EX 8

Câu 1: Hãy nêu Https method dùng cho API sửa và xóa?

* PUT (UPDATE): Cập nhật thông tin cho Resource.
* DELETE (DELETE): Xoá một Resource.

Câu 2: Validate là gì? Cách khai báo và kiểm tra validate được viết như thế nào?

Validate là kiểm tra dữ liệu request được gửi lên trước khi nó được xử lý.

Khai báo và kiểm tra validate trong laravel:  
$validated = $request→validate([

‘title’ => ‘required|unique:posts|max:255’,

‘body’ => ‘required’,

])

Câu 3: Response là gì? Cấu trúc dữ liệu trả response để client/view sử dụng?

Response là dữ liệu server trả về cho client.

Cấu trúc response:

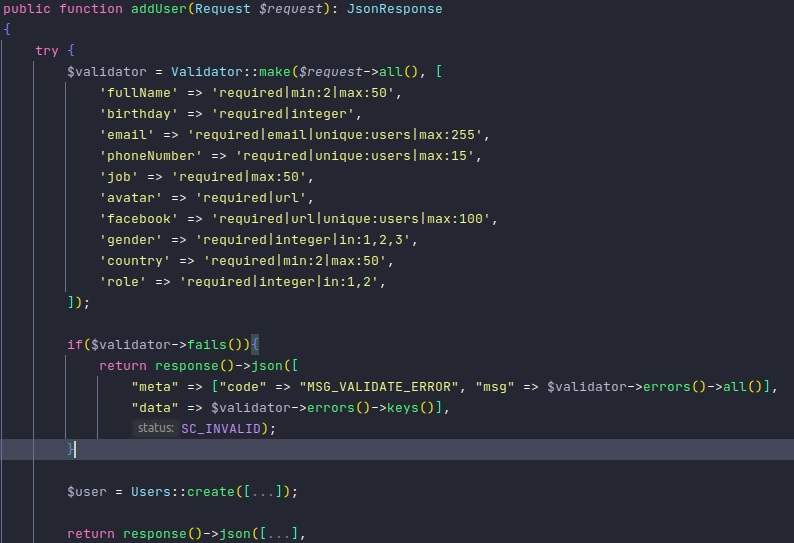
<http-version> <status> <reason-pharse> //Status-line

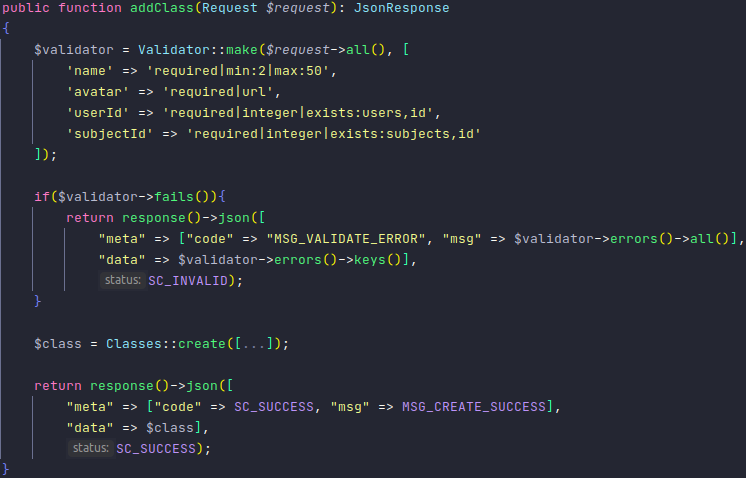
<headers>

<body>

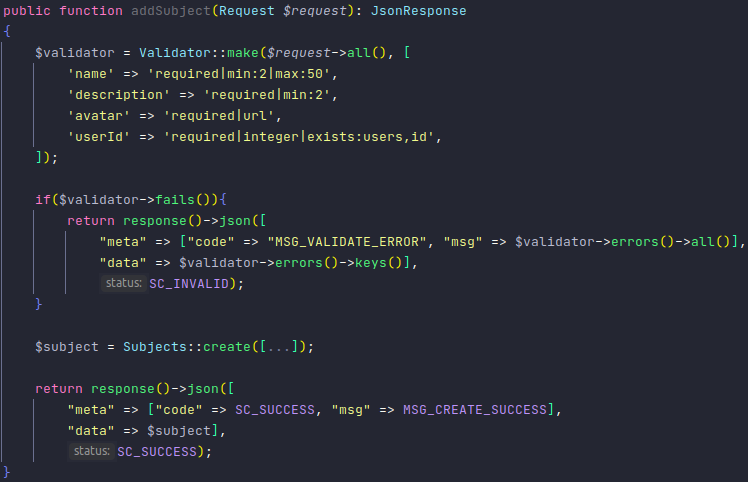
Câu 4: Thực hiện Validate cho API thêm mới Users, Classes, Subjects, CourseRequests?

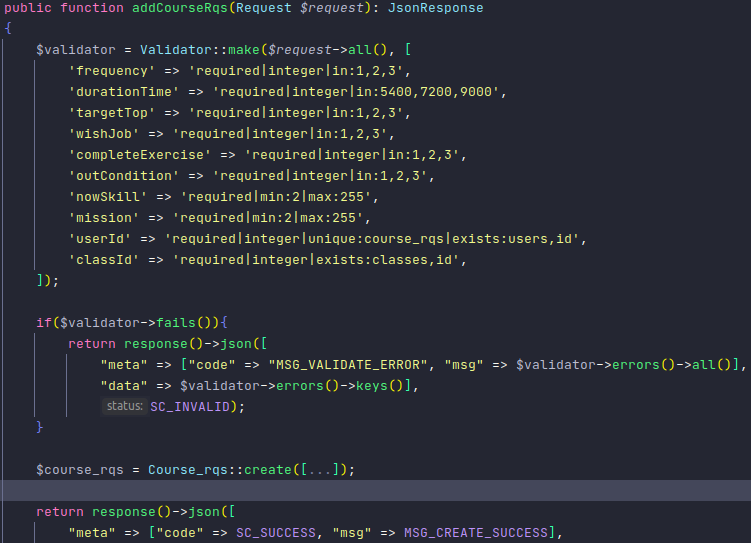
Users

Classes



Subjects

CourseRequests



Câu 5: Thực hiện validate và viết API update Users, Classes, Subjects, CourseRequests?

Update Users

public function updateUser(Request *$request*, *$id*): JsonResponse  
{  
 $user = Users::findOrFail(*$id*);  
 $validator = Validator::make(*$request*->all(), [  
 'fullName' => 'required|min:2|max:50',  
 'birthday' => 'required|integer',  
 'email' => 'required|email|unique:users|max:255',  
 'phoneNumber' => 'required|unique:users|max:15',  
 'job' => 'required|max:50',  
 'avatar' => 'required|url',  
 'facebook' => 'required|url|unique:users|max:100',  
 'gender' => 'required|integer|in:1,2,3',  
 'country' => 'required|min:2|max:50',  
 'role' => 'required|integer|in:1,2',

'status' => 'required|integer|in:1,2,3,4'  
 ]);  
  
 if($validator->fails()){  
 return response()->json([  
 "meta" => ["code" => "MSG\_VALIDATE\_ERROR", "msg" => $validator->errors()->all()],  
 "data" => $validator->errors()->keys()],  
 SC\_INVALID);  
 }  
 $user->fullName = *$request*->fullName;  
 $user->birthday = *$request*->birthday;  
 $user->email = *$request*->email;  
 $user->phoneNumber = *$request*->phoneNumber;  
 $user->job = *$request*->job;  
 $user->avatar = *$request*->avatar;  
 $user->facebook = *$request*->facebook;  
 $user->gender = *$request*->gender;  
 $user->country = *$request*->country;  
 $user->role = *$request*->role;  
 $user->status = *$request*->status;  
 $user->updated\_at = now();  
  
 $user->save();  
  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_UPDATE\_SUCCESS],  
 "data" => $user],  
 SC\_SUCCESS);  
 }  
}

Update Classes

public function updateClas s(Request *$request*, *$id*): JsonResponse  
{  
 $class = Classes::findOrFail(*$id*);  
 $validator = Validator::make(*$request*->all(), [  
 'name' => 'required|min:2|max:50',  
 'avatar' => 'required|url',  
 'userId' => 'required|integer|exists:users,id',  
 'subjectId' => 'required|integer|exists:subjects,id',  
 'status' => 'required|integer|in:1,2,3'  
 ]);  
  
 if($validator->fails()){  
 return response()->json([  
 "meta" => ["code" => "MSG\_VALIDATE\_ERROR", "msg" => $validator->errors()->all()],  
 "data" => $validator->errors()->keys()],  
 SC\_INVALID);  
 }  
 $class->name = *$request*->name;  
 $class->avatar = *$request*->avatar;  
 $class->userId = *$request*->userId;  
 $class->subjectId = *$request*->subjectId;  
 $class->status = *$request*->status;  
 $class->updated\_at = now();  
  
 $class->save();  
  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_UPDATE\_SUCCESS],  
 "data" => $class],  
 SC\_SUCCESS);  
 }  
}

Update Subjects

public function updateSubject(Request *$request*, *$id*): JsonResponse  
{  
 $subject = Subjects::findOrFail(*$id*);  
 $validator = Validator::make(*$request*->all(), [  
 'name' => 'required|min:2|max:50',  
 'description' => 'required|min:2',  
 'avatar' => 'required|url',  
 'userId' => 'required|integer|exists:users,id',  
 'status' => 'required|integer|in:1,2'  
 ]);  
  
 if($validator->fails()){  
 return response()->json([  
 "meta" => ["code" => "MSG\_VALIDATE\_ERROR", "msg" => $validator->errors()->all()],  
 "data" => $validator->errors()->keys()],  
 SC\_INVALID);  
 }  
 $subject->name = *$request*->name;  
 $subject->description = *$request*->description;  
 $subject->avatar = *$request*->avatar;  
 $subject->userId = *$request*->userId;  
 $subject->status = *$request*->status;  
 $subject->updated\_at = now();  
  
 $subject->save();  
  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_UPDATE\_SUCCESS],  
 "data" => $subject],  
 SC\_SUCCESS);  
 }  
}

Update CourseRequest

public function updateCourseRqs(Request *$request*, *$id*): JsonResponse  
{  
 $course\_rqs = Course\_rqs::findOrFail(*$id*);  
 $validator = Validator::make(*$request*->all(), [  
 'frequency' => 'required|integer|in:1,2,3',  
 'durationTime' => 'required|integer|in:5400,7200,9000',  
 'targetTop' => 'required|integer|in:1,2,3',  
 'wishJob' => 'required|integer|in:1,2,3',  
 'completeExercise' => 'required|integer|in:1,2,3',  
 'outCondition' => 'required|integer|in:1,2,3',  
 'nowSkill' => 'required|min:2|max:255',  
 'mission' => 'required|min:2|max:255',  
 'classId' => 'required|integer|exists:classes,id',  
 'status' => 'required|integer|in:1,2,3'  
 ]);  
  
 if($validator->fails()){  
 return response()->json([  
 "meta" => ["code" => "MSG\_VALIDATE\_ERROR", "msg" => $validator->errors()->all()],  
 "data" => $validator->errors()->keys()],  
 SC\_INVALID);  
 }  
 $course\_rqs->frequency = *$request*->frequency;  
 $course\_rqs->durationTime = *$request*->durationTime;  
 $course\_rqs->targetTop = *$request*->targetTop;  
 $course\_rqs->wishJob = *$request*->wishJob;  
 $course\_rqs->completeExercise = *$request*->completeExercise;  
 $course\_rqs->outCondition = *$request*->outCondition;  
 $course\_rqs->nowSkill = *$request*->nowSkill;  
 $course\_rqs->mission = *$request*->mission;  
 $course\_rqs->classId = *$request*->classId;  
 $course\_rqs->status = *$request*->status;  
 $course\_rqs->updated\_at = now();  
  
 $course\_rqs->save();  
  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_UPDATE\_SUCCESS],  
 "data" => $course\_rqs],  
 SC\_SUCCESS);  
 }  
}

Câu 6: Thực hiện viết API detele cho Users, Classes, Subjects, CourseRequests?

Delete Users

public function deleteUser(*$id*): JsonResponse  
{  
 try {  
 $user = Users::findOrFail(*$id*);  
 $user->delete();  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_DELETE\_SUCCESS],  
 "data" => *$id*],  
 SC\_SUCCESS);  
  
 } catch (ModelNotFoundException $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_QUERY\_ERROR, "msg" => $ex->getMessage()],  
 "data" => $ex->getIds()],  
 SC\_SERVER\_ERROR);  
 } catch (Exception $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_SERVER\_ERROR, "msg" => 'SERVER\_ERROR'],  
 "data" => $ex],  
 SC\_SERVER\_ERROR);  
 }  
}

Delete Classes

public function deleteClass(*$id*): JsonResponse  
{  
 try {  
 $class = Classes::findOrFail(*$id*);  
 $class->delete();  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_DELETE\_SUCCESS],  
 "data" => *$id*],  
 SC\_SUCCESS);  
  
 } catch (ModelNotFoundException $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_QUERY\_ERROR, "msg" => $ex->getMessage()],  
 "data" => $ex->getIds()],  
 SC\_SERVER\_ERROR);  
 } catch (Exception $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_SERVER\_ERROR, "msg" => 'SERVER\_ERROR'],  
 "data" => $ex],  
 SC\_SERVER\_ERROR);  
 }  
}

Delete Subjects

public function deleteSubject(*$id*): JsonResponse  
{  
 try {  
 $subject = Subjects::findOrFail(*$id*);  
 $subject->delete();  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_DELETE\_SUCCESS],  
 "data" => *$id*],  
 SC\_SUCCESS);  
  
 } catch (ModelNotFoundException $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_QUERY\_ERROR, "msg" => $ex->getMessage()],  
 "data" => $ex->getIds()],  
 SC\_SERVER\_ERROR);  
 } catch (Exception $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_SERVER\_ERROR, "msg" => 'SERVER\_ERROR'],  
 "data" => $ex],  
 SC\_SERVER\_ERROR);  
 }  
}

Delete CourseRequests

public function deleteCourseRqs(*$id*): JsonResponse  
{  
 try {  
 $course\_rqs = Course\_rqs::findOrFail(*$id*);  
 $course\_rqs->delete();  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_DELETE\_SUCCESS],  
 "data" => *$id*],  
 SC\_SUCCESS);  
  
 } catch (ModelNotFoundException $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_QUERY\_ERROR, "msg" => $ex->getMessage()],  
 "data" => $ex->getIds()],  
 SC\_SERVER\_ERROR);  
 } catch (Exception $ex){  
 return response()->json([  
 "meta" => ["code" => SC\_SERVER\_ERROR, "msg" => 'SERVER\_ERROR'],  
 "data" => $ex],  
 SC\_SERVER\_ERROR);  
 }  
}

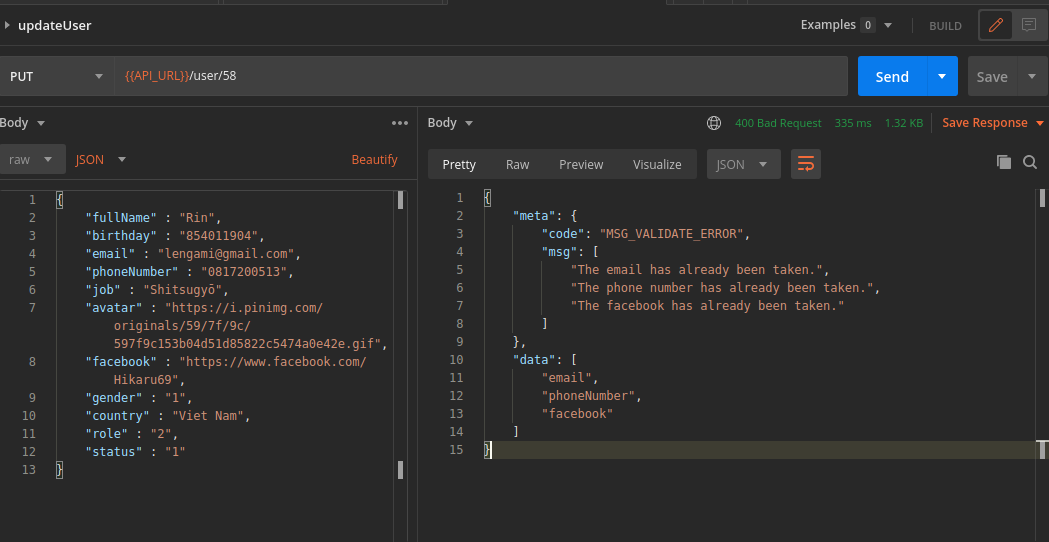
Câu 7: Try-Catch là gì? Tác dụng của nó? Thực hiện viết TryCatch cho API?

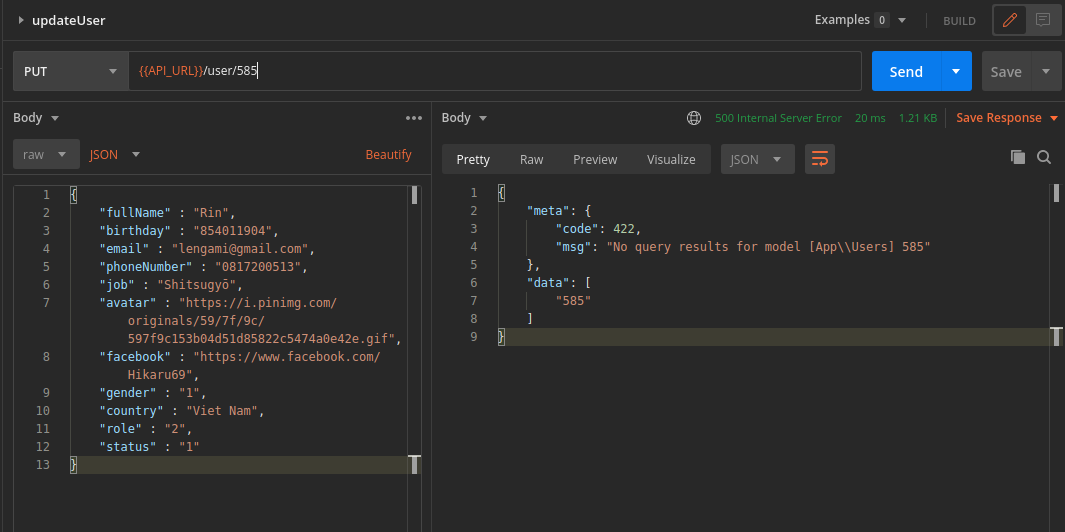
Khi chạy chương trình, có rất nhiều loại lỗi khác nhau có thể xảy ra như: lỗi do sai lầm người viết, lỗi do sai thông tin đầu vào hoặc những lỗi mà không thể lường trước được.

Try-Catch dùng để bắt các ngoại lệ có thể xảy ra trong quá trình chạy chương trình

public function updateUser(Request *$request*, *$id*): JsonResponse  
{  
 try {  
 $user = $this->users->findOrFail(*$id*);  
 $validator = Validator::make(*$request*->all(), [  
 'fullName' => 'required|min:2|max:50',  
 'birthday' => 'required|integer',  
 'email' => 'required|email|unique:users|max:255',  
 'phoneNumber' => 'required|unique:users|max:15',  
 'job' => 'required|max:50',  
 'avatar' => 'required|url',  
 'facebook' => 'required|url|unique:users|max:100',  
 'gender' => 'required|integer|in:1,2,3',  
 'country' => 'required|min:2|max:50',  
 'role' => 'required|integer|in:1,2',  
 'status' => 'required|integer|in:1,2,3,4'  
 ]);  
  
 if ($validator->fails()) {  
 return response()->json([  
 "meta" => ["code" => "MSG\_VALIDATE\_ERROR", "msg" => $validator->errors()->all()],  
 "data" => $validator->errors()->keys()],  
 SC\_INVALID);  
 }  
 $user->fullName = *$request*->fullName;  
 $user->birthday = *$request*->birthday;  
 $user->email = *$request*->email;  
 $user->phoneNumber = *$request*->phoneNumber;  
 $user->job = *$request*->job;  
 $user->avatar = *$request*->avatar;  
 $user->facebook = *$request*->facebook;  
 $user->gender = *$request*->gender;  
 $user->country = *$request*->country;  
 $user->role = *$request*->role;  
 $user->status = *$request*->status;  
 $user->updated\_at = now();  
  
 $user->save();  
  
 return response()->json([  
 "meta" => ["code" => SC\_SUCCESS, "msg" => MSG\_UPDATE\_SUCCESS],  
 "data" => $user],  
 SC\_SUCCESS);  
 } catch (ModelNotFoundException $ex) {  
 return response()->json([  
 "meta" => ["code" => SC\_QUERY\_ERROR, "msg" => $ex->getMessage()],  
 "data" => $ex->getIds()],  
 SC\_SERVER\_ERROR);  
 } catch (Exception $ex) {  
 return response()->json([  
 "meta" => ["code" => SC\_SERVER\_ERROR, "msg" => "SERVER ERROR"],  
 "data" => $ex->getPrevious()],  
 SC\_SERVER\_ERROR);  
 }  
}

Validate Error

ModelNotFoundException



Success

