classID_3" id="classID_3" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="classID_4" id="classID_4" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="classID_5" id="classID_5" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="classID_6" id="classI                                <br>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
                <button id="editButton" type="button" class="btn btn-danger" data-bs-dismiss="modal">Edit</button>
            </div>
        </div>
    </div>
</div>
<!--end::editUserModal-->

<!--end::Modal-->
```

Code segment 12: View – Modals for confirming the deletion & update of record

```
// save id of the user to savedID by looking for the closest <tr> tag
$(document).on('click', '.saveIdRequired', function() {
    savedID = $(this).closest('tr').find('th').text();
});

x request to delete user
$(document).on('click', '#deleteButton', function() {

    // get savedID
    id = savedID;

    $.ajax({
        url: "/admin/users/deleteUser",
        type: "POST",
        data: {
            id: id
        },
        success: function(response) {
            $('tbody').empty();
            loadUsers();
        }
    });
});

 request to edit user
$(document).on('click', '#editButton', function() {

    // get savedID
    id = savedID;

    $.ajax({
        url: "/admin/users/editUser",
        type: "POST",
        data: {
            id: id,
            username: $('#username').val(),
            first_name: $('#first_name').val(),
            last_name: $('#last_name').val(),
            classID_1: $('#classID_1').val(),
            classID_2: $('#classID_2').val(),
            classID_3: $('#classID_3').val(),
            classID_4: $('#classID_4').val(),
            classID_5: $('#classID_5').val(),
            classID_6: $('#classID_6').val()
        },
        success: function(response) {
            $('tbody').empty();
            loadUsers();
        }
    });
});
```

When the add button is clicked...

Save id of the record when the “update” or “delete” button of the record is clicked

Send AJAX request to delete the user with the corresponding id

If the request success, reload the table

When the edit button is clicked...

Send AJAX request to update the user with the corresponding id, and all other attributes from the modal

If the request success, reload the table

Code segment 13: View – AJAX requests for deleting & updating record

Delete user from the model, according to id given in the request

Respond to the AJAX request

Update user in the model, according to id and other attributes given in the request

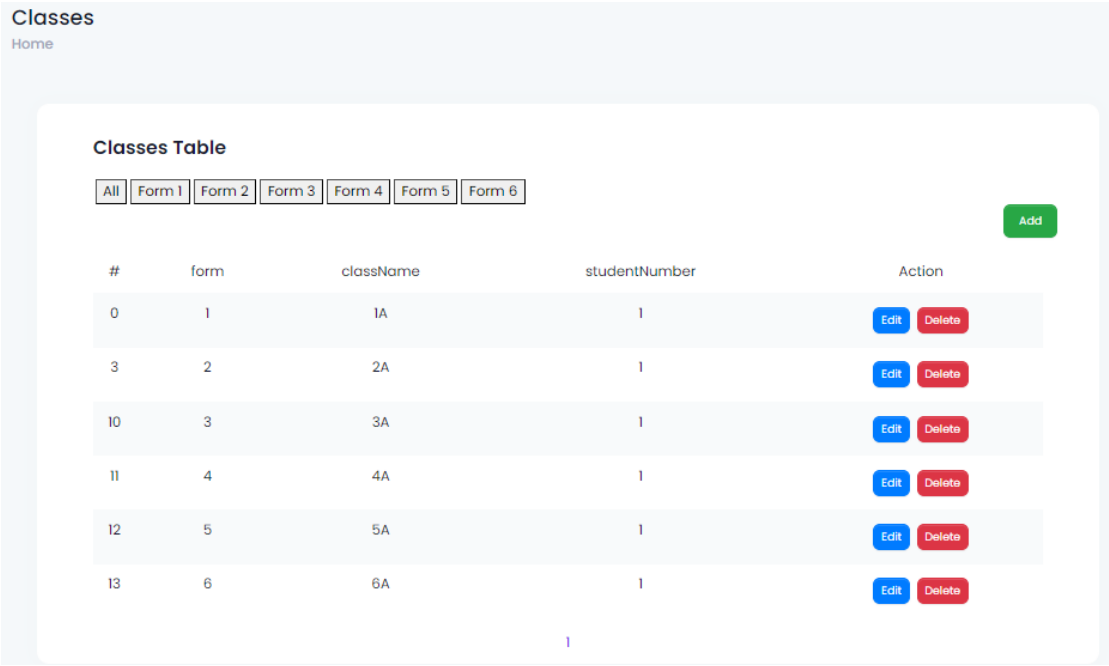
Respond to the AJAX request

```php
// Delete user
public function deleteUser()
{
    // get id from ajax request
    $id = $this->request->getVar('id');

    // Delete model
    $this->model->delete($id);

    // Return a response indicating success
    $response = array('success' => true);
    return $this->response->setJSON($response);
}

// Edit user
public function editUser()
{
    // get data from ajax request
    $id = $this->request->getVar('id');
    $username = $this->request->getVar('username');
    $first_name = $this->request->getVar('first_name');
    $last_name = $this->request->getVar('last_name');
    $classID_1 = $this->request->getVar('classID_1');
    $classID_2 = $this->request->getVar('classID_2');
    $classID_3 = $this->request->getVar('classID_3');
    $classID_4 = $this->request->getVar('classID_4');
    $classID_5 = $this->request->getVar('classID_5');
    $classID_6 = $this->request->getVar('classID_6');

    // Update model
    $user = [
        'username' => $username,
        'first_name' => $first_name,
        'last_name' => $last_name,
        'classID_1' => $classID_1,
        'classID_2' => $classID_2,
        'classID_3' => $classID_3,
        'classID_4' => $classID_4,
        'classID_5' => $classID_5,
        'classID_6' => $classID_6,
        'updated_at' => date('Y-m-d H:i:s'),
    ];
    $this->model->update($id, $user);

    // Return a response indicating success
    $response = array('success' => true);
    return $this->response->setJSON($response);
}
```

Code segment 14: Controller Methods – Handling AJAX requests for deleting & updating record

5.3.3 Admin\Classes

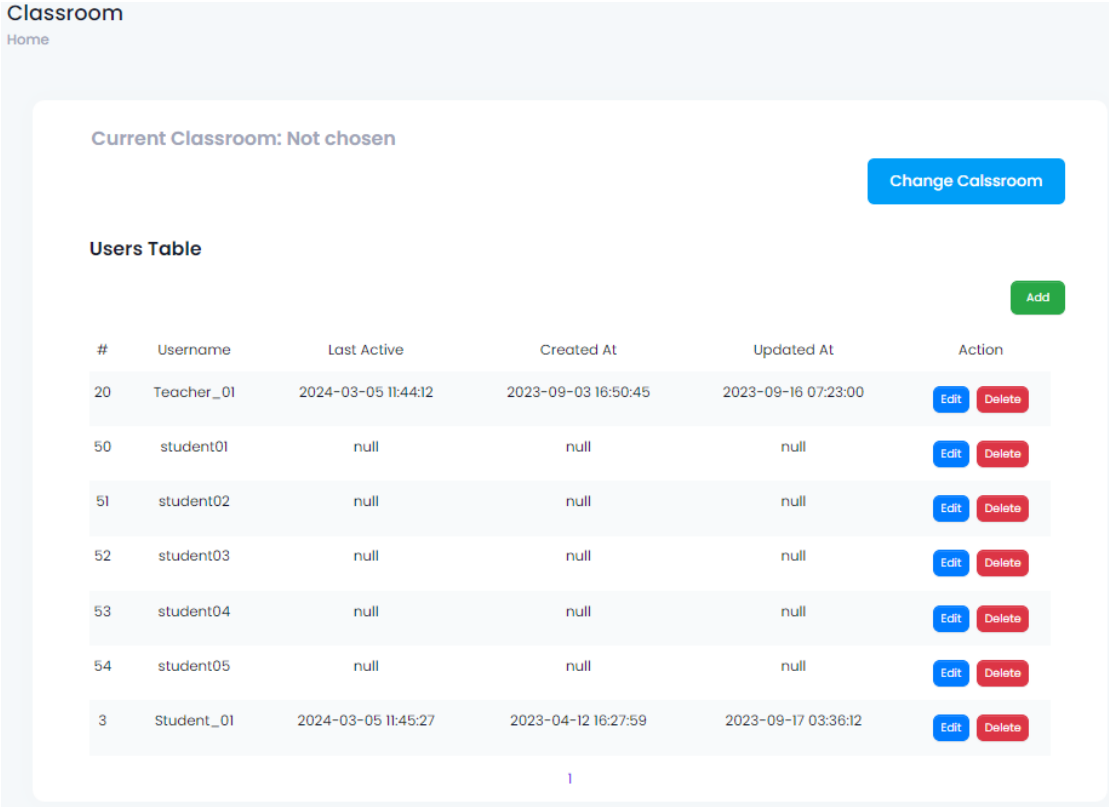
In the view returned by this controller, the ‘classes’ table in the database is shown. This controller and the corresponding view are basically the same as 5.3.2, except the admin now CRUD the ‘classes’ table instead of the ‘users’ table. Also, using the same logic, the admin can filter class records by their ‘form’.



Actual webpage 2: Admin’s Classes Page

5.3.4 Admin\Classroom

In the view returned by this controller, a table of users that are in a specific class is shown. This controller and the corresponding view are basically the same as 5.3.2, except the admin now CRUD the users’ records that are in a specific class (i.e., admin can filter user records by their classes).



Actual webpage 3: Admin's Classroom Page

It is done by the following additional code segments:

```
<!--begin::changeShowClassroomModal-->
<div class="modal fade" id="changeShowClassroomModal" tabindex="-1" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel"> Change Classroom</h1>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="classID" id="classID" row="1"></textarea> <br>
                <textarea type="text" class="form-control form-control-solid ps-15" placeholder="className" id="className" row="1"></textarea> <br>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
                <button id="ChangeButton" type="button" class="btn btn-danger" data-bs-dismiss="modal">Change</button>
            </div>
        </div>
    </div>
</div>
<!--end::changeShowClassroomModal-->
```

The modal that allows user input to change the selected class by classID or className

Code segment 15: View – Modal for the input of class details

The ajax request for changing the selected class.

Get user input

An AJAX request to get classID by className

Display className of selected class on view

Reload user table

An AJAX request to get className by classID

```
// send AJAX request to change currentClassroom
$(document).on('click', '#ChangeButton', function() {
    var classID = $('#classID').val();
    var className = $('#className').val();

    if (classID !== '') {
        var currentClassroomID = classID;

        // AJAX request to get className
        $.ajax({
            url: "/admin/classroom/getClassName",
            type: "GET",
            data: {
                classID: classID
            },
            success: function(response) {
                var className = response.className;
                var currentClassroomName = className;
                $('#currentClassroomName').text(currentClassroomName);
                $('tbody').empty();
                loadUsers(currentClassroomID);
            },
            error: function(error) {
                alert("No class can be found with this classID");
            }
        });
    }
    else if (className !== '') {
        var currentClassroomName = className;

        // AJAX request to get classID
        $.ajax({
            url: "/admin/classroom/getClassID",
            type: "GET",
            data: {
                className: className
            },
            success: function(response) {
                var currentClassroomID = response.classID;
                $('#currentClassroomName').text(currentClassroomName);
                $('tbody').empty();
                loadUsers(currentClassroomID);
            },
            error: function(error) {
                alert("No class can be found with this className");
            }
        });
    }
    else {
        var currentClassroomName = "Not chosen";
        var currentClassroomID = null;
        $('#currentClassroomName').text(currentClassroomName);
    }

    $('#currentClassroomName').text(currentClassroomName);
});
```

Code segment 16: View – AJAX request to change the selected class

```
// handle ajax request to get classID by className
public function getClassID()
{
    $className = $this->request->getVar('className');
    $classID = $this->ClassModel->where('className', $className)->orderBy('created_at', 'DESC')->first(true)['classID'];
    return $this->response->setJSON(['classID' => $classID]);
}

// handle ajax request to get className by classID
public function getClassName()
{
    $classID = $this->request->getVar('classID');
    $className = $this->ClassModel->where('classID', $classID)->orderBy('created_at', 'DESC')->first(true)['className'];
    return $this->response->setJSON(['className' => $className]);
}
```

Controller's responds to the ajax request for getting classID by className and getting className by classID

Code segment 17: Controller Methods – Getting ClassID by ClassName and vice versa

C5.4 – Students' pages and Teachers' pages

In short, students and teachers interact with the 'assignments', 'questions', and 'tags' tables in the database. In total, there are 8 controllers and corresponding views responsible for this:

Path	Controller
'/student/dashboard'	'Student::index'
'/teacher/dashboard'	' Teacher::index'
'/student/classroom/(:num)'	' Student::classroom/$1'
'/teacher/classroom/(:num)'	' Teacher::classroom/$1'
'/student/assignment/(:num)'	' Student::assignment/$1'
'/teacher/assignment/(:num)'	' Teacher::assignment/$1'
'/student/analysis'	' Student::analysis'
'/teacher/analysis'	' Teacher::analysis'

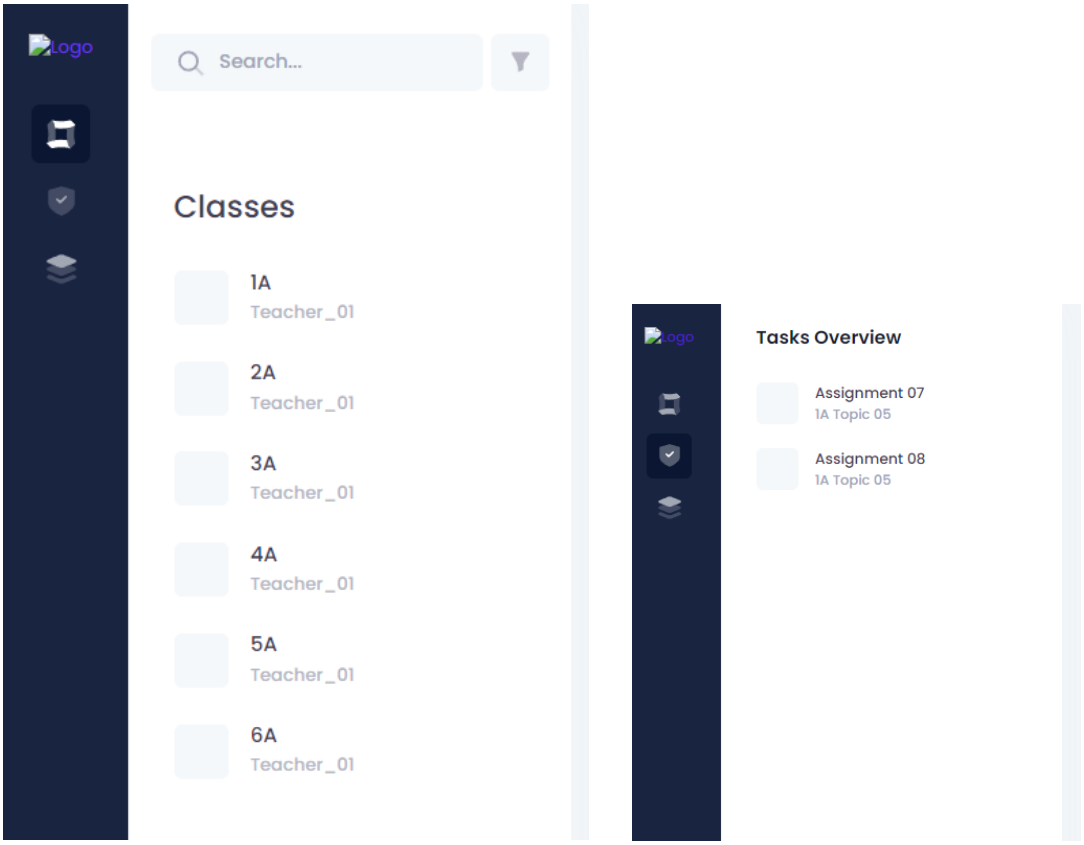
The path takes in an integer as parameter and passes it to the controller method

5.4.1 – Templates

Similar to 5.3.1, template is used in all the views of students and teachers, which is ` www/app/Views/Layouts/default.php`. Additionally, in some views, there is a sidebar on the right-hand side, which is also a template by itself: `www/app/Views/Layouts/sidebar.php`.

5.4.2 – Navigation bar

In all the views of students and teachers, there is a navigation bar on the left-hand side, which is included in the template. It has two main features:



Actual webpage 4 (left)**, Actual webpage 5** (right)**:** Navigation bar in students' and teachers' pages

One is to display the classes that the users are in (Actual webpage 1), another is to display the tasks that are not completed (Actual webpage 2). The code that achieves this is similar between teachers' views and students' views, but there is a slight difference in the algorithm that determines which tasks are not completed.

Following are the code segments that are responsible for this:

```php
$id = auth()->user()->id;

$model = new \App\Models\UserModel();
[$data['userType'], $classIDs] = $model->userConfiguration($id);

$model = new \App\Models\ClassModel();
[$classes, $classNames] = $model->classroomConfiguration_1($classIDs);

// find teacherIDs according to classIDs
$model = new \App\Models\UserModel();
$teacherIDs = [];
foreach ($classes as $class) {
    $teacherIDs[] = $class['teacherID'];
}

// find teacherNames with teacherIDs
$teacherNames = $model->findUsername($teacherIDs);

// output the collection required to load the classrooms in the sidebar
$model = new \App\Models\ClassModel();
$data['loadClassrooms'] = $model->classroomConfiguration_2($classes, $classNames, $teacherNames);

// output the array required to load all the assignments of user
$model = new \App\Models\AssignmentModel();
$results = [];
foreach ($classIDs as $classID) {
    $result = $model->assignmentConfiguration($id, $data['userType'], $classID);
    $results[$classID] = [
        'topics' => $result[0],
        'loadAssignments' => $result[1],
    ];
}
// Assign the final results back to the variables
$topics = [];
$loadAssignments = [];
foreach ($results as $classID => $result) {
    $topics[$classID] = $result['topics'];
    $loadAssignments[$classID] = $result['loadAssignments'];
}

// load data for the to-do list and calculate overall studentProgress
// also load data for Task Review in sidbar
foreach ($loadAssignments as $classID => $assignments) {
    foreach ($assignments as $assignment) {
        if ($assignment['studentProgress'] != 100) {
            $data['toDoList'][$classID][] = [
                'assignmentID' => $assignment['assignmentID'],
                'assignmentName' => $assignment['assignmentName'],
                'topic' => $assignment['topic'],
                'dueDate' => $assignment['dueDate'],
                'studentProgress' => $assignment['studentProgress'],
            ];
        }
    }
}
```

Identify user by id saved in the session

Identify $date['userType'] of the user and the $classIDs of the classes that the user belongs to by the $id

Using the $classIDs, store the corresponding classes as objects in the collection $classes, and store the names of the classes separately in array $classNames

Configure array $teacherIDs

Configure array $teacherNames

Configure collection $data['loadClassrooms'] that can be accessed on the View

Configure associative collection $results that store another associative collection (store topics and the necessary details of each assignment in the class) as the value and the corresponding $classID as the key

From $results, configure associative collection $loadAssignments that store the necessary details of each assignment in the class as value and the corresponding $classID as the key, and configure associative array $topics that store all the topics of the assignments in the same way

The configuration of $results and $loadAssignments may seem redundant, but in fact the method assignmentConfiguration() is reused in this part, which is developed earlier than this part (Code segment 32)

If assignment not done, add to to-do list

Configure associative collection $data['toDoList'] that can be accessed on the View, which stores $classID as keys and the necessary details of each assignment in the class as values

Code segment 18: Student's Controller Method – Loading navigation bar

```php
$id = auth()->user()->id;

$model = new \App\Models\UserModel();
[$data['userType'], $classIDs] = $model->userConfiguration($id);

$model = new \App\Models\ClassModel();
[$classes, $classNames] = $model->classroomConfiguration_1($classIDs);

// find teacherIDs according to classIDs
$model = new \App\Models\UserModel();
$teacherIDs = [];
foreach ($classes as $class) {
    $teacherIDs[] = $class['teacherID'];
}

// find teacherNames with teacherIDs
$teacherNames = $model->findUsername($teacherIDs);

// output the array required to load the classrooms in the sidebar
$model = new \App\Models\ClassModel();
$data['loadClassrooms'] = $model->classroomConfiguration_2($classes, $classNames, $teacherNames);


// output the array required to load all the assignments
$model = new \App\Models\AssignmentModel();
$results = [];
foreach ($classIDs as $classID) {
    $result = $model->assignmentConfiguration($id, $data['userType'], $classID);
    $results[$classID] = [
        'topics' => $result[0],
        'loadAssignments' => $result[1],
        'studentAssignments' => $result[2]
    ];
}
// Assign the final results back to the variables
$topics = [];
$loadAssignments = [];
$studentAssignments = [];

foreach ($results as $classID => $result) {
    $topics[$classID] = $result['topics'];
    $loadAssignments[] = $result['loadAssignments'];
    $studentAssignments[$classID] = $result['studentAssignments'];
}


// load data for the to-do list and calculate overall teacherProgress
// also load data for the Task Review in sidebar
foreach ($studentAssignments as $classID => $classAssignments) {
    foreach ($classAssignments as $Topic_Name => $assignments) {
        $overallTeacherProgress = 0;
        $count = 0;
        foreach ($assignments as $assignment) {
            if ($assignment['teacherProgress'] != 100) {
                $data['toDoList'][$classID][$Topic_Name] = [
                    'assignmentID' => $assignment['assignmentID'],
                    'topic' => $assignment['topic'],
                    'assignmentName' => $assignment['assignmentName'],
                    'dueDate' => $assignment['dueDate'],
                ];
                $overallTeacherProgress += $assignment['teacherProgress'];
                $count++;
            }
        }
        if ($count > 0) {
            $averageTeacherProgress = round($overallTeacherProgress / $count);
            $data['toDoList'][$classID][$Topic_Name]['overallTeacherProgress'] = $averageTeacherProgress;
        }
    }
}
```

Overall quite similar to previous code segment (for student)

However, since controller method cannot be reused in CodeIgniter 4, a great proportion of codes is duplicate

Load students' works

Calculate overall progress of marking students' works in each assignment

Code segment 19: Teacher's Controller Method – Loading navigation bar

```php
public function userConfiguration($id)
{
    //find user record
    $user = $this->find($id);

    // identify user type
    if ($user['last_name'] == '[S]') {
        $userType = 'student';
    } elseif ($user['last_name'] == '[T]') {
        $userType = 'teacher';
    } elseif ($user['last_name'] == '[A]') {
        $userType = 'admin';
    }

    // list of classIDs for the user
    $classIDs = [
        $user['classID_1'],
        $user['classID_2'],
        $user['classID_3'],
        $user['classID_4'],
        $user['classID_5'],
        $user['classID_6'],
    ];

    return [$userType, $classIDs];
}

public function findUsername($ids)
{
    //find username by id
    $usernames = [];
    foreach ($ids as $id) {
        $usernames[] = $this->find($id)['username'];
    }

    return $usernames;
}
```

Code segment 20: Model Methods – Configuring user

```php
public function classroomConfiguration_1($classIDs)
{
    // find classes according to classID
    $classes = $this->find($classIDs);

    // find class names according to classID
    $classNames = [];
    foreach ($classes as $class) {
        $classNames[] = $class['className'];
    }

    return [$classes, $classNames];
}

public function classroomConfiguration_2($classes, $classNames, $teacherNames)
{
    // Output the array required to load the classrooms (combine the three arrays)
    $loadClassrooms = [];
    for ($i = 0; $i < count($classes); $i++) {
        $loadClassroom = [
            'classID' => $classes[$i]['classID'],
            'className' => $classNames[$i],
            'teacherName' => $teacherNames[$i]
        ];
        $loadClassrooms[$i] = $loadClassroom;
    }
    return $loadClassrooms;
}
```

Code segment 21: Model Methods – Configuring classrooms

```php
public function assignmentConfiguration($userID, $userType, $classID)
{
    // array of assignmentIDs for the class
    $assignments = $this->where('classID', $classID)->findAll();

    // array of topicNames (concatenate topic and name) of assignments
    $topicNames = [];
    foreach ($assignments as $assignment) {
        $topicNames[] = ['topic' => $assignment['topic'], 'assignmentName' => $assignment['assignmentName']];
    }

    // remove duplicate in the multi-dimentional array
    $topicNames = array_map("unserialize", array_unique(array_map("serialize", $topicNames)));

    // for student
    if ($userType == 'student')
    {
        $ownedAssignments = [];

        // check if there is any assignment owned by the student for each topicName
        // always select the latest updated one if multiple records are found
        foreach ($topicNames as $topicName) {
            $owned = $this->where('classID', $classID)
                ->where('topic', $topicName['topic'])
                ->where('assignmentName', $topicName['assignmentName'])
                ->where('userID', $userID)->orderBy('updated_at', 'desc')
                ->first();
            if ( $owned != null)
            {
                $ownedAssignments[] = $owned;
            }
            // if not owned, assume the student owned a copy
            // (in future section, it is set that if student clicked into assignment that is not owned by himself/herself,
            //  there will be a copy of the assingment created, which will be owned by the student)
            else {
                $ownedAssignments[] = $this->where('classID', $classID)
                    ->where('topic', $topicName['topic'])
                    ->where('assignmentName', $topicName['assignmentName'])
                    ->where('userID !=', $userID)
                    ->orderBy('updated_at', 'desc')
                    ->first();
            }
        }
        $assignments = $ownedAssignments;

        // this array is not for students, just to define the variable for successful loading
        $studentAssignments = [];
    }
    // for teacher
    elseif ($userType == 'teacher')
    {
        $studentAssignments = [];
        $notOwneds = [];

        // create two arrays that store assignments that are owned and not owned by students
        // (not owned --> not updated by student)
        // aways select the latest updated one if multiple records are found
        foreach ($topicNames as $topicName) {
            $owned = $this->where('classID', $classID)
                ->where('topic', $topicName['topic'])
                ->where('assignmentName', $topicName['assignmentName'])
                ->where('userID !=', $userID)
                ->orderBy('updated_at', 'desc')
                ->findAll();
            if ( $owned != null)
            {
                $studentAssignments[$topicName['topic']."/".$topicName['assignmentName']] = $owned;
            }
            // if assignments are not owned by students, it is owned by teacher, so the userID will be equal to $id
            $notOwneds[] = $this->where('classID', $classID)
                ->where('topic', $topicName['topic'])
                ->where('assignmentName', $topicName['assignmentName'])
                ->where('userID', $userID)
                ->orderBy('updated_at', 'desc')
                ->first();
        }

        // teachers can only see assignments that are not owned by students on the classroom page,
        // teacher has to enter assignment in order to see the assignments owned by students
        $assignments = $notOwneds;

        // array $studentAssignments is for teacher to mark students' works
    }
```

Array $topicNames is configured and used so that each unique assignment in the class is iterated through

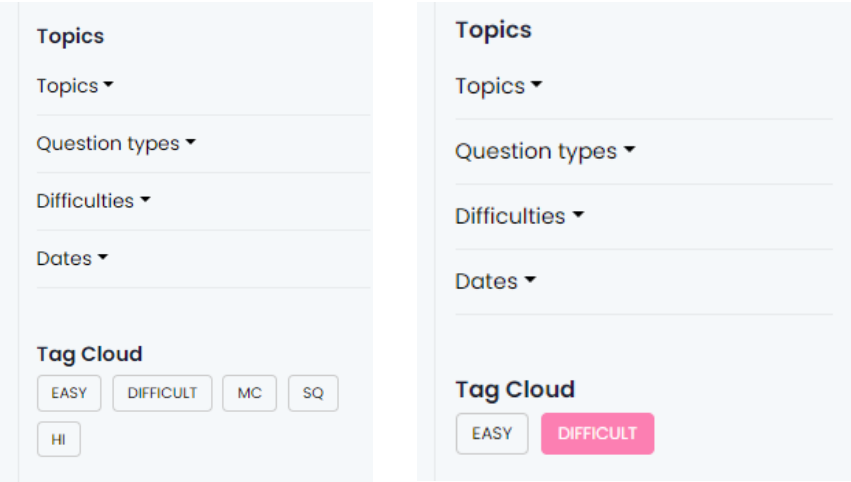
```php
    // list of topic names for assignments
    $topics = [];
    foreach ($assignments as $assignment) {
        // check if assignment is null
        // (theoretically, in normal use, it should not be null, i.e., when assignment is first created by teacher)
        if ($assignment == null)
        {
            continue;
        }
        $topics[] = $assignment['topic'];
    }

    // the array required to load the assignments
    $loadAssignments = [];
    foreach ($assignments as $assignment) {
        // check if assignment is null (theoretically, in normal use, it should not be null,
        // i.e., when assignment is first created by teacher)
        if ($assignment == null)
        {
            continue;
        }
        $loadAssignments[] = [
            'assignmentID' => $assignment['assignmentID'],
            'assignmentName' => $assignment['assignmentName'],
            'topic' => $assignment['topic'],
            'dueDate' => $assignment['dueDate'],
            'totalMark' => $assignment['totalMark'],
            'maxMark' => $assignment['maxMark'],
            'studentProgress' => $assignment['studentProgress'],
            // 'assignmentTag' => $assignment['assignmentTag'],
        ];
    }

    return [$topics, $loadAssignments, $studentAssignments];
}
```

Just for software testing, where assignments are directly created in the database

Code segment 22: Model Methods – Configuring assignments

5.4.3 – Sidebar

In most views of students and teachers (except dashboards), there is a sidebar on the right-hand side, which is included in a template. It has only one purpose:



Actual webpage 6 (left)**, Actual webpage 7** (right)**:** Sidebar in students' and teachers' pages

The only purpose is to display the tags and allow users to filter the content on the page. After users have selected a tag, the selected tag will become pink color, and the data will be stored in the tail of the URL for filtering content on the page (e.g., `?tags=tag123,tag456` if the tag 'tag123' and 'tag456' are selected)

Since the code of the controller and model varies on different pages, they will be discussed in the following section. However, in the tag selection on the view, all pages are in common.

This is the code segment that is responsible for this:

```
<script>
    // Get all the tag elements
    var tags = document.querySelectorAll('.sidebar_tag a');


    // Function to get the selected tags from the URL
    function getSelectedTagsFromURL() {
        var urlParams = new URLSearchParams(window.location.search);
        var selectedTags = urlParams.get('tags');
        return selectedTags ? selectedTags.split(',') : [];
    }

    // Function to set the selected tags' state
    function setSelectedTagsState() {
        var selectedTags = getSelectedTagsFromURL();

        // Add selected class to the corresponding tags
        selectedTags.forEach(function(tagName) {
            tags.forEach(function(tag) {
                if (tag.textContent.trim() === tagName) {
                    tag.classList.add('selected');
                }
            });
        });
    }

    // Call the function to set the selected tags' state initially
    setSelectedTagsState();

    // Function to handle tag click event
    function handleTagClick(event) {
        // Toggle the "selected" class on the clicked tag
        this.classList.toggle('selected');

        // Call addTags function to redirect user with the updated URL
        addTags();

        // Prevent the default link behavior
        event.preventDefault();
    }

    function addTags() {
        // Get all the selected tags
        var selectedTags = document.querySelectorAll('.sidebar_tag .selected');

        // Create an array to store the selected tag names
        var tagNames = [];

        // Collect the names of the selected tags
        selectedTags.forEach(function(tag) {
            tagNames.push(tag.textContent.trim());
        });

        // Get the current URL
        var currentUrl = window.location.href;

        // Remove the fragment identifier from the URL
        var urlWithoutFragment = currentUrl.split('#')[0];

        // Create a new URL object
        var url = new URL(urlWithoutFragment);

        // Set the 'tags' parameter in the URL with selected tag names using the pipe delimiter
        url.searchParams.set('tags', tagNames.join(','));

        // Redirect the user to the updated URL
        window.location.href = url.toString();
    }

    // Attach click event listener to each tag
    tags.forEach(function(tag) {
        tag.addEventListener('click', handleTagClick);
    });

</script>
```

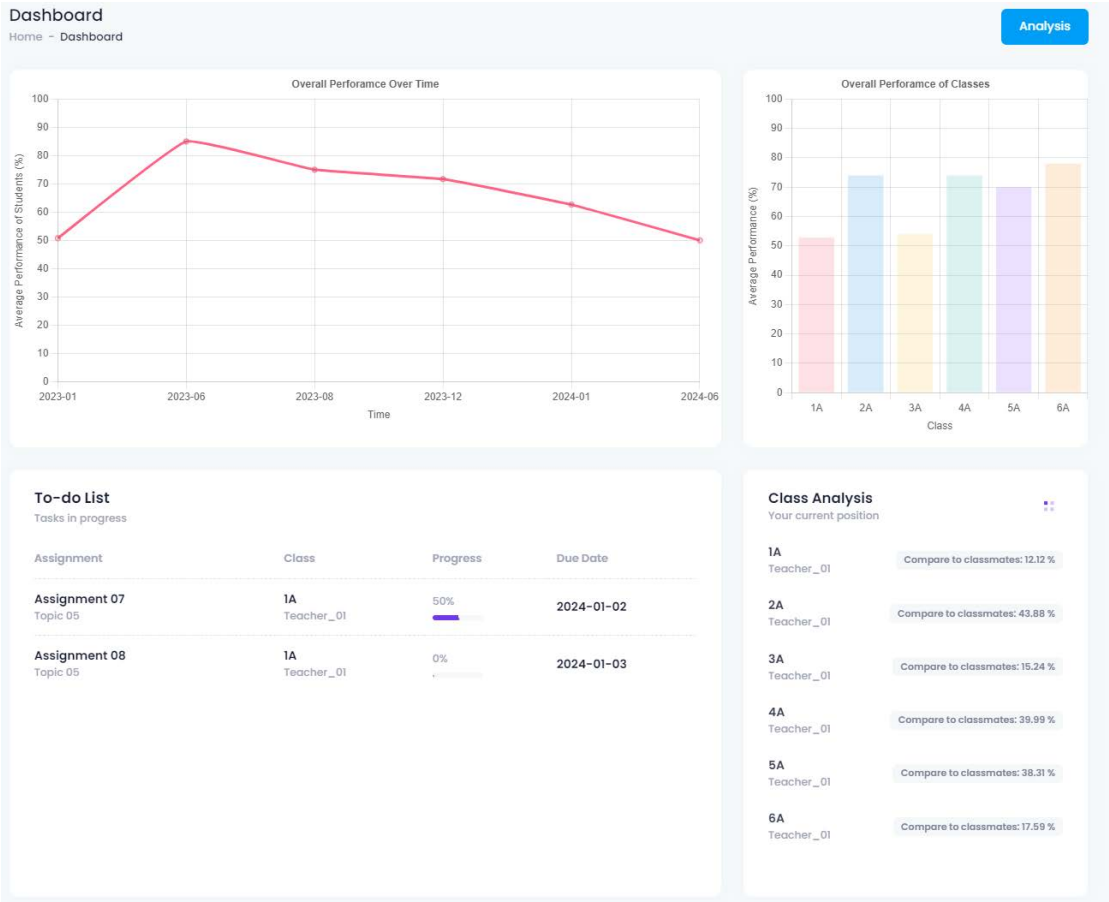
Initialization

When tags are clicked...

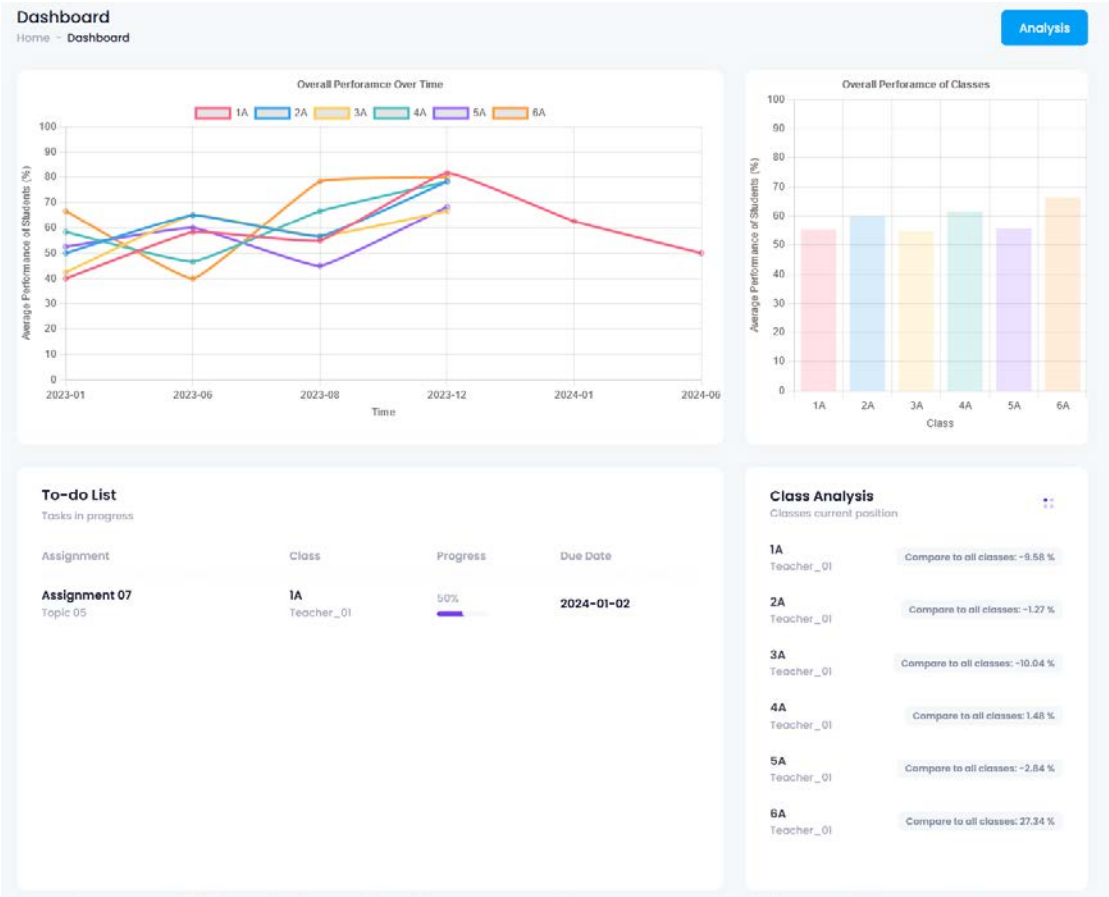
Code segment 23: View – Configuring tag cloud

5.4.4 – Dashboards

In student's dashboards and teacher's dashboards, there are 4 widgets: 2 graphs, 1 to-do list, and 1 comparison widget.



Actual webpage 8: Students' dashboard



Actual webpage 9: Teachers' dashboard

Following are the code segments that are responsible for the 2 graphs:

```php
// load data for the line chart
$dataset_line = [];
$performanceScores = [];
foreach ($loadAssignments as $assignments) {
    foreach ($assignments as $assignment) {
        // if max mark not zero
        if ($assignment['maxMark'] != 0) {
            $performanceScores[] = [
                'performance' => $assignment['totalMark'] / $assignment['maxMark'] * 100,
                'date' => $assignment['dueDate'],
            ];
        }

    }
}
$monthlyPerformance = [];
$monthlyAveragePerformance = [];
// Iterate over the assignments
foreach ($performanceScores as $assignment) {
    $performance = $assignment['performance'];
    $date = date('Y-m', strtotime($assignment['date'])); // Extract the year and month
    // If the month is already present in the array, add the performance score
    if (isset($monthlyPerformance[$date])) {
        $monthlyPerformance[$date][] = $performance;
    } else {
        // If the month is not present, create a new array with the performance score
        $monthlyPerformance[$date] = [$performance];
    }
}
// Calculate the average performance for each month
foreach ($monthlyPerformance as $month => $performanceArray) {
    $averagePerformance = array_sum($performanceArray) / count($performanceArray);
    $monthlyAveragePerformance[$month] = $averagePerformance;
}
// Sort the performance data by month in ascending order
ksort($monthlyAveragePerformance);
// Prepare the data for the line chart
$dataset_line = [];
foreach ($monthlyAveragePerformance as $month => $averagePerformance) {
    $dataset_line[] = ['x' => $month, 'y' => $averagePerformance];
}
// turn dataset into json format
$data['dataset_line'] = json_encode($dataset_line);

// load data for the bar chart
$dataset_bar = [];
$performanceScoreSum = [];
$performanceScoreCount = [];
foreach ($loadAssignments as $classID => $assignments) {
    $performanceScoreSum[$classID] = 0;
    $performanceScoreCount[$classID] = 0;
    foreach ($assignments as $assignment) {
        // if max mark not zero
        if ($assignment['maxMark'] != 0) {
            $performanceScoreSum[$classID] += $assignment['totalMark'] / $assignment['maxMark'] * 100;
            $performanceScoreCount[$classID]++;
        }

    }
}
foreach ($classIDs as $classID) {
    // if $performanceScoreCount[$classID] is zero, then skip
    if ($performanceScoreCount[$classID] == 0) {
        continue;
    }
    $dataset_bar[$classID] = $performanceScoreSum[$classID] / $performanceScoreCount[$classID];
}
// turn dataset into json format
$data['dataset_bar'] = json_encode(array_values($dataset_bar));

// output the array of classIDs in the bar chart
$data['classIDs'] = json_encode(array_keys($dataset_bar));
// output the array of classNames in the bar chart
$model = new \App\Models\ClassModel();
$data['classNames'] = json_encode($model->getClassNames(array_keys($dataset_bar)));
```

Configure associative array $performanceScores, which store each student's assignment in the format: {performance, date}

Configure associative array $monthlyAveragePerformance, which store each student's monthly overall performance in the format: {averagePerformance, month}

Reusing variables defined in the navigation bar

Configure associative array $data['dataset_line'], which can be directly processed by the Chart.js line chart on View

Similarly, configure associative array $data['dataset_bar'], which store student's overall performance in each class in the format {classID, averagePerformance}; it can also be directly processed by the Chart.js bar chart on View

Configure arrays $data['classIDs'] and $data['classNames'] so that relevant information can be displayed on the charts

Code segment 24: Student's Controller Method – Loading the data of the charts

Overall quite similar to previous code segment (for student)

Monthly average performance of students are grouped by class

```php
// load data for the line chart
$dataset_line = [];
$performanceScores = [];
foreach ($studentAssignments as $classID => $topicName) {
    foreach ($topicName as $assignments) {
        foreach ($assignments as $assignment) {
            // if max mark not zero
            if ($assignment['maxMark'] != 0) {
                $performanceScores[$classID][] = [
                    'performance' => $assignment['totalMark'] / $assignment['maxMark'] * 100,
                    'date' => $assignment['dueDate'],
                ];
            }
        }
    }
}
$monthlyPerformance = [];
$monthlyAveragePerformance = [];
// Iterate over the assignments
foreach ($classIDs as $classID) {
    // if $performanceScores do not have the index classID, then skip
    if (!isset($performanceScores[$classID])) {
        continue;
    }
    foreach ($performanceScores[$classID] as $assignment) {
        $performance = $assignment['performance'];
        $date = date('Y-m', strtotime($assignment['date'])); // Extract the year and month
        // If the month is already present in the array, add the performance score
        if (isset($monthlyPerformance[$classID][$date])) {
            $monthlyPerformance[$classID][$date][] = $performance;
        } else {
            // If the month is not present, create a new array with the performance score
            $monthlyPerformance[$classID][$date] = [$performance];
        }
    }
    // Calculate the average performance for each month
    foreach ($monthlyPerformance[$classID] as $month => $performanceArray) {
        $averagePerformance = array_sum($performanceArray) / count($performanceArray);
        $monthlyAveragePerformance[$classID][$month] = $averagePerformance;
    }
    // Sort the performance data by month in ascending order
    ksort($monthlyAveragePerformance[$classID]);
    // Prepare the data for the line chart
    $dataset_line[$classID] = [];
    foreach ($monthlyAveragePerformance[$classID] as $month => $averagePerformance) {
        $dataset_line[$classID][] = ['x' => $month, 'y' => $averagePerformance];
    }
}
// turn dataset into json format
$data['dataset_line'] = json_encode($dataset_line);


// load data for the bar chart
$dataset_bar = [];
$performanceScoreSum = [];
$performanceScoreCount = [];
foreach ($studentAssignments as $classID => $topicName) {
    $performanceScoreSum[$classID] = 0;
    $performanceScoreCount[$classID] = 0;
    foreach ($topicName as $assignments) {
        foreach ($assignments as $assignment) {
            // if max mark not zero
            if ($assignment['maxMark'] != 0) {
                $performanceScoreSum[$classID] += $assignment['totalMark'] / $assignment['maxMark'] * 100;
                $performanceScoreCount[$classID]++;
            }
        }
    }
}
foreach ($classIDs as $classID) {
    // if $performanceScoreCount[$classID] is zero, then skip
    if ($performanceScoreCount[$classID] == 0) {
        continue;
    }
    $dataset_bar[] = $performanceScoreSum[$classID] / $performanceScoreCount[$classID];
}
// turn dataset into json format
$data['dataset_bar'] = json_encode($dataset_bar);


// output the array of classIDs in the charts
$data['classIDs'] = json_encode(array_keys($dataset_line));
// output the array of classNames in the charts
$model = new \App\Models\ClassModel();
$data['classNames'] = json_encode($model->getClassNames(array_keys($dataset_line)));
```

Code segment 25: Teacher's Controller Method – Loading the data of the charts

Chart.js is called through CDN

```
<!--begin::ChartJS Bundle-->
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<!--end::ChartJS Bundle-->
```

Locate the chart on the webpage

```
<!--begin::ChartJS-->
<script>

    const lineChart = document.getElementById('lineChart');

    const data_area = {
        datasets: [{
            label: 'Average Performance of Students (%)',
            data: <?= $dataset_line ?>,
            borderColor: 'rgb(255, 99, 132)',
            fill: false,
            tension: 0.1
        }],
    };

    new Chart(lineChart, {
        type: 'line',
        data: data_area,
        options: {
            scales: {
                x: {
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Time',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Average Performance of Students (%)',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Overall Perforamce Over Time',
                }
            }
        }
    });
```

Retrieve data from back-end

Declare chart type

Load data to chart

```
    const barChart = document.getElementById('barChart');

    const data_bar = {
        labels: <?= $classNames ?>,
        datasets: [{
            label: 'Average Performance (%)',
            data: <?= $dataset_bar ?>,
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
            ],
            borderColor: [
                'rgb(255, 99, 132)',
                'rgb(54, 162, 235)',
                'rgb(255, 206, 86)',
                'rgb(75, 192, 192)',
                'rgb(153, 102, 255)',
                'rgb(255, 159, 64)'
            ],
        }]
    };

    new Chart(barChart, {
        type: 'bar',
        data: data_bar,
        options: {
            scales: {
                x: {
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Class',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Average Performance (%)',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Overall Perforamce of Classes',
                }
            }
        }
    });

</script>
<!--end::ChartJS-->
```

Code segment 26: Student's View – Loading Chart.js charts on dashboard

```
<!--begin::ChartJS-->
<script>

    const lineChart = document.getElementById('lineChart');

    const data_area = {
        datasets: [{
            label: <?= $classNames ?>[0],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[0]],
            borderColor: 'rgb(255, 99, 132)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[1],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[1]],
            borderColor: 'rgb(54, 162, 235)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[2],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[2]],
            borderColor: 'rgb(255, 206, 86)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[3],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[3]],
            borderColor: 'rgb(75, 192, 192)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[4],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[4]],
            borderColor: 'rgb(153, 102, 255)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[5],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[5]],
            borderColor: 'rgb(255, 159, 64)',
            fill: false,
            tension: 0.1
        }]
    };

    new Chart(lineChart, {
        type: 'line',
        data: data_area,
        options: {
            scales: {
                x: {
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Time',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Average Performance of Students (%)',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Overall Perforamce Over Time',
                }
            }
        }
    });
```

Overall quite similar to previous code segment (for student)

Retrieve data of each class, and group the data with different color

```
    const barChart = document.getElementById('barChart');

    const data_bar = {
        labels: <?= $classNames ?>,
        datasets: [{
            label: 'Average Performance of Students (%)',
            data: <?= $dataset_bar ?>,
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
            ],
            borderColor: [
                'rgb(255, 99, 132)',
                'rgb(54, 162, 235)',
                'rgb(255, 206, 86)',
                'rgb(75, 192, 192)',
                'rgb(153, 102, 255)',
                'rgb(255, 159, 64)'
            ],
        }]
    };

    new Chart(barChart, {
        type: 'bar',
        data: data_bar,
        options: {
            scales: {
                x: {
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Class',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Average Performance of Students (%)',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Overall Perforamce of Classes',
                }
            }
        }
    });

</script>
<!--end::ChartJS-->
```

Code segment 27: Teacher's View – Loading Chart.js charts on dashboard

Following are the code segments that are responsible for the comparison widget:

Configure associative array $data['classDetails'] to show relevant information on View

Configure the data shown in the widget

Reusing variable from previous code segment

```
// array of class details, with classID as key
$model = new \App\Models\ClassModel();
$data['classDetails'] = $model->getClassDetails($classIDs);
// attach the teacher name of each class in the class details, to show on the widget
$model = new \App\Models\UserModel();
foreach ($data['classDetails'] as $classID => $classDetail) {
    $data['classDetails'][$classID]['teacherName'] = $model->findUsername([$classDetail['teacherID']])[0];
}

// load data for the bottom right comparison, compare user's performance in each class with other students in the same class
// can use the dataset in the bar chart as the user's performance score
$model = new \App\Models\AssignmentModel();
foreach ($dataset_bar as $classID => $performanceScore) {
    $classMeanPerforamceScore = $model->findClassMeanPerformanceScore($classID);
    $data['comparisonData'][$classID] = round(($performanceScore - $classMeanPerforamceScore) / $classMeanPerforamceScore * 100, 2);
}
```

Code segment 28: Student's Controller Method – Configuring comparison widget

Overall quite similar to previous code segment (for student), except previous code segment cannot be reused

```
// $data['classDetails'] is an array that takes classID as key and others (className and teacherID) as values
$model = new \App\Models\ClassModel();
$data['classDetails'] = $model->getClassDetails($classIDs);
// add teachers' name based on teacherID
$model = new \App\Models\UserModel();
foreach ($data['classDetails'] as $classID => $classDetail) {
    $data['classDetails'][$classID]['teacherName'] = $model->findUsername([$classDetail['teacherID']])[0];
}

// load data for the bottom right comparison, compare overall perforamce of each class with other classes
$model = new \App\Models\AssignmentModel();
foreach ($classIDs as $classID) {
    $classMeanPerformanceScore = $model->findClassMeanPerformanceScore($classID);
    // if $classMeanPerformanceScore is zero, then skip {it was set zero because it was used as denominator in student's dashboard}
    if ($classMeanPerformanceScore == 0) {
        continue;
    }
    $allClassesMeanPerforamceScore = $model->findAllClassesMeanPerformaceScore($classIDs);
    $data['comparisonData'][$classID] = round(($classMeanPerformanceScore - $allClassesMeanPerforamceScore) / $allClassesMeanPerforamceScore * 100, 2);
}
```

Code segment 29: Teacher's Controller Method – Configuring comparison widget

```
// get class details according to classIDs, with classID as key and others as values
public function getClassDetails($classIDs)
{
    $classes = $this->find($classIDs);
    $classDetails = [];
    foreach ($classes as $class) {
        $classDetails[$class['classID']] = [
            'className' => $class['className'],
            'teacherID' => $class['teacherID']
        ];
    }
    return $classDetails;
}
```

Code segment 30: Model Methods – Getting class details

```php
// find average performance of the class
public function findClassMeanPerformanceScore($classID)
{
    // get all assigments in the class
    $assignments = $this->where('classID', $classID)->findAll();
    $totalMark = 0;
    $maxMark = 0;
    foreach ($assignments as $assignment) {
        $totalMark += $assignment['totalMark'];
        $maxMark += $assignment['maxMark'];
    }
    // skip if max mark of the assignment is zero
    if ($maxMark == 0) {
        return 0;
    }
    else {
        // round the output value to 2 significant figures
        return round($totalMark/$maxMark*100, 2);
    }
}

// find average performance of all classes
public function findAllClassesMeanPerformaceScore($classIDs)
{
    $totalMark = 0;
    $maxMark = 0;
    foreach ($classIDs as $classID) {
        // get all assigments in all class
        $assignments = $this->where('classID', $classID)->findAll();
        foreach ($assignments as $assignment) {
            $totalMark += $assignment['totalMark'];
            $maxMark += $assignment['maxMark'];
        }
    }
    // skip if max mark of the assignment is zero
    if ($maxMark == 0) {
        return 0;
    }
    else {
        // round the output value to 2 significant figures
        return round($totalMark/$maxMark*100, 2);
    }
}
```

Code segment 31: Model Methods – Finding overall performance of class/classes

To display the to-do list, 2 segments of code from previous sections are used directly: `$data['toDoList']` from the navigation bar, and `$data[classDetails]` from the comparison widget in the dashboards. Other than iterating through the variables and display the corresponding content directly, the progress of each task is visually displayed with progress bars in Bootstrap 5 using the lines:

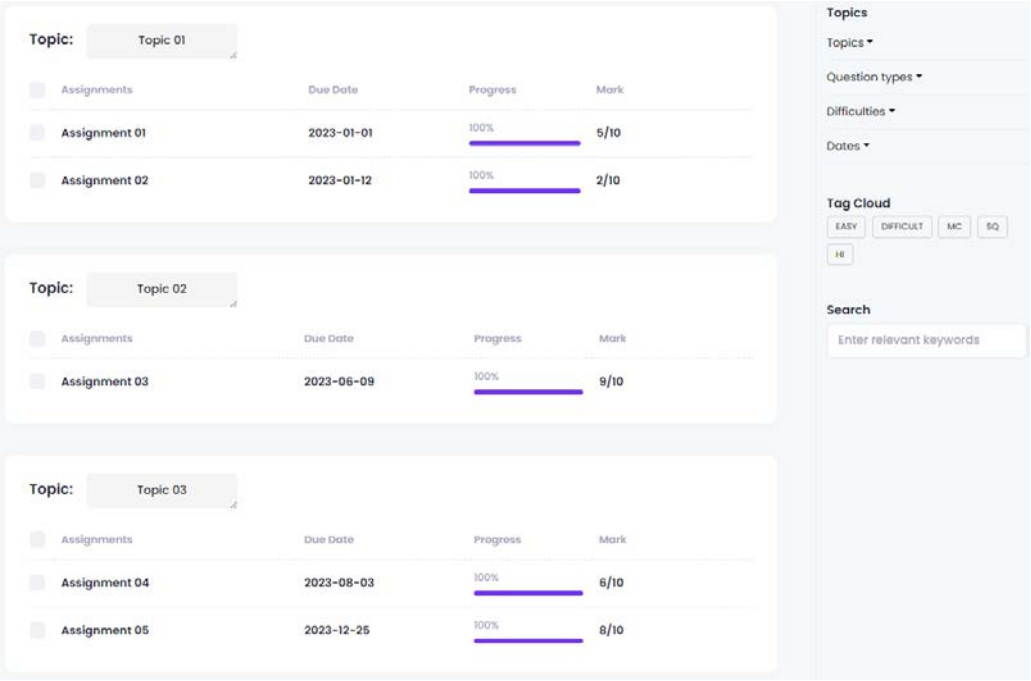
Visually displayed value | Minimum value | Maximum value

```php
<div class="progress h-6px w-50">
    <div class="progress-bar bg-primary" role="progressbar" style="width: <?= $toDoItem['studentProgress'] ?>%" aria-valuenow="<?= $toDoItem['studentProgress'] ?>" aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

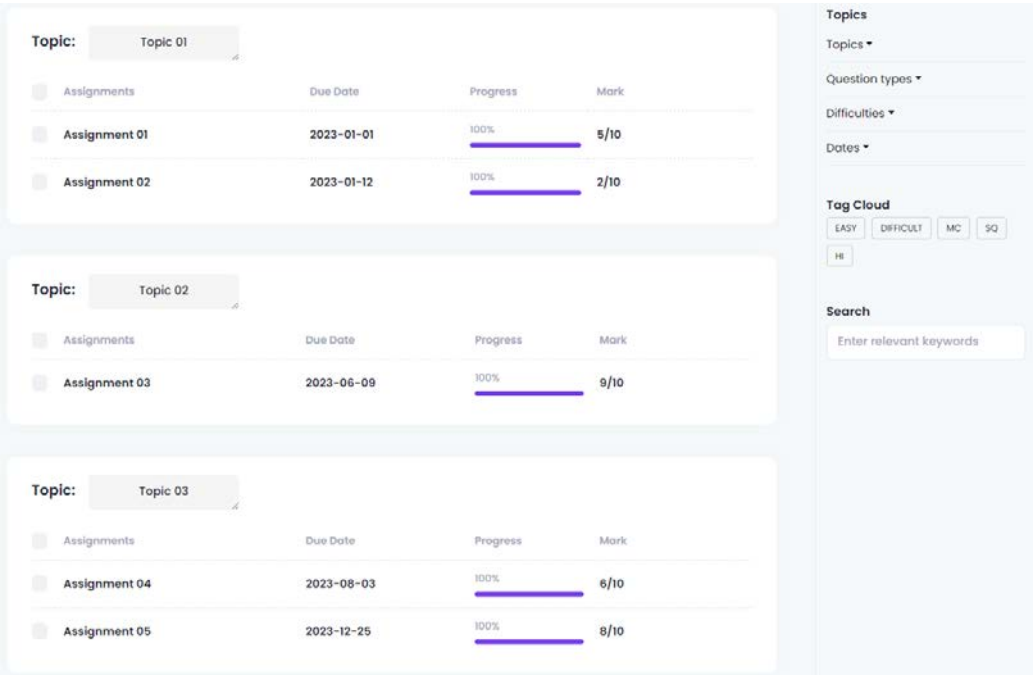
Declare as a progress bar

5.4.5 – Classroom Pages

In the classroom pages, all assignments of a specific class are shown. To access the classroom of a class, the ‘classID’ of the class is required in the path (e.g., '/student/classroom/0' if the ‘classID’ of the class is ‘0’). It will then pass to the controller as `$currentClassID`.



Actual webpage 10: Students’ classroom page



Actual webpage 11: Teachers’ classroom page

In both pages, assignments on the left are grouped by topics, and users can filter assignments by selecting tags in the sidebar.

Following are the code segments that are responsible for this:

As mentioned in Code segment 18, $topics and $loadAssignments, which is defined in the navigation bar, is used to load the assignments in the page

Enable $currentClassID to be accessed on View

Filter assignments by selected tags

Load tags on the sidebar

```
// output the array required to load the assignments
[$data['topics'], $data['loadAssignments']] = [$topics[$currentClassID], $loadAssignments[$currentClassID]];
$data['currentClassID'] = $currentClassID;

// retrieve the tagNames of the selected tags from the url
$selectedTagNames = $this->request->getGet('tags');

// if there are selected tags
if (!empty($selectedTagNames)) {
    $selectedTagNames = explode(',', $selectedTagNames);

    // array of tagIDs of the selected tags
    $model = new \App\Models\TagModel();
    $selectedTagIDs = $model->findTagIDs($selectedTagNames);

    // array of assignmentIDs that have the selected tags
    $assignmentIDs = [];
    $model = new \App\Models\TagModel();
    $assignmentIDs[] = $model->findAssignmentIDs($selectedTagIDs);

    // delete assignments in the $data['loadAssignments'] that do not have the selected tags
    foreach ($data['loadAssignments'] as $key => $assignment) {
        if (!in_array($assignment['assignmentID'], $assignmentIDs[0])) {
            unset($data['loadAssignments'][$key]);
        }
    }
}

// array of all assignmentIDs
$assignmentIDs = [];
foreach ($data['loadAssignments'] as $assignment) {
    $assignmentIDs[] = $assignment['assignmentID'];
}

// output the array required to load the tags in the sidebar
$model = new \App\Models\TagModel();
$data['tagNames'] = $model->sidebarConfiguration($assignmentIDs, $data['userType'], $id);
```

Code segment 32: Student's & Teacher's Controller Method – Loading assignments and sidebar

```php
public function assignmentConfiguration($userID, $userType, $classID)
{
    // array of assignmentIDs for the class
    $assignments = $this->where('classID', $classID)->findAll();

    // array of topicNames (concatenate topic and name) of assignments
    $topicNames = [];
    foreach ($assignments as $assignment) {
        $topicNames[] = ['topic' => $assignment['topic'], 'assignmentName' => $assignment['assignmentName']];
    }

    // remove duplicate in the multi-dimentional array
    $topicNames = array_map("unserialize", array_unique(array_map("serialize", $topicNames)));

    // for student
    if ($userType == 'student')
    {
        $ownedAssignments = [];

        // check if there is any assignment owned by the student for each topicName
        // always select the latest updated one if multiple records are found
        foreach ($topicNames as $topicName) {
            $owned = $this->where('classID', $classID)
                ->where('topic', $topicName['topic'])
                ->where('assignmentName', $topicName['assignmentName'])
                ->where('userID', $userID)->orderBy('updated_at', 'desc')
                ->first();
            if ( $owned != null)
            {
                $ownedAssignments[] = $owned;
            }
            // if not owned, assume the student owned a copy
            // (in future section, it is set that if student clicked into assignment that is not owned by himself/herself,
            //  there will be a copy of the assingment created, which will be owned by the student)
            else {
                $ownedAssignments[] = $this->where('classID', $classID)
                    ->where('topic', $topicName['topic'])
                    ->where('assignmentName', $topicName['assignmentName'])
                    ->where('userID !=', $userID)
                    ->orderBy('updated_at', 'desc')
                    ->first();
            }
        }
        $assignments = $ownedAssignments;

        // this array is not for students, just to define the variable for successful loading
        $studentAssignments = [];
    }
```

Code segment 33: Model Method – Configuring assignments (1st section)

```php
    // for teacher
    elseif ($userType == 'teacher')
    {
        $studentAssignments = [];
        $notOwneds = [];

        // create two arrays that store assignments that are owned and not owned by students
        // (not owned --> not updated by student)
        // aways select the latest updated one if multiple records are found
        foreach ($topicNames as $topicName) {
            $owned = $this->where('classID', $classID)
                ->where('topic', $topicName['topic'])
                ->where('assignmentName', $topicName['assignmentName'])
                ->where('userID !=', $userID)
                ->orderBy('updated_at', 'desc')
                ->findAll();
            if ( $owned != null)
            {
                $studentAssignments[$topicName['topic']."/".$topicName['assignmentName']] = $owned;
            }
            // if assignments are not owned by students, it is owned by teacher, so the userID will be equal to $id
            $notOwneds[] = $this->where('classID', $classID)
                ->where('topic', $topicName['topic'])
                ->where('assignmentName', $topicName['assignmentName'])
                ->where('userID', $userID)
                ->orderBy('updated_at', 'desc')
                ->first();
        }

        // teachers can only see assignments that are not owned by students on the classroom page,
        // teacher has to enter assignment in order to see the assignments owned by students
        $assignments = $notOwneds;

        // array $studentAssignments is for teacher to mark students' works
    }


    // list of topic names for assignments
    $topics = [];
    foreach ($assignments as $assignment) {
        // check if assignment is null
        // (theoretically, in normal use, it should not be null, i.e., when assignment is first created by teacher)
        if ($assignment == null)
        {
            continue;
        }
        $topics[] = $assignment['topic'];
    }

    // the array required to load the assignments
    $loadAssignments = [];
    foreach ($assignments as $assignment) {
        // check if assignment is null (theoretically, in normal use, it should not be null,
        // i.e., when assignment is first created by teacher)
        if ($assignment == null)
        {
            continue;
        }
        $loadAssignments[] = [
            'assignmentID' => $assignment['assignmentID'],
            'assignmentName' => $assignment['assignmentName'],
            'topic' => $assignment['topic'],
            'dueDate' => $assignment['dueDate'],
            'totalMark' => $assignment['totalMark'],
            'maxMark' => $assignment['maxMark'],
            'studentProgress' => $assignment['studentProgress'],

        ];
    }

    return [$topics, $loadAssignments, $studentAssignments];
}
```

Code segment 34: Model Method – Configuring assignments (2nd section)

```php
// configure the tags in the sidebar with the assignmentIDs
public function sidebarConfiguration($assignmentIDs, $userType, $userID)
{
    // if teacher
    if ($userType == 'teacher') {
        // array of tags belong to the assignment index by assignmentID
        $tagss = [];
        foreach ($assignmentIDs as $assignmentID) {
            $tagss[$assignmentID] = $this->where('assignmentID', $assignmentID)
                                        // tags created by students will not be shown
                                        ->where('userID', $userID)
                                        ->orderBy('created_at', 'asc')->findAll();
        }
    }
    else {
        // array of tags belong to the assignment index by assignmentID
        $tagss = [];
        foreach ($assignmentIDs as $assignmentID) {
            $tagss[$assignmentID] = $this->where('assignmentID', $assignmentID)
                                        ->orderBy('created_at', 'asc')->findAll();
        }
    }

    // array of tag names in tagss
    $tagNames = [];
    foreach ($tagss as $tags) {
        // $tags is an array of tags belong to the assignment with assignmentID
        foreach ($tags as $tag) {
            // if there is tag with corresponding assignmentID
            if (!empty($tag['assignmentID'])) {
                $tagNames[] = $tag['tagName'];
            }
        }
    }

    // array of tag names in tagss without duplicates
    $tagNames = array_unique($tagNames);

    // return the array of tag names
    return $tagNames;
}
```

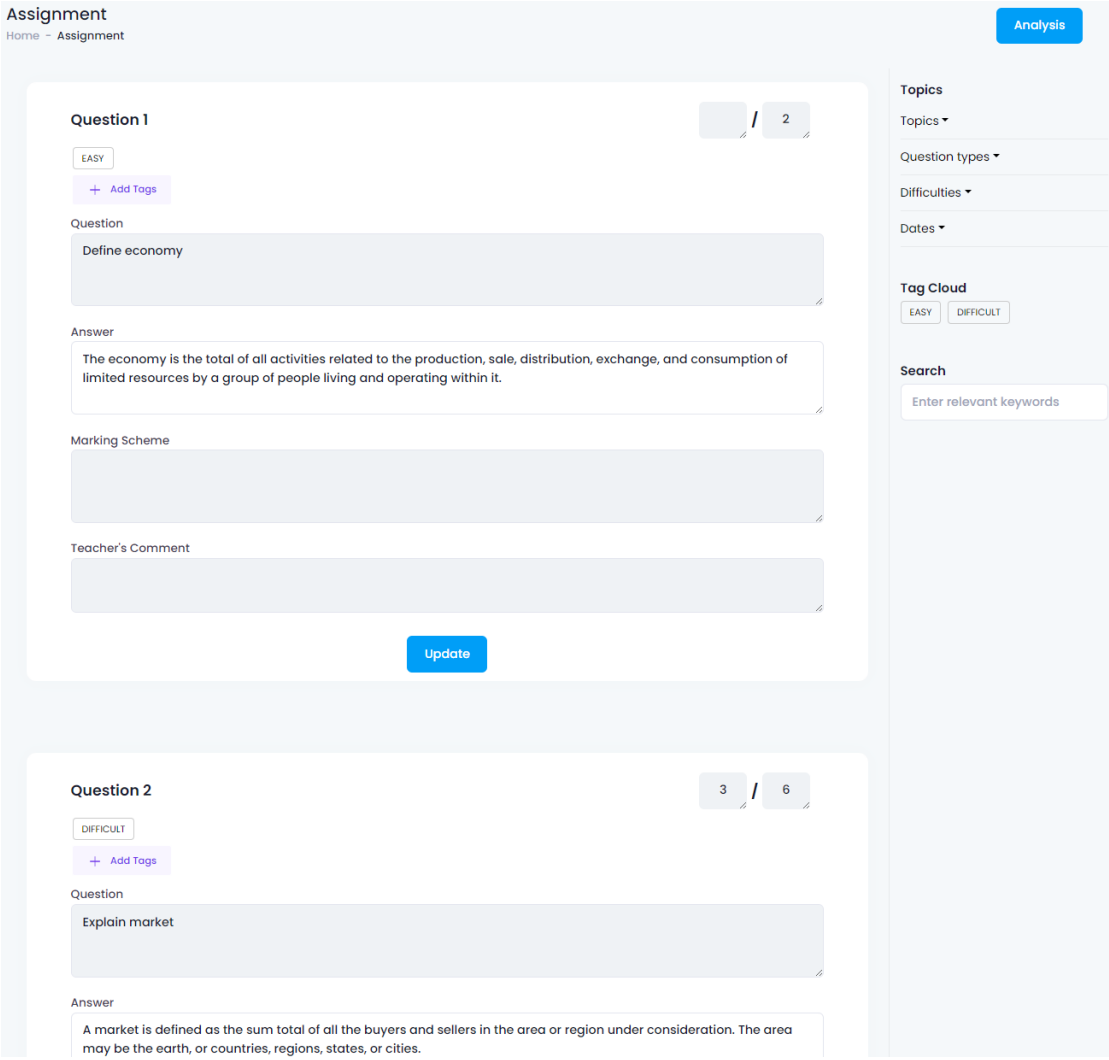
Code segment 35: Model Method – Configuring sidebar

Additionally, on the teacher's page, assignments can be added, deleted, and updated, and the topics of assignments can also be edited. Indeed, the exact same logic in Section 5.3.2 is used again, which involves the use of modals, AJAX requests, and database queries to update the database (i.e., create, update, and delete records).

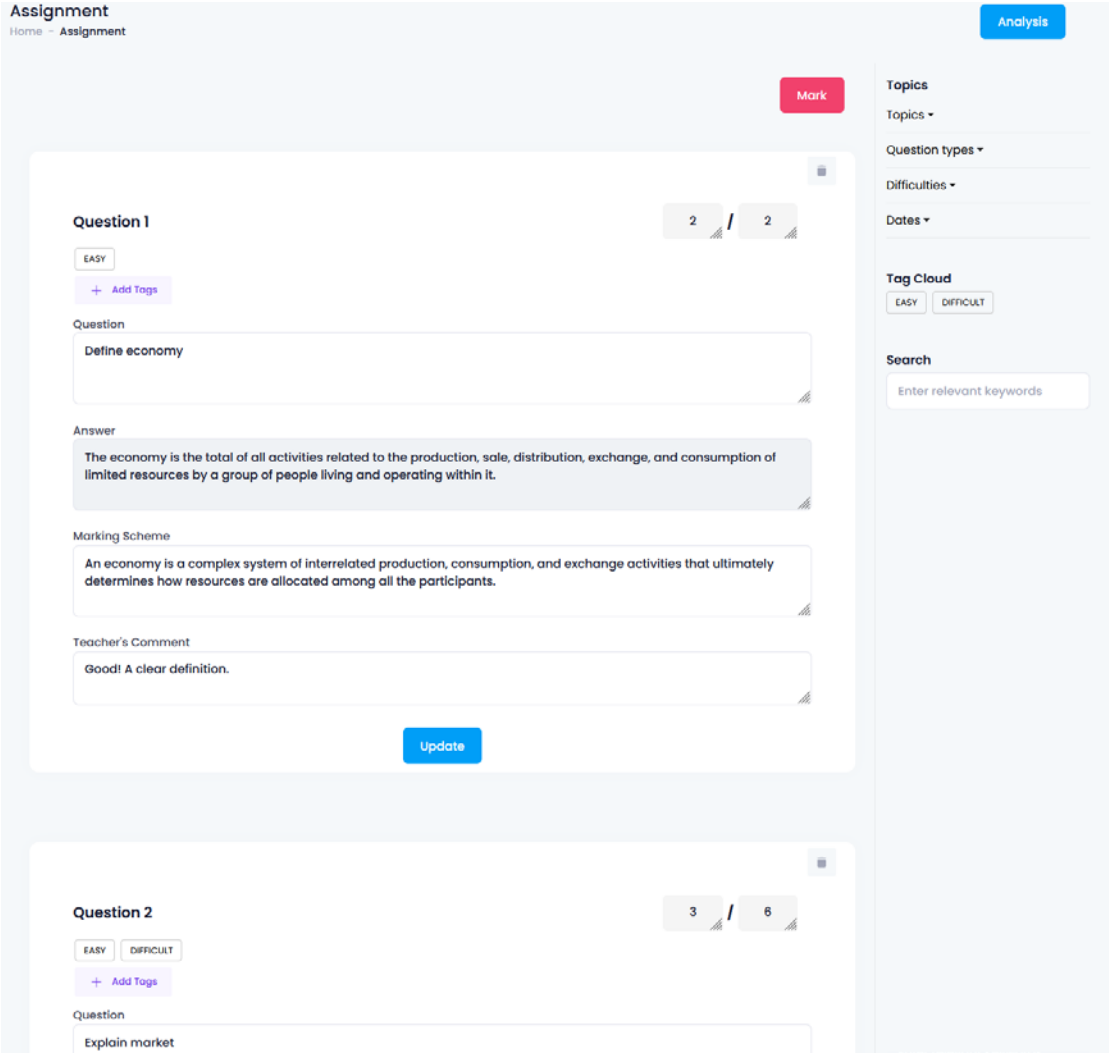
Nonetheless, there is a small difference: since assignments are not loaded by AJAX requests, instead of sending an AJAX request to reload the displayed assignments, the entire web page is reloaded whenever an AJAX request that updates the database is completed.

5.4.6 – Assignment Pages

In the assignment pages, all questions of a specific assignment are shown. Similar to Classroom Pages, to access the questions of an assignment, the ‘assignmentID’ of the assignment is required in the path.



Actual webpage 12: Students’ assignment page



Actual webpage 13: Teachers' assignment page

Similar to 5.4.3, in both pages, questions are shown with corresponding tags, and questions shown on the page can be filtered by the tags in the sidebar.

Load questions

Overall quite similar to how assignments are loaded in Assignment page(Code segment 34)

```
// output the array required to load the questions
$model = new \App\Models\QuestionModel();
$data['loadQuestions'] = $model->questionConfiguration($assignmentID);
$data['assignmentID'] = $assignmentID;
```

Filter questions by selected tags

```
// retrieve the tagNames of the selected tags from the url
$selectedTagNames = $this->request->getGet('tags');

// if there are selected tags
if (!empty($selectedTagNames)) {
    $selectedTagNames = explode(',', $selectedTagNames);

    // array of tagIDs of the selected tags
    $model = new \App\Models\TagModel();
    $selectedTagIDs = $model->findTagIDs($selectedTagNames);

    // array of questionIDs that have the selected tags
    $questionIDs = [];
    $model = new \App\Models\TagModel();
    $questionIDs[] = $model->findQuestionIDs($selectedTagIDs);

    // delete questions in the $data['loadQuestions'] that do not have the selected tags
    foreach ($data['loadQuestions'] as $key => $question) {
        if (!in_array($question['questionID'], $questionIDs[0])) {
            unset($data['loadQuestions'][$key]);
        }
    }
}
```

Load tags

```
// output the array required to load the tags on the questions
$model = new \App\Models\TagModel();
$data['loadTagss'] = $model->tagConfiguration($data['loadQuestions']);


// output the array required to load the tags in the sidebar
$model = new \App\Models\TagModel();
$data['tagNames'] = $model->sidebarConfiguration([$assignmentID], $data['userType'], $id);
```

Code segment 36: Student's & Teacher's Controller Method – Loading Questions and Tags

```php
public function questionConfiguration($assignmentID)
{
    // array of questions belong to the class
    $questions = $this->where('assignmentID', $assignmentID)->orderBy('created_at', 'asc')->findAll();

    // the array required to load the questions
    $loadQuestions = [];
    foreach ($questions as $question) {
        $loadQuestions[] = [
            'questionID' => $question['questionID'],
            'assignmentID' => $question['assignmentID'],
            'totalMark' => $question['totalMark'],
            'maxMark' => $question['maxMark'],
            'question' => $question['question'],
            'answer' => $question['answer'],
            'markingScheme' => $question['markingScheme'],
            'comment' => $question['comment'],
        ];
    }

    return $loadQuestions;
}
```

Code segment 37: Configuring questions

```php
public function tagConfiguration($loadQuestions)
{
    // array of tags belong to the assignment index by questionID
    $tagss = [];
    foreach ($loadQuestions as $loadQuestion) {
        $tagss[$loadQuestion['questionID']] = $this->where('questionID', $loadQuestion['questionID'])
                                                ->orderBy('created_at', 'asc')->findAll();
    }

    // the array required to load the tags
    $loadTagss = [];
    foreach ($tagss as $tags) {
        // $tags is an array of tags belong to the question with questionID
        foreach ($tags as $tag) {
            // if there is tag with corresponding questionID
            if (!empty($tag['questionID'])) {
                // $loadTagss is actually similar to $tagss, but each {tag} is [tagID, tagName]
                $loadTagss[$tag['questionID']][] = [
                    'tagID' => $tag['tagID'],
                    'tagName' => $tag['tagName'],
                ];
            }
        }
    }

    return $loadTagss;
}

// configure the tags in the sidebar with the assignmentIDs
public function sidebarConfiguration($assignmentIDs)
{
    // array of tags belong to the assignment index by assignmentID
    $tagss = [];
    foreach ($assignmentIDs as $assignmentID) {
        $tagss[$assignmentID] = $this->where('assignmentID', $assignmentID)
                                        ->orderBy('created_at', 'asc')->findAll();
    }

    // array of tag names in tagss
    $tagNames = [];
    foreach ($tagss as $tags) {
        // $tags is an array of tags belong to the assignment with assignmentID
        foreach ($tags as $tag) {
            // if there is tag with corresponding assignmentID
            if (!empty($tag['assignmentID'])) {
                $tagNames[] = $tag['tagName'];
            }
        }
    }

    // array of tag names in tagss without duplicates
    $tagNames = array_unique($tagNames);

    // return the array of tag names
    return $tagNames;
}
```

Code segment 38: Model Method – Configuring tags and sidebar

Most of the functionalities on the page (e.g., creating, updating and deleting questions/tags) are basically similar to what has been mentioned in Section 5.4.5, except that the 'questions' and 'tags' tables are being updated instead of the 'assignments' table in the database.

Nevertheless, there are two differences. Firstly, some additional operations are performed when students update questions, in addition to calculating the progresses of the assignment:

```
public function questionUpdate()
{
    // retrieve input values from the request
    $userID = $this->request->getPost('userID');
    $assignmentID = $this->request->getPost('assignmentID');
    $questionID = $this->request->getPost('questionID');
    $question = $this->request->getPost('question');
    $answer = $this->request->getPost('answer');
    $markingScheme = $this->request->getPost('markingScheme');
    $comment = $this->request->getPost('comment');
    $totalMark = $this->request->getPost('totalMark');
    $maxMark = $this->request->getPost('maxMark');

    // check if the assignment is owned by the user -> if not, create a copy of the assignment and update the copy
    $model = new \App\Models\AssignmentModel();
    $newFlag = $model->checkNewAssignment($assignmentID, $userID);
    if ($newFlag) {
        // create a copy of the assignment
        $newAssignmentID = $model->copyAssignment($assignmentID, $userID);

        // copy all questions to the new assignment
        $model = new \App\Models\QuestionModel();
        $newQuestionID = $model->copyQuestion($questionID, $assignmentID, $newAssignmentID);
        $model->copyAllOtherQuestions($assignmentID, $newAssignmentID, $questionID);

        // update the assignmentID
        $assignmentID = $newAssignmentID;

        // update the questionID
        $questionID = $newQuestionID;
    }

    // update the database
    $model = new \App\Models\QuestionModel();
    $model->updateQuestion($questionID, $question, $answer, $markingScheme, $comment, $totalMark, $maxMark);

    // get $studentProgress and $teacherProgress
    $studentProgress = $model->findAssignmentStudentProgress($assignmentID);
    $teacherProgress = $model->findAssignmentTeacherProgress($assignmentID);

    // update the assignment database (update the pregresses of the assignment)
    $model = new \App\Models\AssignmentModel();
    $model->updateAssignmentProgress($assignmentID, $studentProgress, $teacherProgress);

    // redirect to the assignment page using AJAX response
    return $this->response->setJSON(['route' => 'student/assignment/' . $assignmentID]);
}
```

Check if a copy of assignment needs to be made

Calculate the progress of the assignment

Redirect student to their own assignment if an copy is made

Code segment 39: Student's Controller Method – Responding to AJAX request for Question Update

```php
public function checkNewAssignment($assignmentID, $userID)
{
    $assignment = $this->find($assignmentID);
    if ($assignment['userID'] == $userID) {
        return FALSE;
    }
    else {
        return TRUE;
    }
}

public function copyAssignment($assignmentID, $userID)
{
    $assignment = $this->find($assignmentID);

    unset($assignment['assignmentID']); // Remove the ID to insert it as a new assignment
    $assignment['userID'] = $userID; // Modify the userID column with the new value
    $assignment['updated_at'] = date('Y-m-d H:i:s'); // Set the updated_at field to the current time

    // Insert the new assignment
    $this->insert($assignment);

    // Return the new assignment ID
    return $this->getInsertID();
}
```

Code segment 40: Model Method – Checking and copying assignment

```php
public function copyQuestion($sourceQuestionID, $sourceAssignmentID, $targetAssignmentID)
{
    // Create a copy of the Source Question
    $sourceQuestion = $this->find($sourceQuestionID);
    unset($sourceQuestion['questionID']);
    $sourceQuestion['assignmentID'] = $targetAssignmentID;
    $sourceQuestion['updated_at'] = date('Y-m-d H:i:s');

    // Insert the new question
    $this->insert($sourceQuestion);
    $newQuestionID = $this->getInsertID();

    // Return the New Question ID
    return $newQuestionID;
}

public function copyAllOtherQuestions($sourceAssignmentID, $targetAssignmentID, $copiedQuestionID)
{
    // Create a copy of the Source Assignment
    $sourceQuestions = $this->where('assignmentID', $sourceAssignmentID)->findAll();
    // loop through all questions in the source assignment
    foreach ($sourceQuestions as $sourceQuestion) {
        // if the question is not the copied question, copy it
        if ($sourceQuestion['questionID'] != $copiedQuestionID) {
            unset($sourceQuestion['questionID']);
            $sourceQuestion['assignmentID'] = $targetAssignmentID;
            $sourceQuestion['updated_at'] = date('Y-m-d H:i:s');
            $this->insert($sourceQuestion);
        }
    }
}
```

Code segment 41: Model Method – Copying questions

```php
public function findAssignmentTotalMark($assignmentID)
{
    $questions = $this->where('assignmentID', $assignmentID)->findAll();
    $totalMark = 0;
    foreach ($questions as $question) {
        $totalMark += $question['totalMark'];
    }
    return $totalMark;
}

public function findAssignmentMaxMark($assignmentID)
{
    $questions = $this->where('assignmentID', $assignmentID)->findAll();
    $maxMark = 0;
    foreach ($questions as $question) {
        $maxMark += $question['maxMark'];
    }
    return $maxMark;
}

// find the studentProgress of assignment by assignmentID (calculating the percentage of questions answered by students)
public function findAssignmentStudentProgress($assignmentID)
{
    $questions = $this->where('assignmentID', $assignmentID)->findAll();
    $totalQuestionNumber = count($questions);
    $completedQuestionNumber = 0;
    foreach ($questions as $question) {
        // if the anwer is not empty, the question is completed
        if ($question['answer'] != '') {
            $completedQuestionNumber++;
        }
    }
    if ($totalQuestionNumber == 0) {
        return 0;
    } else {
        return round($completedQuestionNumber / $totalQuestionNumber * 100);
    }
}

public function findAssignmentTeacherProgress($assignmentID)
{
    $questions = $this->where('assignmentID', $assignmentID)->findAll();
    $totalQuestionNumber = count($questions);
    $completedQuestionNumber = 0;
    foreach ($questions as $question) {
        if ($question['totalMark'] != '') {
            $completedQuestionNumber++;
        }
    }
    if ($totalQuestionNumber == 0) {
        return 0;
    } else {
        return round($completedQuestionNumber / $totalQuestionNumber * 100);
    }
}
```

Code segment 42: Model Method – Finding assignment marks and progresses

Secondly, teachers can mark students' works by clicking the 'Mark' button, which will then pop up a modal for teachers to redirect himself/herself to different students' works:

```php
// output the array required to load students' assignments (to be marked)
$model = new \App\Models\AssignmentModel();
$data['loadStudentAssignments'] = $model->studentAssignmentConfiguration($assignmentID, $id);

// output the name list of students
$model = new \App\Models\UserModel();
$data['studentNames'] = $model->findUsernames_attachedToIds(array_keys($data['loadStudentAssignments']));
```

To match the order of student names and students' assignments, associated array and array_keys() are used.

Code segment 43: Teacher's Controller Method – Loading students' assignments

```php
public function studentAssignmentConfiguration($assignmentID, $userID)
{
    // find topics and assignment name according to assignmentID
    $assignment = $this->find($assignmentID);
    $topic = $assignment['topic'];
    $assignmentName = $assignment['assignmentName'];

    // find students' assignments with the same topic and assignment name (i.e., $assignments['userID'] != $userID of the teacher)
    $assignments = $this->where('topic', $topic)->where('assignmentName', $assignmentName)->where('userID !=', $userID)->findAll();

    // find the corresponding userIDs of the students
    $userIDs = [];
    foreach ($assignments as $assignment) {
        $userIDs[] = $assignment['userID'];
    }

    // return the final array with userIDs as the key, and the corresponding assignment as the value
    $loadStudentAssignments = [];
    foreach ($assignments as $assignment) {
        $loadStudentAssignments[$assignment['userID']] = [
            'assignmentID' => $assignment['assignmentID'],
            // 'assignmentName' => $assignment['assignmentName'],
            // 'topic' => $assignment['topic'],
            // 'dueDate' => $assignment['dueDate'],
            'totalMark' => $assignment['totalMark'],
            'maxMark' => $assignment['maxMark'],
        ];
    }

    return $loadStudentAssignments;
}
```

Code segment 44: Model Method – Configuring students' assignments

```php
public function findUsernames_attachedToIds($ids)
{
    //find username by id
    $usernames = [];
    foreach ($ids as $id) {
        $usernames[] = $this->find($id)['username'];
    }

    // output ids as the key, usernames as the value
    return array_combine($ids, $usernames);
}
```

Code segment 45: Model Method – Finding usernames by ids (output as an associated array: {ids: usernames})

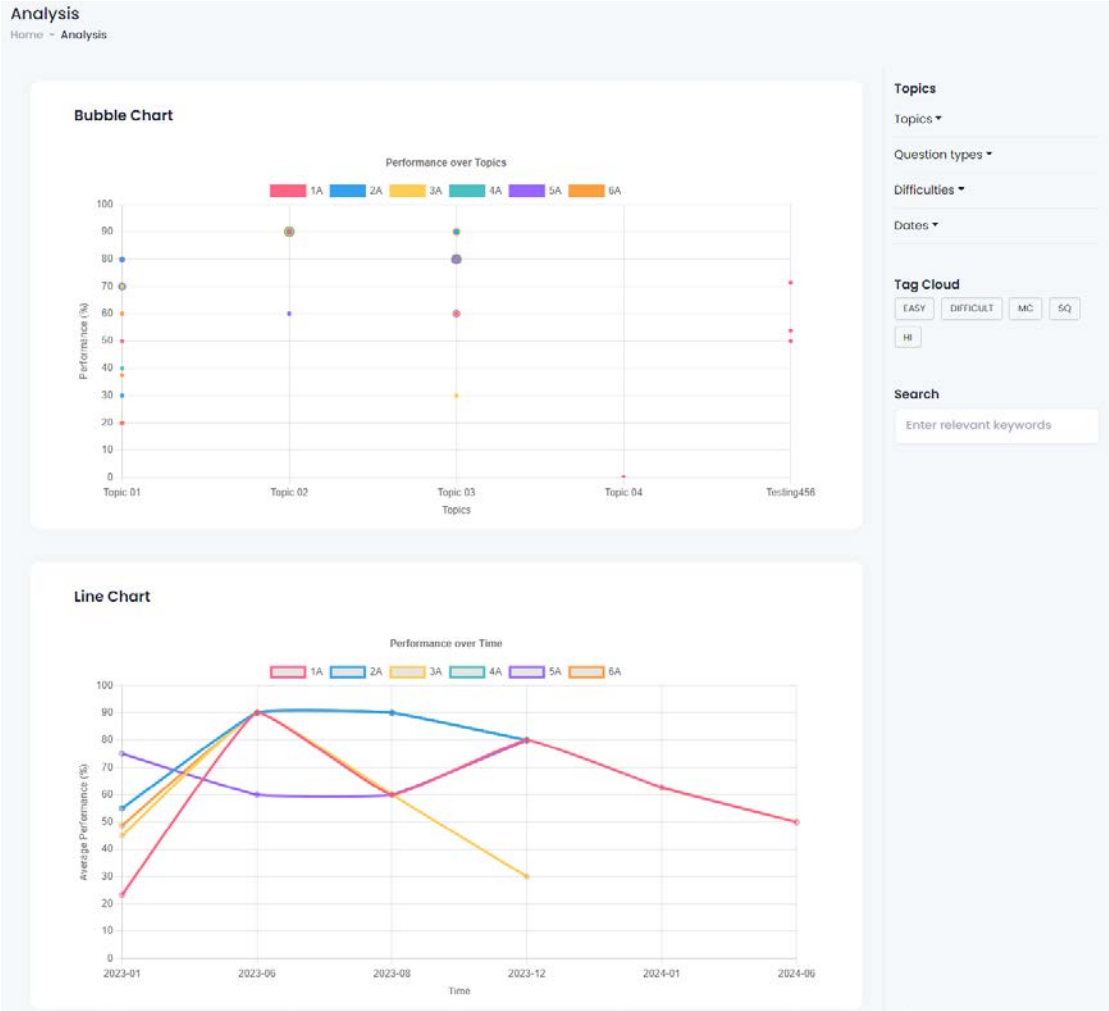
```
<!--begin::Modal - Mark Assignment-->
<!-- a pop up modal to let teacher to select students' work, so that it can be marked the assignment -->
<div class="modal fade" id="markAssignmentModal" tabindex="-1" aria-hidden="true">
    <!--begin::Modal dialog-->
    <div class="modal-dialog modal-dialog-centered">
        <!--begin::Modal content-->
        <div class="modal-content">
            <!--begin::Modal header-->
            <div class="modal-header pb-0 border-0 justify-content-end">
                <!--begin::Close-->
                <div class="btn btn-sm btn-icon btn-active-color-primary" data-bs-dismiss="modal">
                    <!--begin::Svg Icon | path: icons/duotune/arrows/arr061.svg-->
                    <span class="svg-icon svg-icon-1">
                        <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24" fill="none">
                            <rect opacity="0.5" x="6" y="17.3137" width="16" height="2" rx="1" transform="rotate(-45 6 17.3137)" fill="black" />
                            <rect x="7.41422" y="6" width="16" height="2" rx="1" transform="rotate(45 7.41422 6)" fill="black" />
                        </svg>
                    </span>
                    <!--end::Svg Icon-->
                </div>
                <!--end::Close-->
            </div>
            <!--begin::Modal header-->
            <!--begin::markAssignmentForm-->
            <div id="markAssignmentForm">
                <!--begin::Modal body-->
                <div class="modal-body scroll-y mx-5 mx-xl-18 pt-0 pb-15">
                    <!--begin::Heading-->
                    <div class="text-center mb-13">
                        <!--begin::Title-->
                        <h1 class="mb-3">Mark Assignment</h1>
                        <!--end::Title-->
                        <!--begin::Description-->
                        <div class="text-muted fw-bold fs-5">
                            Select Students' work to mark
                        </div>
                        <!--end::Description-->
                    </div>
                    <!--end::Heading-->
                    <br/>
                    <div>
                        <!--begin::Student list (drop-down menu)-->
                        <div class="d-flex justify-content-center">
                            <select class="form-select form-select-solid mb-8" id="studentAssignmentList" name="studentAssignmentList">
                                <option value="0">Select a student</option>
                                <?php foreach ($studentNames as $studentID => $studentName) : ?>
                                    <option value="<?= $loadStudentAssignments[$studentID]['assignmentID'] ?>"><?= $studentName ?></option>
                                <?php endforeach; ?>
                            </select>
                        </div>
                        <!--end::Student list (drop-down menu)-->
                    <br/>
                    <div>
                        <!--begin::Submit Button-->
                        <div class="d-flex justify-content-between">
                            <button type="button" class="btn btn-lg btn-secondary me-3" data-bs-dismiss="modal">Cancel</button>
                            <button id="submitMarkAssigmentForm" type="button" class="btn btn-lg btn-primary">Select</button>
                        </div>
                        <!--end::Submit Button-->
                    </div>
                    <!--end::Modal body-->
                </div>
                <!--end::Modal content-->
            </div>
            <!--end::markAssignmentForm-->
        </div>
    </div>
    <!--end::Modal dialog-->
</div>
<!--end::Modal - Mark Assignment-->
```

Drop-down menu to select students' works

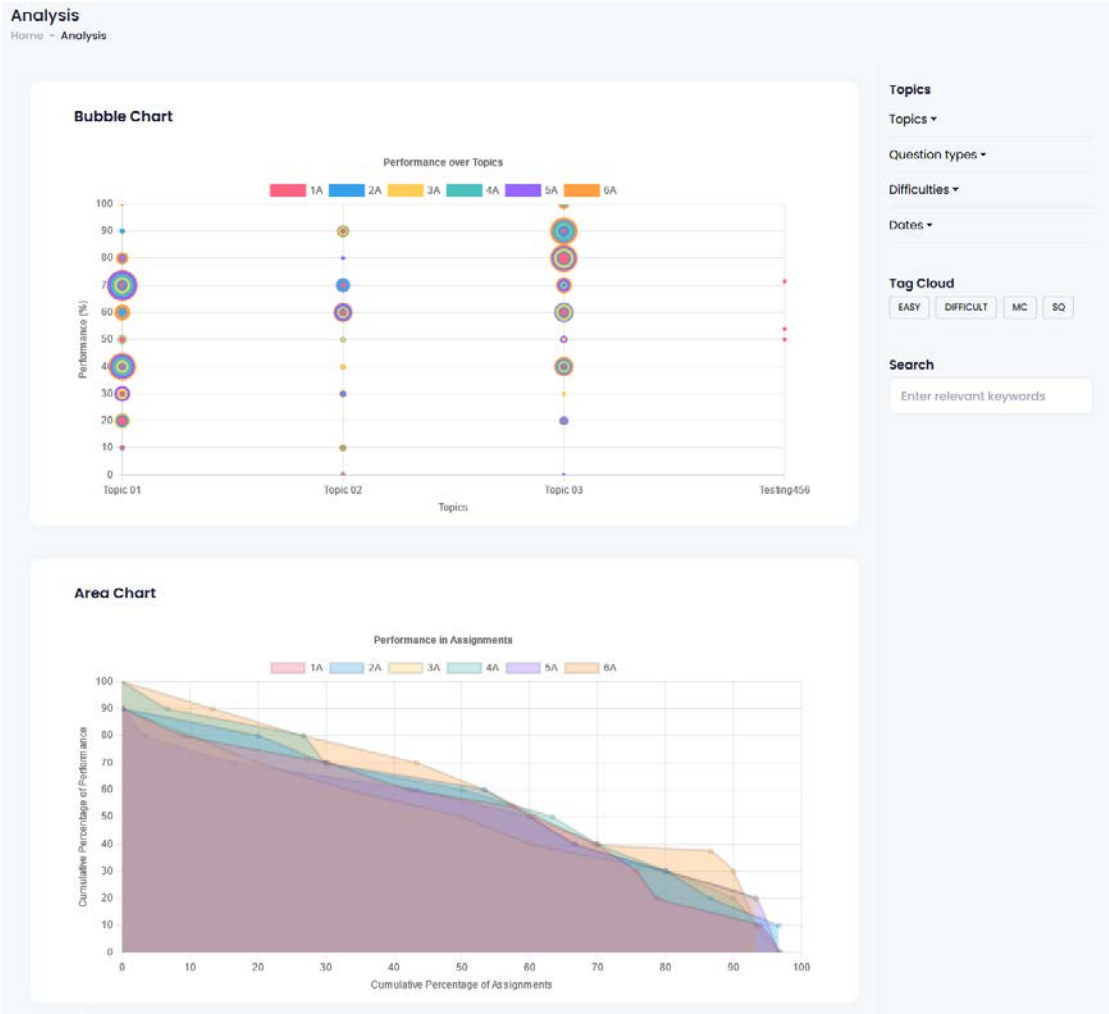
Code segment 46: Teacher's View – Modal for teachers to redirect to students' works

5.4.7 – Analysis Pages

In the analysis page of both students and teachers, there are 2 charts that can analysis their performance visually.



Actual webpage 14: Students' analysis page



Actual webpage 15: Teachers' analysis page

Users can filter the assignments to be analyzed by the tags on the sidebar, which is operated in the same logic as Section 5.4.5.

Following are the code segments that are responsible for loading the charts:

Loading the data for the bubble chart

```php
// load data for the bubble chart
$dataset_bubble = [];
foreach ($loadAssignments as $classID => $assignments) {
    foreach ($assignments as $assignment) {
        // if max mark not zero
        if ($assignment['maxMark'] != 0) {
            $dataset_bubble[$classID][] = [
                'x' => $assignment['topic'],
                'y' => $assignment['totalMark'] / $assignment['maxMark'] * 100,
            ];
        }
    }
}
// add r to the dataset --> number of students with the same x & y values
$counts = [];
foreach ($dataset_bubble as &$dataset) {
    foreach ($dataset as &$item) {
        $key = $item['x'] . '_' . $item['y'];
        $item['r'] = isset($counts[$key]) ? ++$counts[$key] : ($counts[$key] = 1);
    }
}
// turn it into json format
$data['dataset_bubble'] = json_encode($dataset_bubble);
```

Loading the data for the line chart

```php
// load data for the line chart
$dataset_line = [];
$performanceScores = [];
foreach ($loadAssignments as $classID => $assignments) {
    foreach ($assignments as $assignment) {
        // if max mark not zero
        if ($assignment['maxMark'] != 0) {
            $performanceScores[$classID][] = [
                'performance' => $assignment['totalMark'] / $assignment['maxMark'] * 100,
                'date' => $assignment['dueDate'],
            ];
        }
    }
}
$monthlyPerformance = [];
$monthlyAveragePerformance = [];
// Iterate over the assignments
foreach ($classIDs as $classID) {
    // if $performanceScores do not have the index classID, then skip
    if (!isset($performanceScores[$classID])) {
        continue;
    }
    foreach ($performanceScores[$classID] as $assignment) {
        $performance = $assignment['performance'];
        $date = date('Y-m', strtotime($assignment['date'])); // Extract the year and month
        // If the month is already present in the array, add the performance score
        if (isset($monthlyPerformance[$classID][$date])) {
            $monthlyPerformance[$classID][$date][] = $performance;
        } else {
            // If the month is not present, create a new array with the performance score
            $monthlyPerformance[$classID][$date] = [$performance];
        }
    }
    // Calculate the average performance for each month
    foreach ($monthlyPerformance[$classID] as $month => $performanceArray) {
        $averagePerformance = array_sum($performanceArray) / count($performanceArray);
        $monthlyAveragePerformance[$classID][$month] = $averagePerformance;
    }
    // Sort the performance data by month in ascending order
    ksort($monthlyAveragePerformance[$classID]);
    // Prepare the data for the line chart
    $dataset_line[$classID] = [];
    foreach ($monthlyAveragePerformance[$classID] as $month => $averagePerformance) {
        $dataset_line[$classID][] = ['x' => $month, 'y' => $averagePerformance];
    }
}
// turn dataset into json format
$data['dataset_line'] = json_encode($dataset_line);
```

Loading relevant information to the bubble chart

```php
// output the array of classIDs in the bubble chart
$data['classIDs'] = json_encode(array_keys($dataset_bubble));
// output the array of classNames in the bubble chart
$model = new \App\Models\ClassModel();
$data['classNames'] = json_encode($model->getClassNames(array_keys($dataset_bubble)));
```

Code segment 47: Student's Controller – Loading the data of the Charts

Iterate through student's assignments

Overall quite similar to previous code segment (for student)

```php
// load data for the bubble chart
$dataset_bubble = [];
foreach ($studentAssignments as $classID => $classAssignments) {
    foreach ($classAssignments as $assignments) {
        foreach ($assignments as $assignment) {
            if ($assignment['classID'] == $classID) {
                // if max mark not zero
                if ($assignment['maxMark'] != 0) {
                    $dataset_bubble[$classID][] = [
                        'x' => $assignment['topic'],
                        'y' => $assignment['totalMark'] / $assignment['maxMark'] * 100,
                    ];
                }
            }
        }
    }
}
// add r to the dataset --> number of students with the same x & y values
$counts = [];
foreach ($dataset_bubble as &$dataset) {
    foreach ($dataset as &$item) {
        $key = $item['x'] . '_' . $item['y'];
        $item['r'] = isset($counts[$key]) ? ++$counts[$key] : ($counts[$key] = 1);
    }
}
// turn it into json format
$data['dataset_bubble'] = json_encode($dataset_bubble);

// load data for the area chart
$dataset_area = [];
// array performance scores of each assignment per class
$performanceScores = [];
foreach ($studentAssignments as $classID => $classAssignments) {
    foreach ($classAssignments as $assignments) {
        foreach ($assignments as $assignment) {
            if ($assignment['classID'] == $classID) {
                // if max mark not zero
                if ($assignment['maxMark'] != 0) {
                    $performanceScores[$classID][] = strval($assignment['totalMark'] / $assignment['maxMark'] * 100);
                }
            }
        }

    }
    if (!empty($performanceScores[$classID])) {
        // sort the performance scores in ascending order
        sort($performanceScores[$classID]);
    }
}
$assignments = [];
$performance = [];
$cumulativePercentage = [];
foreach ($classIDs as $classID) {
    // if $performanceScores[$classID] is empty, then skip
    if (empty($performanceScores[$classID])) {
        continue;
    }
    // Count the frequency of each performance score
    $scoreFrequencies = array_count_values($performanceScores[$classID]);
    // Iterate over the score frequencies to populate $assignments and $performance arrays
    foreach ($scoreFrequencies as $score => $frequency) {
        $assignments[$classID][] = $frequency; // Set the number of assignments based on the frequency
        $performance[$classID][] = $score; // Add the performance score to the $performance array
    }
    // Calculate the total count of assignments
    $totalCount = array_sum($assignments[$classID]);
    // Calculate cumulative count and cumulative percentage
    $runningCount = 0;
    foreach ($assignments[$classID] as $index => $count) {
        $runningCount += $count;
        $cumulativePercentage[$classID] = 100 - (($runningCount / $totalCount) * 100);
        $dataset_area[$classID][] = array('x' => $cumulativePercentage[$classID], 'y' => $performance[$classID][$index]);
    }
}
// turn dataset into json format
$data['dataset_area'] = json_encode($dataset_area);

// output the array of classIDs in the charts
$data['classIDs'] = json_encode(array_keys($dataset_bubble));
// output the array of classNames in the charts
$model = new \App\Models\ClassModel();
$data['classNames'] = json_encode($model->getClassNames(array_keys($dataset_bubble)));
```

Code segment 48: Teacher's Controller – Loading the data of the Charts

Overall quite similar to code segment 27 in Section 5.4.4

Retrieve data of each class, and group the data with different color

Displaying bubble chart

```
<!--begin::ChartJS-->
<script>
    const bubbleChart = document.getElementById('bubbleChart');

    const data_bubble = {
        datasets: [{
            label: <?= $classNames ?>[0],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[0]],
            backgroundColor: 'rgb(255, 99, 132)'
        }
        ,{
            label:  <?= $classNames ?>[1],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[1]],
            backgroundColor: 'rgb(54, 162, 235)'
        }
        ,{
            label:  <?= $classNames ?>[2],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[2]],
            backgroundColor: 'rgb(255, 206, 86)'
        }
        ,{
            label:  <?= $classNames ?>[3],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[3]],
            backgroundColor: 'rgb(75, 192, 192)'
        }
        ,{
            label:  <?= $classNames ?>[4],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[4]],
            backgroundColor: 'rgb(153, 102, 255)'
        }
        ,{
            label:  <?= $classNames ?>[5],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[5]],
            backgroundColor: 'rgb(255, 159, 64)'
        }]
    };

    new Chart(bubbleChart, {
        type: 'bubble',
        data: data_bubble,
        options: {
            scales: {
                x: {
                    type: 'category',
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Topics',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Performance (%)',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Performance over Topics',
                }
            }
        }
    });
```

```
    const lineChart = document.getElementById('lineChart');

    const data_area = {
        datasets: [{
            label: <?= $classNames ?>[0],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[0]],
            borderColor: 'rgb(255, 99, 132)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[1],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[1]],
            borderColor: 'rgb(54, 162, 235)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[2],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[2]],
            borderColor: 'rgb(255, 206, 86)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[3],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[3]],
            borderColor: 'rgb(75, 192, 192)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[4],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[4]],
            borderColor: 'rgb(153, 102, 255)',
            fill: false,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[5],
            data: <?= $dataset_line ?>[<?= $classIDs ?>[5]],
            borderColor: 'rgb(255, 159, 64)',
            fill: false,
            tension: 0.1
        }]
    };

    new Chart(lineChart, {
        type: 'line',
        data: data_area,
        options: {
            scales: {
                x: {
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Time',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Average Performance (%)',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Performance over Time',
                }
            }
        }
    });
</script>

<!--end::ChartJS-->
```

Code segment 49: Student's View – Loading Chart.js charts on analysis page

Overall quite similar to previous code segment (for student)

```
<!--begin::ChartJS-->
<script>
    const bubbleChart = document.getElementById('bubbleChart');

    const data_bubble = {
        datasets: [{
            label: <?= $classNames ?>[0],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[0]],
            backgroundColor: 'rgb(255, 99, 132)'
        }
        ,{
            label:  <?= $classNames ?>[1],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[1]],
            backgroundColor: 'rgb(54, 162, 235)'
        }
        ,{
            label:  <?= $classNames ?>[2],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[2]],
            backgroundColor: 'rgb(255, 206, 86)'
        }
        ,{
            label:  <?= $classNames ?>[3],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[3]],
            backgroundColor: 'rgb(75, 192, 192)'
        }
        ,{
            label:  <?= $classNames ?>[4],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[4]],
            backgroundColor: 'rgb(153, 102, 255)'
        }
        ,{
            label:  <?= $classNames ?>[5],
            data: <?= $dataset_bubble ?>[<?= $classIDs ?>[5]],
            backgroundColor: 'rgb(255, 159, 64)'
        }]
    };

    new Chart(bubbleChart, {
        type: 'bubble',
        data: data_bubble,
        options: {
            scales: {
                x: {
                    type: 'category',
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Topics',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Performance (%)',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Performance over Topics',
                }
            }
        }
    });
```

```
    const areaChart = document.getElementById('areaChart');

    const data_area = {
        datasets: [{
            label: <?= $classNames ?>[0],
            data: <?= $dataset_area ?>[<?= $classIDs ?>[0]],
            backgroundColor: 'rgb(255, 99, 132, 0.3)',
            fill: true,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[1],
            data: <?= $dataset_area ?>[<?= $classIDs ?>[1]],
            backgroundColor: 'rgb(54, 162, 235, 0.3)',
            fill: true,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[2],
            data: <?= $dataset_area ?>[<?= $classIDs ?>[2]],
            backgroundColor: 'rgb(255, 206, 86, 0.3)',
            fill: true,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[3],
            data: <?= $dataset_area ?>[<?= $classIDs ?>[3]],
            backgroundColor: 'rgb(75, 192, 192, 0.3)',
            fill: true,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[4],
            data: <?= $dataset_area ?>[<?= $classIDs ?>[4]],
            backgroundColor: 'rgb(153, 102, 255, 0.3)',
            fill: true,
            tension: 0.1
        }
        ,{
            label:  <?= $classNames ?>[5],
            data: <?= $dataset_area ?>[<?= $classIDs ?>[5]],
            backgroundColor: 'rgb(255, 159, 64, 0.3)',
            fill: true,
            tension: 0.1
        }]
    };

    new Chart(areaChart, {
        type: 'line',
        data: data_area,
        options: {
            scales: {
                x: {
                    type: 'linear',
                    position: 'bottom',
                    min: 0,
                    max: 100,
                    reverse: true,
                    title: {
                        display: true,
                        text: 'Cumulative Percentage of Assignments',
                    }
                },
                y: {
                    min: 0,
                    max: 100,
                    title: {
                        display: true,
                        text: 'Cumulative Percentage of Performance',
                    }
                },
            },
            plugins: {
                title: {
                    display: true,
                    text: 'Performance in Assignments',
                }
            }
        }
    });
</script>

<!--end::ChartJS-->
```

Area chart is derived from line chart, by adding 'fill: ture' in each dataset

Code segment 50: Teacher's View – Loading Chart.js charts on analysis page