Database Create Table Scripts

```
CREATE  TABLE  assignment  (

        assignmentID  int(11)  NOT NULL  AUTO_INCREMENT,
        title  varchar(100)  NOT NULL,
        details  longtext  NOT NULLL,
        due_date  datetime  NOT NULL,
        userID  varchar(9)  NOT NULL,
        fileID  int(11),

        PRIMARY KEY (assignmentID),

        FOREIGN KEY (userID) REFERENCES user(userID),
        FOREIGN KEY (fileID) REFERENCES assignment_file(file_id)

);

CREATE  TABLE  assignment_file  (

        file_id  int(11)  NOT NULL,
        file_url  varchar(100)  NOT NULL,
        file_name  varchar(100)  NOT NULL,

        PRIMARY KEY (file_id)

);

CREATE  TABLE  avatar  (

        avatarID  int(11)  NOT NULL,
        avatarURL  varchar(100)  NOT NULL,

        PRIMARY KEY (avatarID)

);

CREATE  TABLE  comment  (

        commentID  int(11)  NOT NULL  AUTO_INCREMENT,
        time_created  datetime  NOT NULL,
        content  longtext  NOT NULL,
        postedBy  varchar(9)  NOT NULL,
        threadID  int(11)  NOT NULL,

        PRIMARY KEY (commentID),

        FOREIGN KEY (postedBy) REFERENCES user(userID),
        FOREIGN KEY (threadID) REFERENCES thread(threadID)

);
```

```
CREATE  TABLE  group  (

        groupID  varchar(50)  NOT NULL,
        teacherID  varchar(9),

        PRIMARY KEY (groupID),

        FOREIGN KEY (teacherID) REFERENCES user(userID)



);

CREATE  TABLE  group_has_assignment  (

        groupID  varchar(50)  NOT NULL,
        assignmentID  int(11)  NOT NULL,

        PRIMARY KEY (groupID, assignmentID),

        FOREIGN KEY (groupID) REFERENCES group(groupID),
        FOREIGN KEY (assignmentID) REFERENCES assignment(assignmentID)

);

CREATE  TABLE  level  (

        levelID  int(11)  NOT NULL,
        levelStage  varchar(45)  NOT NULL,
        levelName  varchar(45)  NOT NULL,
        levelDescription  longtext  NOT NULL,
        worldID  varchar(10)  NOT NULL,

        PRIMARY KEY (levelID, worldID),

        FOREIGN KEY (worldID) REFERENCES world(worldID)

);

CREATE  TABLE  powerup  (

        powerID  int(11)  NOT NULL  AUTO_INCREMENT,
        cost  int(11)  NOT NULL,
        description  varchar(100)  NOT NULL,
        name  varchar(45)  NOT NULL,

        PRIMARY KEY (powerID)

);
```

```
CREATE  TABLE  question  (

        worldID  varchar(10)  NOT NULL,
        levelID  int(11)  NOT NULL,
        questionID  int(11)  NOT NULL  AUTO_INCREMENT,
        questionTitle  longtext  NOT NULL,
        option1  longtext  NOT NULL,
        option2  longtext  NOT NULL,
        option3  longtext  NOT NULL,
        option4 longtext  NOT NULL,
        difficulty  int(11)  NOT NULL,
        correctOption  int(11)  NOT NULL,

        PRIMARY KEY (worldID, levelID, questionID),

        FOREIGN KEY (worldID) REFERENCES world(worldID),
        FOREIGN KEY (levelID) REFERENCES level(levelID)

);

CREATE  TABLE  thread  (

        threadID  int(11)  NOT NULL  AUTO_INCREMENT,
        time_created  datetime  NOT NULL,
        title  varchar(100)  NOT NULL,
        details  longtext  NOT NULL,
        userID  varchar(9)  NOT NULL,

        PRIMARY KEY (threadID),

        FOREIGN KEY (userID) REFERENCES user(userID)

);

CREATE  TABLE  UCL  (

        uclID  int(11)  NOT NULL,
        userId  varchar(45)  NOT NULL,
        questionId  varchar(45)  NOT NULL,
        uclName  varchar(45)  NOT NULL,
        uclDesc  varchar(100)  NOT NULL,
        questionTitle  varchar(45)  NOT NULL,
        option1  varchar(45)  NOT NULL,
        option2  varchar(45)  NOT NULL,
        option3  varchar(45)  NOT NULL,
        option4  varchar(45)  NOT NULL,
        correctOption  int(11)  NOT NULL,

        PRIMARY KEY (uclID, userId, questionId)

);
```

```
CREATE  TABLE  user  (

        userID  varchar(9)  NOT NULL,
        name  varchar(50)  NOT NULL,
        email  varchar(100)  NOT NULL,
        password  varchar(50)  NOT NULL,
        role  varchar(50)  NOT NULL,
        avatarID  int(11)  NOT NULL  DEFAULT 1,
        userGroup  varchar(10),
        currency  int(11),

        PRIMARY KEY (userID)

);

CREATE  TABLE  user_has_powerup  (

        userID  varchar(9)  NOT NULL,
        powerID  int(11)  NOT NULL,
        quantity  int(11),

        PRIMARY KEY (userID, powerID),

        FOREIGN KEY (userID) REFERENCES user(userID),
        FOREIGN KEY (powerID) REFERENCES powerup(powerID)

);

CREATE  TABLE  user_unlock_level  (

        userID  varchar(9)  NOT NULL,
        levelID  int(11)  NOT NULL,
        worldID  varchar(10)  NOT NULL,
        unlock  int(11)  NOT NULL  DEFAULT 0,
        score  int(11)  NOT NULL  DEFAULT 0,

        PRIMARY KEY (userID, levelID, worldID),

        FOREIGN KEY (userID) REFERENCES user(userID),
        FOREIGN KEY (levelID) REFERENCES level(levelID),
        FOREIGN KEY (worldID) REFERENCES world(worldID)

);
```

```
CREATE  TABLE  user_unlock_world  (

        userID  varchar(9)  NOT NULL,
        worldID  varchar(10)  NOT NULL,
        unlock  int(11)  NOT NULL  DEFAULT 0,
        score  int(11)  NOT NULL  DEFAULT 0,

        PRIMARY KEY (userID, worldID),

        FOREIGN KEY (userID) REFERENCES user(userID),
        FOREIGN KEY (worldID) REFERENCES world(worldID)

);

CREATE  TABLE  world  (

        worldID  varchar(10)  NOT NULL,
        worldName  varchar(45)  NOT NULL,

        PRIMARY KEY (worldID)

);
```

Database Query Scripts

Account Manager

1. Verify users by their userID and password to handle user login

```
public function verifyUser($id,$password){
	if(Users::where('userID', strtoupper($id))->where('password', $password)->exists()) {
		$user = Users::where('userID', strtoupper($id))->where('password',
			$password)>leftJoin('avatar','user.avatarID','=','avatar.avatarID')->get();
		return response()->json($user[0], 200);
	} else {
		return response()->json([
			"message" => "User not found",
			"status" => 404
		], 404);
	}
}
```

2. Get all the information of the user from the database once the user is logged in

```
public function getUser($id){
	if(Users::where('userID', $id)->exists()) {
		$user = Users::where('userID', $id)->get();
		return response()->json($user[0], 200);
	} else {
		return response()->json([
			"message" => "User not found",
			"error" => 404
		], 404);
	}
}
```

3. Update the password of the user when the user changes his/her password

```
public function updateUserPassword(Request $request, $id){
	if (Users::where('userID', $id)->exists()) {
		$user = Users::find($id);
		$user->password = is_null($request->password) ? $user->password : $request-
			>password;
		$user->timestamps = false;
		$user->save();
		return response()->json([
			"message" => "User's password Updated"
		], 200);
	} else {
		return response()->json([
			"message" => "User not found"
		], 404);
	}
}
```

4. Send an email to the user when the user forgets his/her password

```
public function resetPasswordLink($id){
        if (Users::where('userID', $id)->exists()) {
                $user = Users::find($id);
                $url = "https://learnez.a2hosted.com/public/api/user/resetpassword/link/reset/".$id;
                $data = array( 'email' => $user->email, 'url' => $url);
                Mail::send([],$data, function ($message) use ($data) {
                        $message->to($data['email'])
                                ->subject('Reset Password Request')
                                ->setBody('Please Click on this link to reset your password: '.
                                        $data['url']);
                });
                return response()->json([
                        "message" => $user->email
                ], 200);
        } else {
                return response()->json([
                        "message" => "User xneot found"
                ], 404);

        }
}
```

5. Update the password of the user when the user clicks on the password reset link sent to his/her email

```
public function resetPassword($id){
        if (Users::where('userID', $id)->exists()) {
                $user = Users::find($id);
                $characters =
                '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
                $charactersLength = strlen($characters);
                $randomString = '';
                for ($i = 0; $i < 5; $i++) {
                        $randomString .= $characters[rand(0, $charactersLength - 1)];
                }
                $user->password = $randomString;
                $user->timestamps = false;
                $user->save();
                $data = array( 'email' => $user->email, 'password' => $randomString);
                Mail::send([],$data, function ($message) use ($data) {
                        $message->to($data['email'])
                                ->subject('Password has been reset')
                                ->setBody('Your New Password  is: '.$data['password']);
                        });
                return response()->json([
                        "message" => "A New Password Has been Sent to your Email"
                ], 200);
        } else {
                return response()->json([
                        "message" => "User not found"
                ], 404);
        }
}
```

6. Update the avatar of the user

```
public function updateUserAvatar(Request $request, $id){
        if (Users::where('userID', $id)->exists()) {
                $user = Users::find($id);
                $user->avatarID = is_null($request->avatarID) ? $user->avatarID : $request->avatarID;
                $user->timestamps = false;
                $user->save();
                return response()->json([
                        "message" => "User's Avatar Updated"
                ], 200);
        } else {
                return response()->json([
                        "message" => "User not found"
                ], 404);
        }
}
```

7. Update the currency of the user when the user completes a level or purchase powerups

```
public function updateUserCurrency(Request $request,$id){
        if (Users::where('userID', $id)->exists()) {
                $user = Users::find($id);
                $user->currency = $request->currency;
                $user->timestamps = false;
                $user->save();
                return response()->json([
                        "message" => "User's Currency Updated"
                ], 200);
        } else {
                return response()->json([
                        "message" => "User not found"
                ], 404);
        }
}
```

Performance Analytics

8. Get all the groups that the teacher is in charge of

```
public function getTeacherGroups($id){
        if(Group::where('teacherID', $id)->exists()) {
                $groups = Group::select('groupID')->where('teacherID', $id)->get();
                return response()->json($groups, 200);
        } else {
                return response()->json([
                        "message" => "Group not found"
                ], 404);
        }
}
```

9. Get all the students that are in a selected group

```
public function getStudentsInGroup($groupID){
        $students = Users::where('userGroup',$groupID)->get();
        return response()->json($students, 200);
}
```

10. Get the score of all the students that are in a selected group

```
public function getStudentsScore($groupID){
        $score = Users::where('userGroup',$groupID)->join('user_unlock_level as ul','ul.userID', '=',
                'user.userID')->groupBy('worldID','levelID')->selectRaw('sum(score) as
                score,worldID,levelID')->get();
        return response()->json($score, 200);
}
```

11. Generate performance report for a selected group

```
public function generateReport($groupID){
	$reportFront = strval($groupID);
	$reportEnd = "-Performance-Report.csv";
	$reportName = $reportFront.$reportEnd;
	$headers = array(
		"Content-type" => "text/csv",
		"Content-Disposition" => "attachment; filename=".$reportName,
		"Pragma" => "no-cache",
		"Cache-Control" => "must-revalidate, post-check=0, pre-check=0",
		"Expires" => "0"
	);
	$score = Users::where('userGroup',$groupID)->join('user_unlock_level as ul','ul.userID', '=',
		'user.userID')->join("world as w","ul.worldID","=","w.worldID")->join("level as
		l","ul.levelID","=","l.levelID")->groupBy('w.worldName','l.levelName')-
		>selectRaw('sum(ul.score) as score,l.levelName,w.worldName')->get();
	$columns = array('World Name','Level Name','Total Score');
	$callback = function() use ($score, $columns) {
		$file = fopen('php://output', 'w');
		fputcsv($file, $columns);
		foreach($score as $score) {
			fputcsv($file, array($score->worldName,$score->levelName,$score->score));
		}
		fclose($file);
	};
	return response()->stream($callback, 200, $headers);
}
```

12. Get all the world score for a selected student

```
public function getStudentScoreByWorld($userID,$worldID){
	$score = Users::join('user_unlock_level as ul','ul.userID', '=', 'user.userID')->join("world as
		w","ul.worldID","=","w.worldID")->join("level as l","ul.levelID","=","l.levelID")-
		>where('user.userID',$userID)->where('l.worldID',$worldID)->get();
	return response()->json($score, 200);
}
```

13. Generate performance report for a selected student

```
public function generateStudentReport($userID){
        $reportFront = strval($userID);
        $reportEnd = "-Performance-Report.csv";
        $reportName = $reportFront.$reportEnd;
        $headers = array(
                "Content-type" => "text/csv",
                "Content-Disposition" => "attachment; filename=".$reportName,
                "Pragma" => "no-cache",
                "Cache-Control" => "must-revalidate, post-check=0, pre-check=0",
                "Expires" => "0"
        );
        $score = Users::join('user_unlock_level as ul','ul.userID', '=', 'user.userID')->join("world as
                w","ul.worldID","=","w.worldID")->join("level as l","ul.levelID","=","l.levelID")-
                >where('user.userID',$userID)->selectRaw('ul.score,l.levelName,w.worldName')->get();
        $columns = array('World Name','Level Name','Total Score');
        $callback = function() use ($score, $columns) {
                $file = fopen('php://output', 'w');
                fputcsv($file, $columns);
                foreach($score as $score) {
                        fputcsv($file, array($score->worldName,$score->levelName,$score->score));
                }
                fclose($file);
        };
        return response()->stream($callback, 200, $headers);
}
```

User Creations

14. Get all the information of all the user-created levels

```
public function getAllUCL(){
	$UCLList = UCL::all();
	return response()->json($UCLList, 200);
}
```

15. Get the detailed information of a selected user-created level

```
public function getUCL($id){
	if(UCL::where('uclID', $id)->exists()) {
		$UCL = UCL::where('uclID', $id)->get();
		return response()->json($UCL, 200);
	} else {
		return response()->json([
			"message" => "UCL not found",
			"error" => 404
		], 404);
	}
}
```

16. Create a user-created level

```
public function postUCL(Request $request, $id){
	$UCL = new UCL;
	$UCL->userId = $id;
	$UCL->uclID = $request->uclID;
	$UCL->questionId = $request->questionId;
	$UCL->uclName = $request->uclName;
	$UCL->uclDesc = $request->uclDesc;
	$UCL->questionTitle = $request->questionTitle;
	$UCL->option1 = $request->option1;
	$UCL->option2 = $request->option2;
	$UCL->option3 = $request->option3;
	$UCL->option4 = $request->option4;
	$UCL->correctOption = $request->correctOption;
	$UCL->timestamps = false;
	$UCL->save();
	return response()->json([
		"message" => "UCL Created"
	], 200);
}
```

Discussion Board

17. Get all the information of all the discussion

```
public function getAllDiscussion(){
        $thread = Discussion::leftJoin('user', 'thread.userID', '=', 'user.userID')->get();
        return response()->json($thread, 200);
}
```

18. Get the detailed information of a selected discussion

```
public function getDetailedDiscussion($id){
        $detailed_thread = Discussion::where('threadID', $id)->leftJoin('user', 'thread.userID',
                '=', 'user.userID')->get();
        return response()->json($detailed_thread, 200);
}
```

19. Get the number of comments on a selected discussion

```
public function getNumOfComments($id){
        $numOfComments = Discussion::where('thread.threadID', $id)->join('comment',
                'thread.threadID', '=', 'comment.threadID')->count();
        return response()->json($numOfComments, 200);
}
```

20. Get all all the comments on a selected discussion

```
public function getDiscussionComments($id){
        $discussionComments = Discussion::where('thread.threadID', $id)->join('comment',
                'thread.threadID', '=', 'comment.threadID')->join('user', 'comment.postedBy', '=',
                'user.userID')->get();
        return response()->json($discussionComments, 200);
}
```

21. Post a discussion

```
public function postDiscussion(Request $request, $id){
        $thread = new Discussion;
        $thread->title = $request->title;
        $thread->details = $request->details;
        $thread->time_created = $request->time;
        $thread->userID = $id;
        $thread->timestamps = false;
        $thread->save();
        return response()->json([
                "message" => "Discussion Posted"
        ], 200);
}
```

22. Post a comment on a selected discussion

```
public function postComment(Request $request, $id){
        $comment = new Comment;
        $comment->content = $request->content;
        $comment->time_created = $request->time;
        $comment->postedBy = $request->postedBy;
        $comment->threadID = $id;
        $comment->timestamps = false;
        $comment->save();
        return response()->json([
                "message" => "Comment Posted"
        ], 200);
}
```

Assignment Manager

23. Get all the assignment that a student in a group have

```
public function getStudentAssignment($id){
        $groupAssignment = GroupAssignment::where('group_has_assignment.groupID', $id)-
                >join('assignment', 'group_has_assignment.assignmentID', '=',
                'assignment.assignmentID')->join('user', 'assignment.userID', '=', 'user.userID')-
                >join('assignment_file', 'assignment.fileID', '=', 'assignment_file.file_id')->get();
        return response()->json($groupAssignment, 20);
}
```

24. Get all the assignment that a teacher posted

```
public function getTeacherAllAssignment($id){
        $groupAssignment = Users::where('user.userID', $id)->join('assignment', 'user.userID', '=',
                'assignment.userID')->join('assignment_file', 'assignment.fileID', '=',
                'assignment_file.file_id')->get();
        return response()->json($groupAssignment, 200);
}
```

25. Get all the assignment that a teacher posted for a selected group

```
public function getTeacherGroupAssignment($id){
        $groupAssignment = GroupAssignment::where('group_has_assignment.groupID', $id)-
                >join('assignment', 'group_has_assignment.assignmentID', '=',
                'assignment.assignmentID')->join('user', 'assignment.userID', '=', 'user.userID')-
                >join('assignment_file', 'assignment.fileID', '=', 'assignment_file.file_id')->get();
        return response()->json($groupAssignment, 200);
}
```

26. Get all the file of assignments that is in the database

```
public function getAllFile(){
        $file = AssignmentFile::all();
        return response()->json($file, 200);
}
```

27. Post an assignment

```
public function postAssignment(Request $request, $id){
        $fileID = AssignmentFile::where('assignment_file.file_name', $request->file_id)-
                >pluck('assignment_file.file_id');

        $assignment = new Assignment;
        $assignment->title = $request->title;
        $assignment->details = $request->details;
        $assignment->due_date = $request->due_date;
        $assignment->userID = $id;
        $assignment->fileID = $fileID[0];
        $assignment->timestamps = false;
        $assignment->save();

        $assignmentID = Assignment::orderBy('assignmentID', 'DESC')->take(1)-
                >pluck('assignment.assignmentID');

        $groupAssignment = new GroupAssignment;
        $groupAssignment->groupID = $request->group_id;
        $groupAssignment->assignmentID = $assignmentID[0];
        $groupAssignment->timestamps = false;
        $groupAssignment->save();

        $fileURL = AssignmentFile::where('assignment_file.file_name', $request->file_id)-
                >pluck('assignment_file.file_url');

        return response()->json([
                "message" => $fileURL
        ], 200);

}
```

Game Manager

28. Get all the information of all the world

```
public function getAllWorlds(){
		$world = World::all();
		return response()->json($world, 200);
}
```

29. Get all the world that a user has unlocked

```
public function getUserUnlockedWorlds($id){
		$user_unlocked_worlds = UserUnlockWorld::where('userID', $id)->get();
		return response()->json($user_unlocked_worlds, 200);
}
```

30. Get all the levels in a selected world

```
public function getLevelsInWorld($id){
		$level = Level::where('worldID', $id)->get();
		return response()->json($level, 200);
}
```

31. Get all the levels that a user has unlocked in a selected world

```
public function getUserUnlockedLevels($userID,$worldID){
		$user_unlocked_levels = UserUnlockLevel::where('userID', $userID)->where('worldID',
				$worldID)->get();
		return response()->json($user_unlocked_levels, 200);
}
```

32. Get all the information of the powerups in the game

```
public function getPowerupsInfo(){
        $powerups = Powerup::all();
        return response()->json($powerups, 200);
}
```

33. Get the user inventory

```
public function getUserInventory($id){
        $user_inventory = UserPowerup::where('userID', $id)->get();
        return response()->json($user_inventory, 200);
}
```

34. Update the user inventory when he buy a powerup

```
public function updateUserInventory(Request $request,$userid){
        $user_inventory = UserPowerup::where('userID',$userid)->where('powerID',$request-
                >powerID)->first();
        if($request->add == "True"){
                $user_inventory->quantity++;
        }else{
                $user_inventory->quantity--;
        }
        $user_inventory->timestamps = false;
        $user_inventory->save();

        return response()->json([
                "message" => $user_inventory->quantity
        ], 200);
}
```

35. Update the user inventory after a game

```
public function updateUserInventoryAfterGame(Request $request,$userid){
        $user_inventory_ff = UserPowerup::where('userID',$userid)->where('powerID',1)->first();
        $user_inventory_time = UserPowerup::where('userID',$userid)->where('powerID',2)->first();
        $user_inventory_ff->quantity = $request->power1Quantity;
        $user_inventory_time->quantity = $request->power2Quantity;
        $user_inventory_ff->timestamps = false;
        $user_inventory_ff->save();
        $user_inventory_time->timestamps = false;
        $user_inventory_time->save();

        return response()->json([
                "message" => "User Inventory Updated"
        ], 200);
}
```

36. Get the level leaderboard when a user selects a level

```
public function getLevelLeaderboard($levelID){
        $lvl_leaderboard = UserUnlockLevel::where('levelID',$levelID)->orderBy('score', 'DESC')-
                >take(5)->join('user','user.userID','=','user_unlock_level.userID')->get();
        return response()->json($lvl_leaderboard, 200);
}
```

37. Get the user highest level cleared

```
public function getUserHighestLvl($userID){
        $highest_lvl = UserUnlockLevel::where('userID',$userID)->where('unlock',1)->where('score',0)-
                >join('level','user_unlock_level.levelID','=','level.levelID')->get();

        if(sizeof($highest_lvl) == 0){
                $highest_lvl = UserUnlockLevel::where('userID',$userID)->where('unlock',1)-
                        >join('level','user_unlock_level.levelID','=','level.levelID')-
                        >orderBy('level.levelID', 'DESC')->first();
                $totalScore = UserUnlockLevel::where('userID',$userID)->sum('score');
                $highest_lvl["totalscore"] = $totalScore;
                return response()->json($highest_lvl, 200);
        }else{
                $totalScore = UserUnlockLevel::where('userID',$userID)->sum('score');
                $highest_lvl[0]["totalscore"] = $totalScore;
                return response()->json($highest_lvl[0], 200);
        }
}
```

38. Get all the information of the questions in a selected level

```
public function getQuestions($levelID,$worldID){
        $questions = Question::where('levelID',$levelID)->where('worldID',$worldID)->get();
        return response()->json($questions,200);
}
```

39. Update the information of the user when he cleared a level

```
public function updateUserGameClear(Request $request,$userID){
        $newScore = $request->score;
        $user = Users::find($userID);
        $user->currency += $newScore;
        $user->timestamps = false;
        $user->save();
        $user_curr_unlock_level = UserUnlockLevel::where('userID',$userID)->where('levelID',$request-
                >levelID)->first()'
        $curr_Score = $user_curr_unlock_level->score;

        if($newScore > $curr_Score){
                $user_curr_unlock_level->score = $newScore;
                $user_curr_unlock_level->timestamps = false;
                $user_curr_unlock_level->save();
                $user_curr_unlock_world = UserUnlockWorld::where('userID',$userID)-
                        >where('worldID',$request->worldID)->first();
                $newCalcScore = UserUnlockLevel::where('userID',$userID)->groupBy('worldID')-
                        >where('worldID',$request->worldID)->sum('score');
                $user_curr_unlock_world->score = $newCalcScore;
                $user_curr_unlock_world->timestamps = false;
                $user_curr_unlock_world->save();
        }

        $curr_level_id = intval($request->levelID);
        $next_level = strval($curr_level_id+=1);

        if($next_level != '31'){
                $user_next_unlock_level = UserUnlockLevel::where('userID',$userID)->where('levelID',
                $next_level)->first();
                $user_next_unlock_level->unlock = 1;
                $user_next_unlock_level->timestamps = false;
                $user_next_unlock_level->save();
        }

        $curr_level_id = intval($request->levelID);
        $curr_world_id = $request->worldID;
        $worldName = substr($curr_world_id, 0, 5);
        $worldNum = intval(substr($curr_world_id, 5));
        $worldNum++;

        if($worldNum < 7 && ($curr_level_id%5 == 0)){
                $worldName .= strval($worldNum);
                $user_next_unlock_world = UserUnlockWorld::where('userID',$userID)-
                        >where('worldID',$worldName)->first();
                $user_next_unlock_world->unlock = 1;
                $user_next_unlock_world->timestamps = false;
                $user_next_unlock_world->save();
        }

        return response()->json([
                "message" => $newCalcScore
        ], 200);

}
```

Leaderboard

40. Get all the information of the leaderboard

```
public function getLeaderboard(Request $request,$worldID){

        if($request->type == “class"){

                $ldrboard = Users::where('userGroup',$request->group)-
                        >join('user_unlock_world','user_unlock_world.userID', '=','user.userID')-
                        >where(‘worldID',$worldID)->orderBy('score','DESC')->take(5)->get();

                $rank = Users::where('userGroup',$request->group)->join('user_unlock_world     as
                        u1’,'u1.userID','=','user.userID')->where("u1.worldID",$worldID)->get();

                $userRank = Users::where('user.userID',$request->userID)->join('user_unlock_world as
                        u1','u1.userID', ‘=','user.userID')->where("u1.worldID",$worldID)->get();

                $count = 1;

                foreach ($rank as $i) {
                        if((int)$userRank[0]['score'] < (int)$i['score']){
                                $count+= 1;
                        }
                }

                $user = UserUnlockWorld::where('worldID',$worldID)->where('userID',$request-
                        >userID)->get();

                $user[0]['rank'] = $count;
                $ldrboard->push($user[0]);

                return response()->json($ldrboard, 200);
        }else{

                $ldrboard = Users::join('user_unlock_world','user_unlock_world.userID',
                        '=','user.userID')->where('worldID',$worldID)->orderBy('score','DESC')-
                        >take(5)->get();

                $rank = UserUnlockWorld::where('user_unlock_world.userID', $request->userID)-
                        >join('user_unlock_world as u1','u1.userID','<>','user_unlock_world.userID')-
                        >where('u1.worldID',$worldID)->where('user_unlock_world.worldID',$worldID)-
                        >join('user_unlock_world as u2','u2.score', '>', 'user_unlock_world.score')-
                        >where(‘u2.worldID',$worldID)->distinct('u2.userID')->count('u2.userID');

                $rank += 1;
                $len = sizeof($ldrboard)-1;

                $user = UserUnlockWorld::where('worldID',$worldID)->where('userID',$request-
                        >userID)->get();

                $user[0]['rank'] = $rank;
                $ldrboard->push($user[0]);

                return response()->json($ldrboard, 200);
        }
}
```