**2.31 - PHP PDO Tutorial Part 2 - Transactions - Env Variables & PHPDotEnv - Full PHP 8 Tutorial**

Video giải thích cách kết nối PHP với cơ sở dữ liệu MySQL sử dụng PDO. Nó đề cập đến các vấn đề:

* Sử dụng các biến cứng để lưu thông tin kết nối có thể dẫn đến lỗ hổng bảo mật.
* Giải pháp là sử dụng các biến môi trường để lưu thông tin nhạy cảm. Tạo tệp .env chứa các biến môi trường và tệp .env.example không chứa giá trị để commit lên repo.
* Sử dụng gói phpdotenv để tải các biến môi trường từ tệp .env.
* Sử dụng cơ chế giao dịch (transaction) để đảm bảo tính toàn vẹn dữ liệu khi thực hiện nhiều truy vấn. Bắt đầu transaction, thực hiện các truy vấn, commit nếu thành công hoặc rollback nếu có lỗi.
* Tách biệt các mối quan tâm trong code. Controller chỉ nên gọi các phương thức từ model chứ không nên trực tiếp thực hiện các truy vấn.

Như vậy video đã giới thiệu cách kết nối với cơ sở dữ liệu an toàn và đúng chuẩn trong PHP.

1. Video giảng dạy các kiến thức cơ bản về kết nối cơ sở dữ liệu (CSDL) sử dụng ngôn ngữ SQL và PHP/PDO. Giải thích khái niệm injection trong SQL để tránh bị tấn công.

- Cách kết nối tới CSDL MySQL sử dụng ngôn ngữ truy vấn SQL

- Sử dụng thư viện PDO trong PHP để kết nối tới CSDL

- Cách viết và thực thi các câu truy vấn SELECT, INSERT, UPDATE với PHP/PDO

- Giải thích khái niệm SQL injection để tránh bị tấn công bảo mật

1. Giới thiệu về database transaction - giao dịch trong CSDL để đảm bảo tính nhất quán dữ liệu khi thực hiện nhiều thao tác liên quan.

* Dùng để nhóm nhiều thao tác liên quan thành một giao dịch
* Đảm bảo tính nhất quán dữ liệu, nếu có lỗi có thể rollback toàn bộ giao dịch
* Sử dụng beginTransaction(), commit(), rollback() để quản lý transaction.

1. Sử dụng file .env và biến môi trường để lưu trữ thông tin nhạy cảm như username, password thay vì hardcore trong code. Giúp code sạch sẽ và an toàn hơn.

* Dùng để lưu các thông tin nhạy cảm như username, password
* Tránh việc lưu hardcoded các thông tin quan trọng trong code
* Tăng tính bảo mật và dễ quản lý hơn

1. Tách biệt các layer model, view, controller để tránh việc controller biết quá nhiều chi tiết về CSDL. Tạo model riêng để làm việc với dữ liệu.

* Giúp tách các tầng logic riêng biệt
* Controller chỉ quản lý logic xử lý request, không biết chi tiết về data
* Model làm việc trực tiếp với CSDL, không biết giao diện
* View chỉ định nghĩa giao diện, không biết logic xử lý

1. Sử dụng view để tách biệt giao diện người dùng với logic nghiệp vụ.

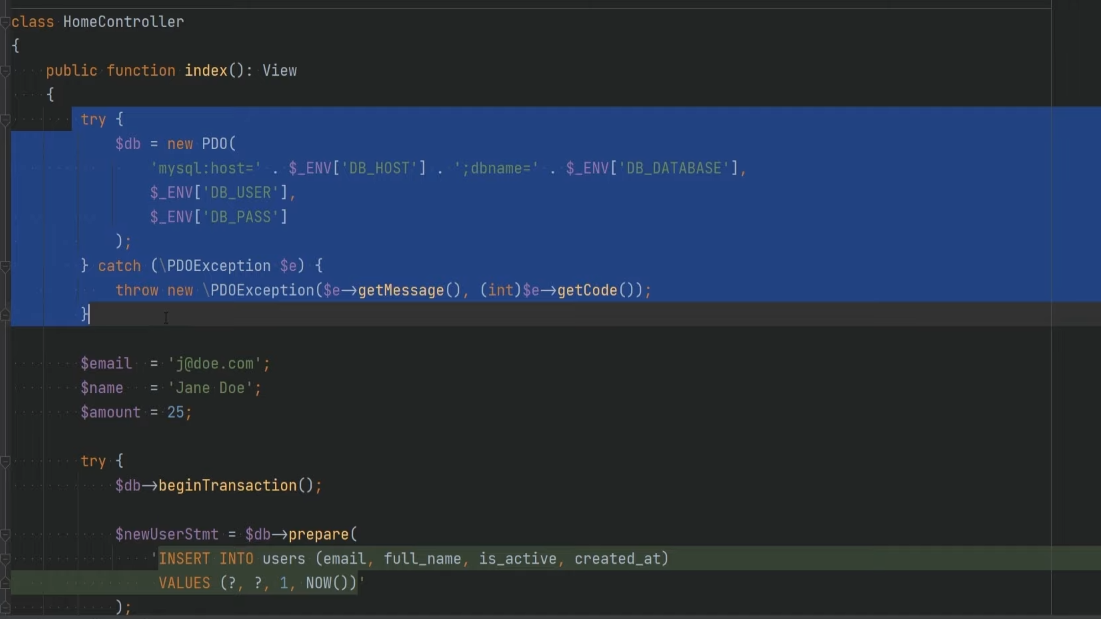
* Tách biệt phần hiển thị với logic xử lý code
* Dễ dàng thay đổi giao diện mà không ảnh hưởng logic
* Nâng cao tính bảo trì và tái sử dụng code

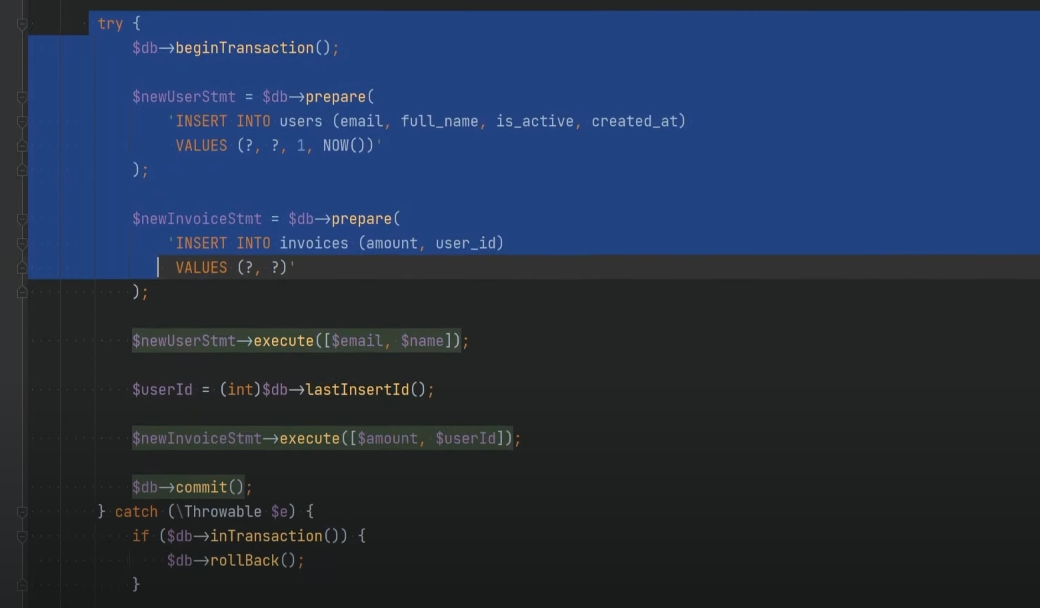
1. Ví dụ minh họa ứng dụng transaction, exception handling để xử lý việc tạo user thành công nhưng tạo invoice thất bại.

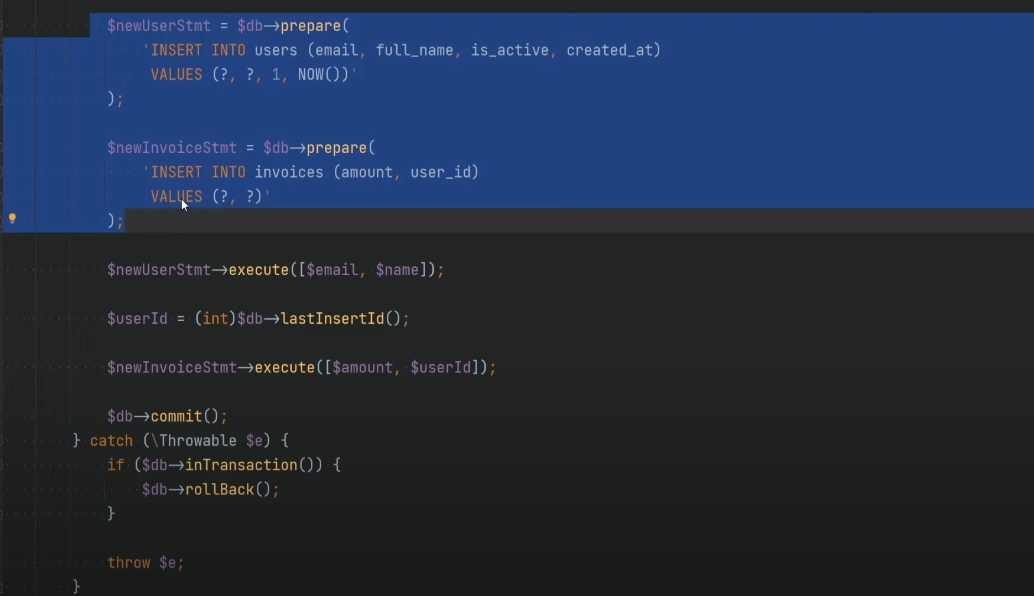
* Xử lý trường hợp tạo user thành công nhưng tạo invoice thất bại
* Sử dụng transaction để rollback lại toàn bộ thao tác khi có lỗi
* Sử dụng try catch để bắt và xử lý exception một cách tốt hơn

**2.32: PHP PDO Tutorial Part 3 - Models & Refactoring - Full PHP 8 Tutorial**

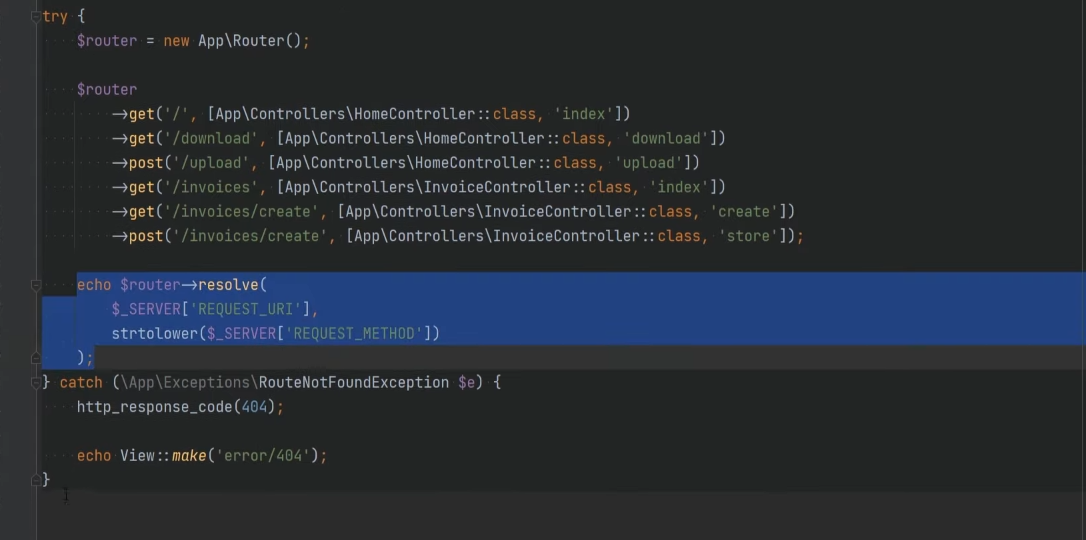
* Vấn đề:
  + Trước đó code được viết không tốt, home controller làm quá nhiều việc: kết nối database, chạy truy vấn, tạo user, invoice, etc.
  + Cần tách các chức năng ra để dễ quản lý và bảo trì code.





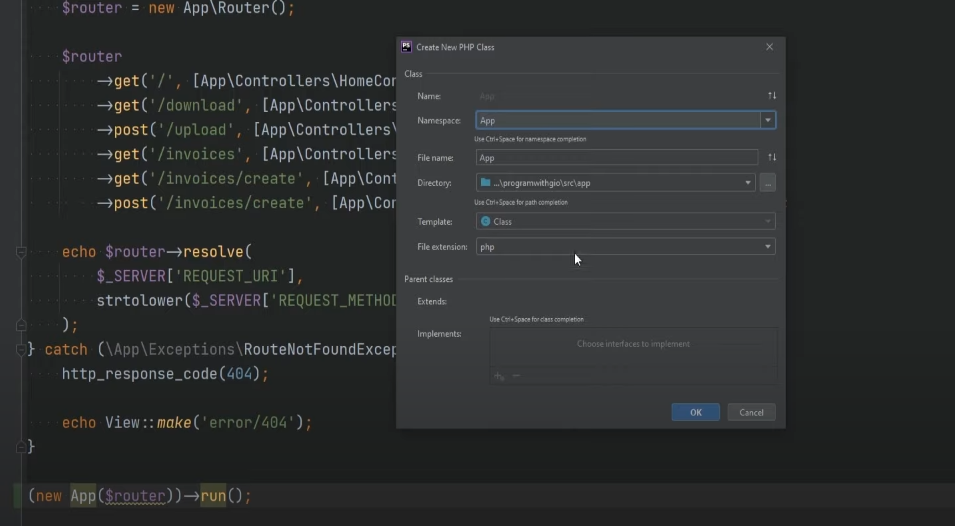


Trong vài video gần đây, chúng ta đã nói về pdo, các giao dịch cơ sở dữ liệu mysql và các thứ tương tự. Chúng ta đã viết một số đoạn code nhưng bây giờ đã đến lúc tái cấu trúc lại code này để tránh làm quá nhiều việc trong home controller của chúng ta. Hiện tại nó kết nối đến cơ sở dữ liệu vì vậy nó chịu trách nhiệm về kết nối cơ sở dữ liệu, sau đó chạy một số truy vấn, tạo người dùng, hóa đơn, v.v. Vì vậy, ở thời điểm này nó trông khá xấu và chúng ta cần tái cấu trúc nó.

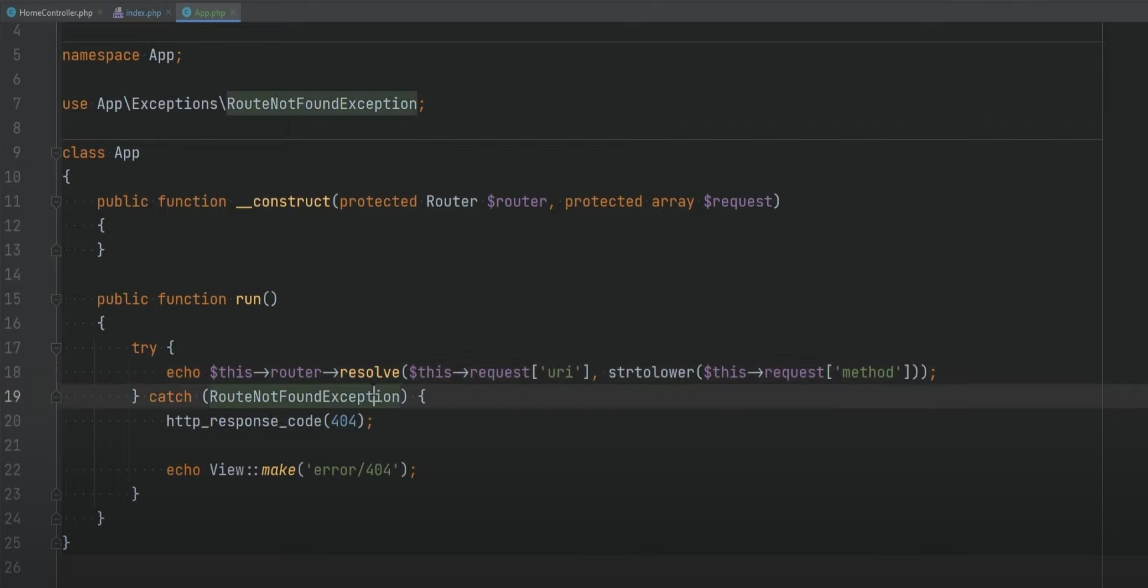


* Giải pháp:
  + Tách file index.php thành các class riêng biệt như App, Router.
  + Chỉ giữ code khởi tạo cần thiết trong index.php.
  + Chuyển kết nối database vào class App dưới dạng static method để truy xuất single instance.
  + Tạo class Database riêng để wrap quanh PDO thay vì dùng PDO trực tiếp.
  + Tương tự với class Config để lấy config từ $\_ENV.
  + Chuyển các xử lý nghiệp vụ vào các model tương ứng thay vì để trong controller.

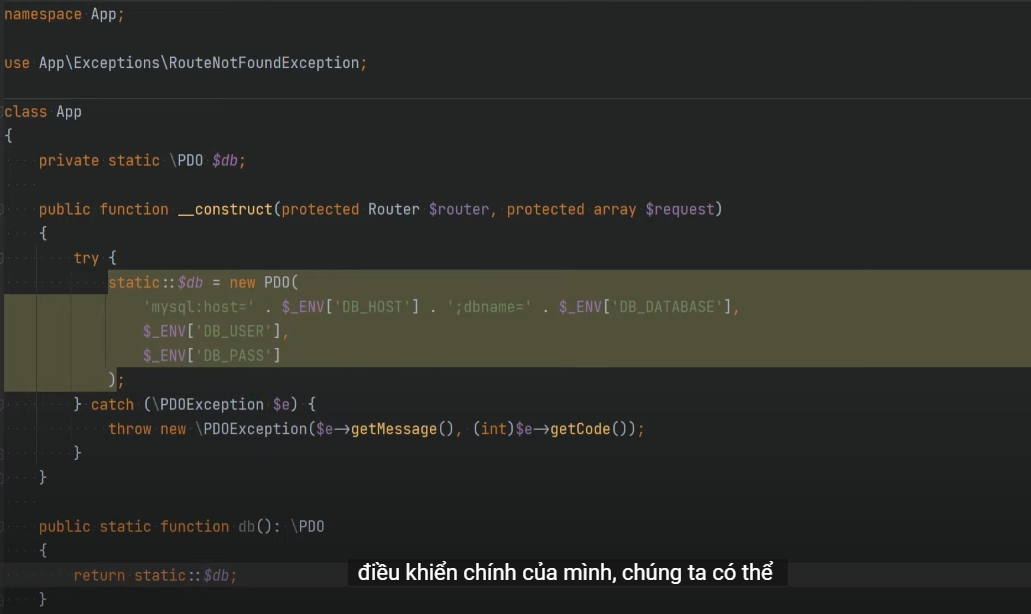
Đầu tiên, chúng ta cần xem xét file index.php, đây là điểm vào ứng dụng của chúng ta. Tôi muốn tách lệnh echo này ra khỏi đây và khối try catch. Chúng ta có thể tạo một lớp gọi là app.php hoặc application.php sẽ chịu trách nhiệm khởi động và chạy ứng dụng của chúng ta.

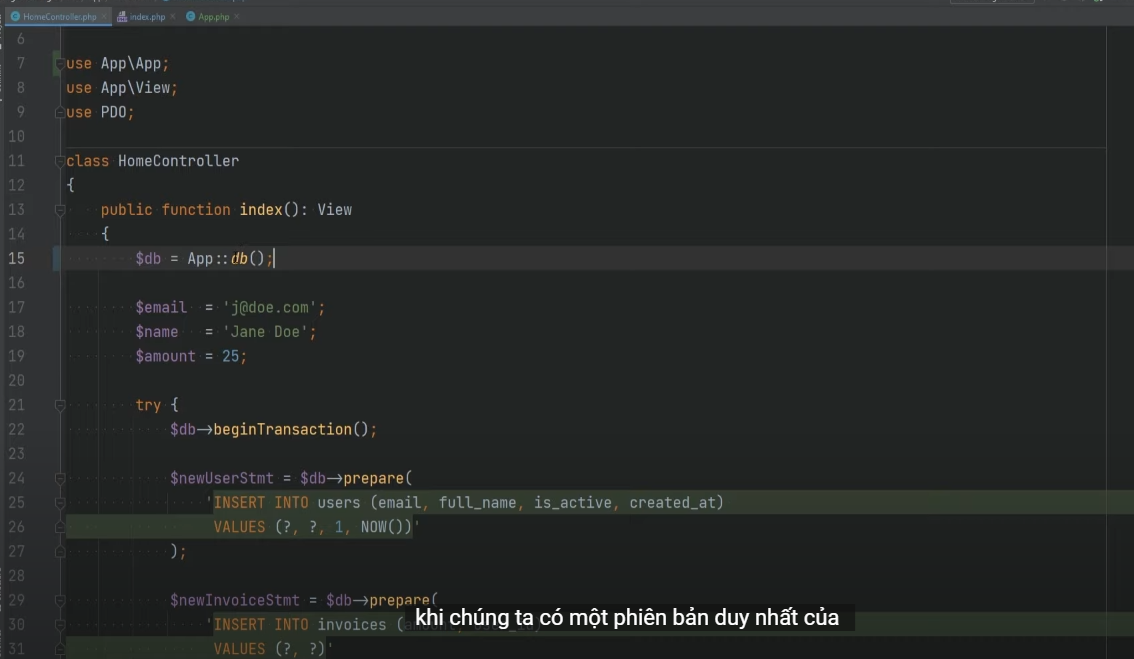


Tôi muốn đoạn code này trông giống như thế này: chúng ta có một lớp mới gọi là App và sau đó chúng ta có một phương thức trên nó gọi là run và để làm cho nó linh hoạt hơn, chúng ta có thể truyền đối tượng router xuống trong các đối số của hàm khởi tạo. Như vậy nếu chúng ta muốn, chúng ta có thể truyền một đối tượng router khác.

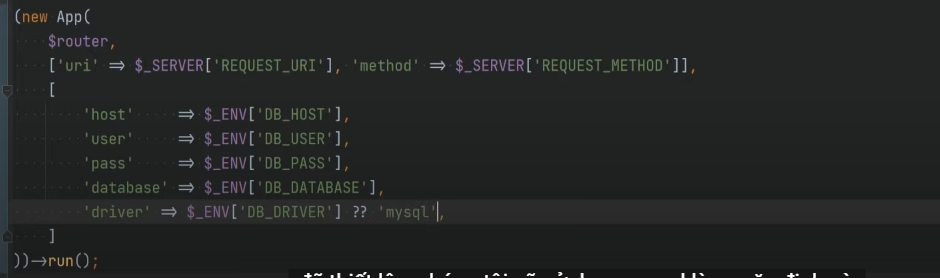


Chúng ta hãy tạo lớp này. Tôi sẽ trích xuất phần try catch khỏi đây và lệnh resolve route. Bây giờ chúng ta chỉ còn lại code để tải file .env, một số hằng số, tạo đối tượng router và sau đó truyền router đó xuống đối tượng app. Tiếp theo, chúng ta cần có một đối tượng kết nối cơ sở dữ liệu có sẵn trong toàn bộ ứng dụng của chúng ta. Hiện tại, kết nối cơ sở dữ liệu chỉ có sẵn trong home controller. Nếu chúng ta cần kết nối cơ sở dữ liệu trong một lớp hoặc phương thức khác của controller, chúng ta sẽ cần sao chép lại đoạn code này và về cơ bản sẽ tạo nhiều kết nối đến cơ sở dữ liệu, điều này không tốt.





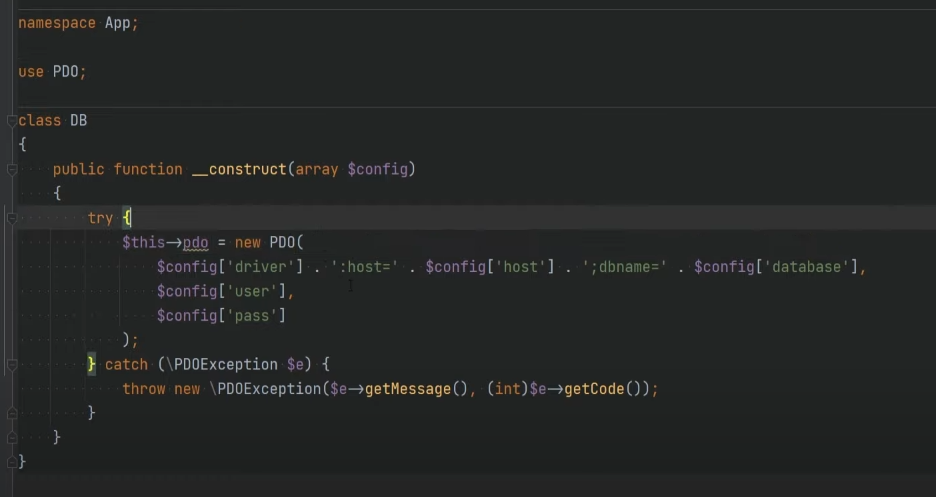


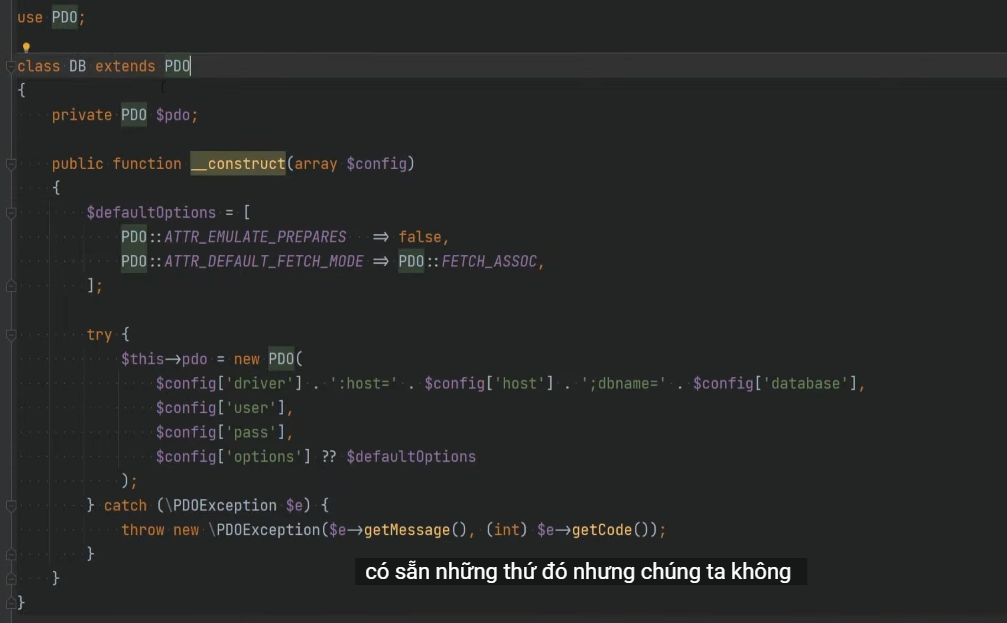


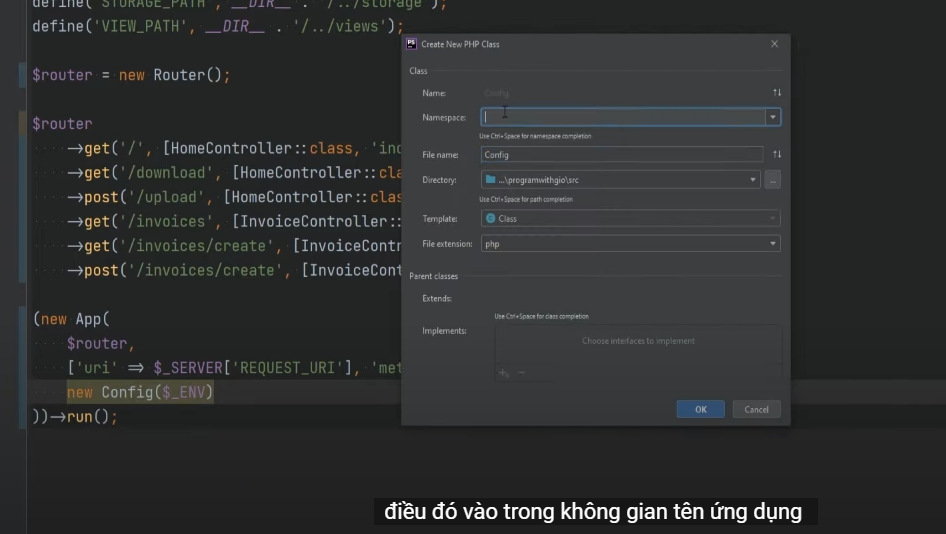
Vì vậy, mục tiêu của chúng ta bây giờ là có một kết nối cơ sở dữ liệu duy nhất và đối tượng kết nối cơ sở dữ liệu đó cần có sẵn trong toàn bộ ứng dụng. Có nhiều cách để đạt được kết nối cơ sở dữ liệu đơn, một trong số đó là sử dụng mẫu singleton mà tôi không thích vì lý do đã nêu ở phần trước của khóa học. Một lựa chọn khác là sử dụng dependency injection và truyền kết nối cơ sở dữ liệu thông qua hàm khởi tạo bất cứ nơi nào cần thiết, đó là một giải pháp tốt nhưng chúng ta chưa đề cập chi tiết về dependency injection với các container dependency injection và các thứ tương tự. Vì vậy, chúng ta sẽ không làm điều đó ngay bây giờ. Thay vào đó, chúng ta sẽ sử dụng các phương thức tĩnh.

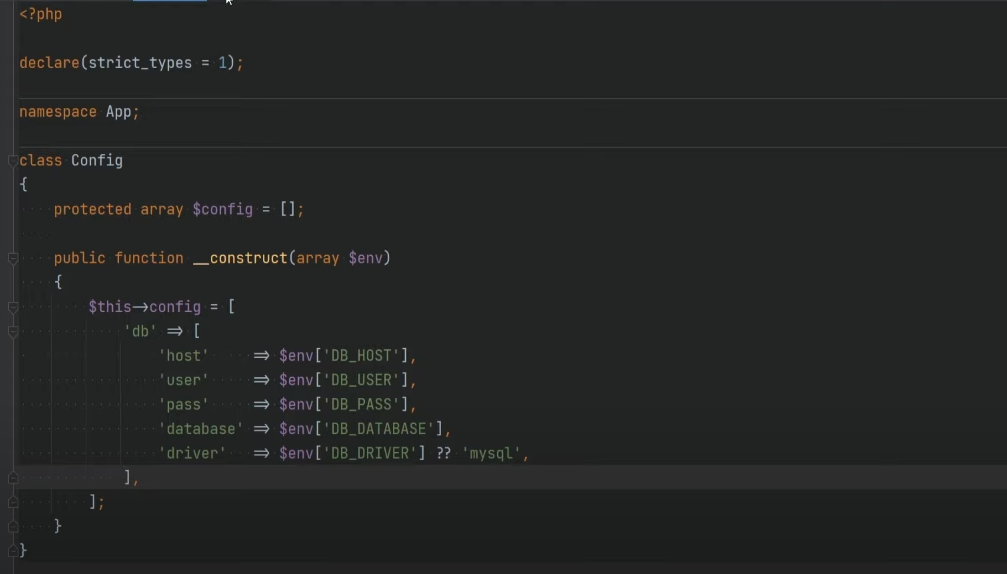
Vậy chúng ta có thể trích xuất phần code này đi đâu? Lựa chọn rõ ràng ở đây là file app.php vì như tôi đã đề cập, đây là lớp chính khởi động ứng dụng của chúng ta. Chúng ta có thể lưu trữ đối tượng kết nối cơ sở dữ liệu đơn trong lớp app đó.

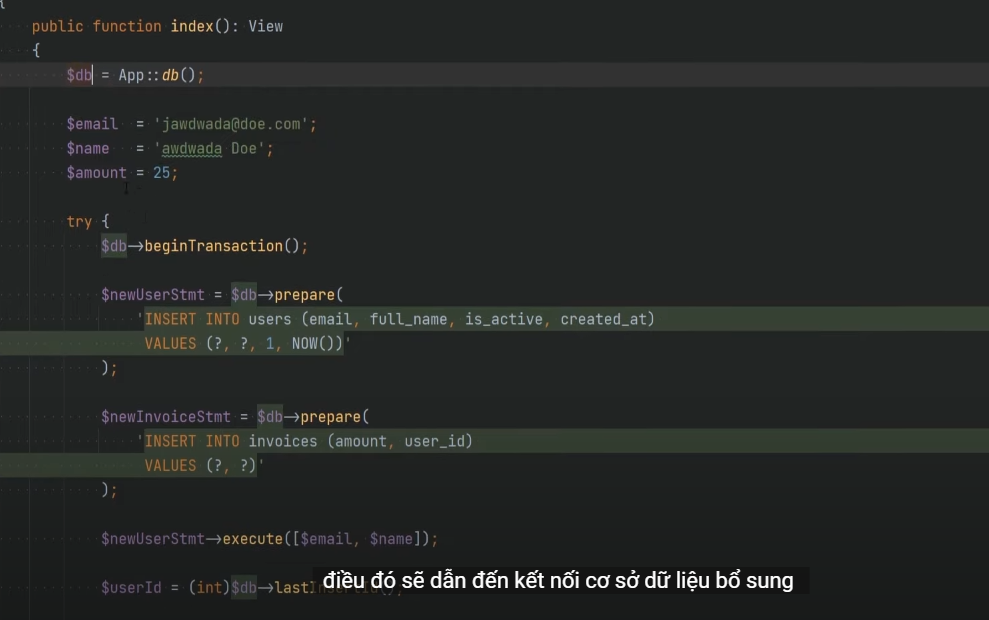
Chúng ta có thể cải thiện điều này thêm bằng cách tạo một lớp riêng biệt xử lý PDO thay vì sử dụng PDO trực tiếp trong lớp App. Chúng ta cũng có thể tạo một lớp Config riêng biệt để trích xuất tất cả dữ liệu cần thiết từ siêu toàn cục $\_ENV thay vì truy cập trực tiếp trong lớp App.



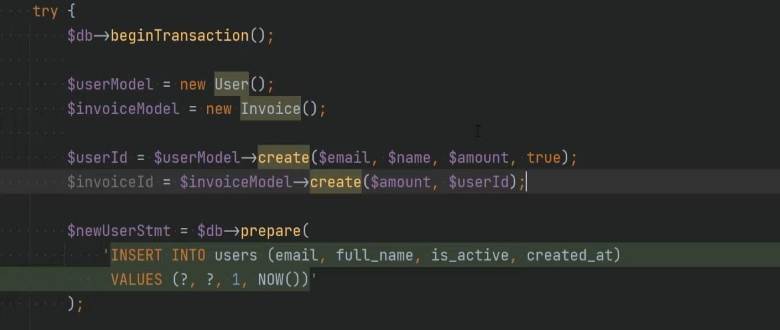


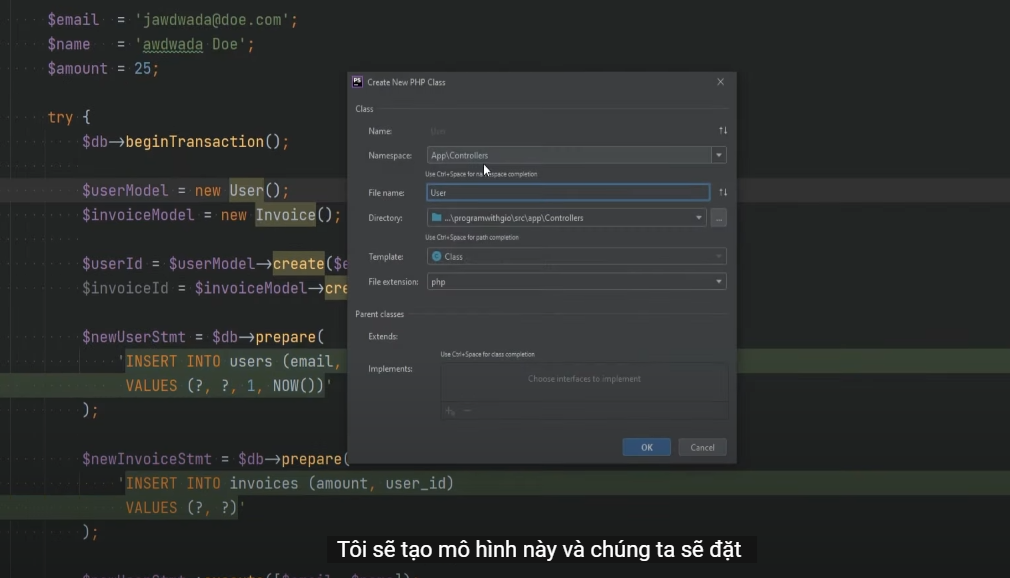


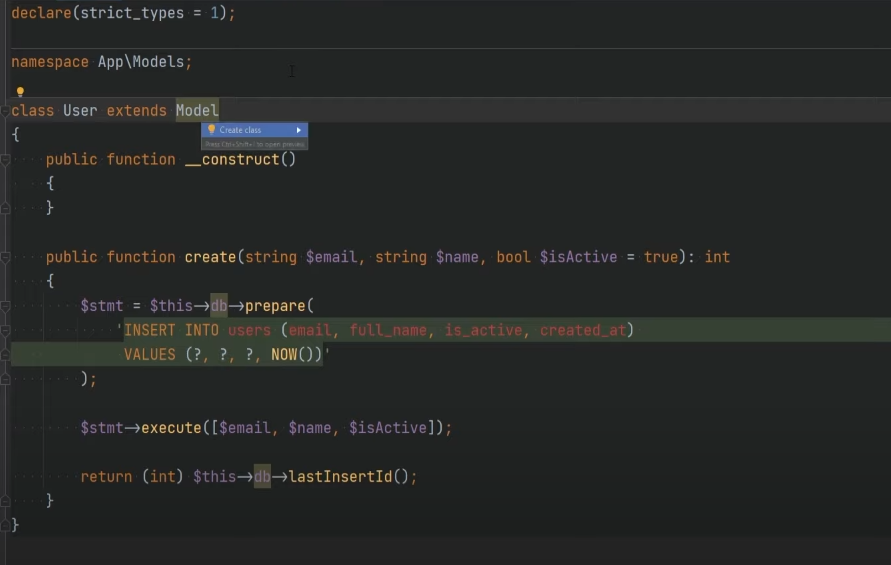


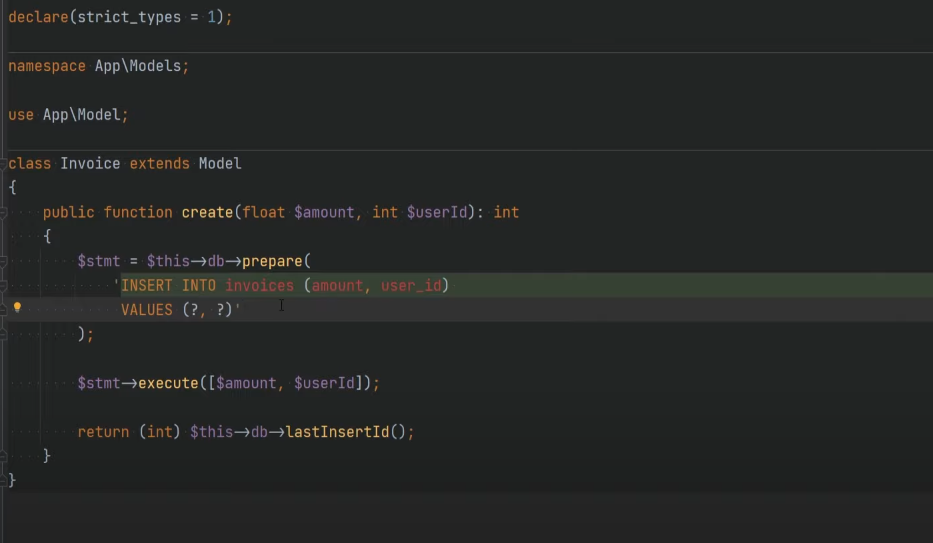


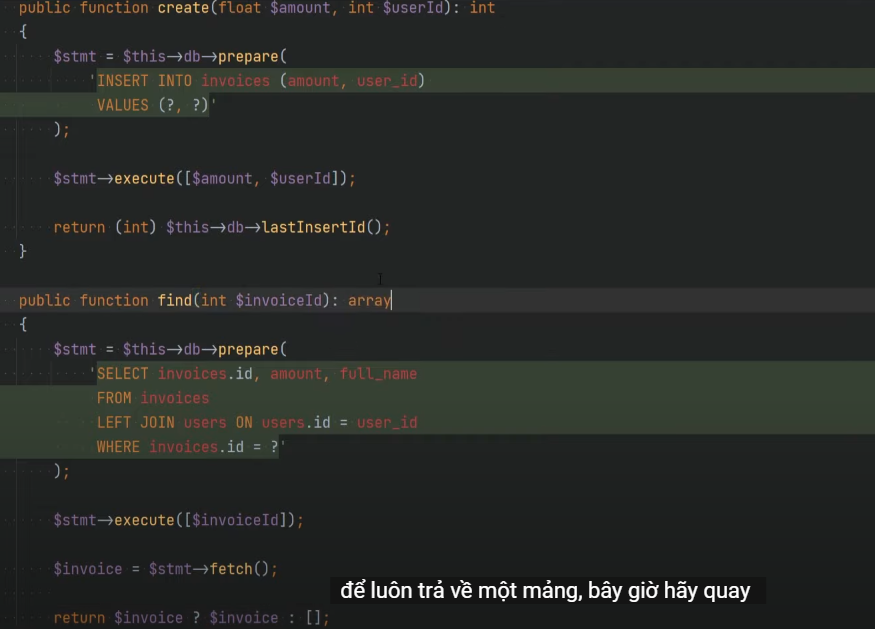
Bây giờ chúng ta cần ít nhất hai mô hình: một cho người dùng và một cho hóa đơn. Chúng ta có thể di chuyển phần lớn logic nghiệp vụ khỏi controller vào các mô hình này.



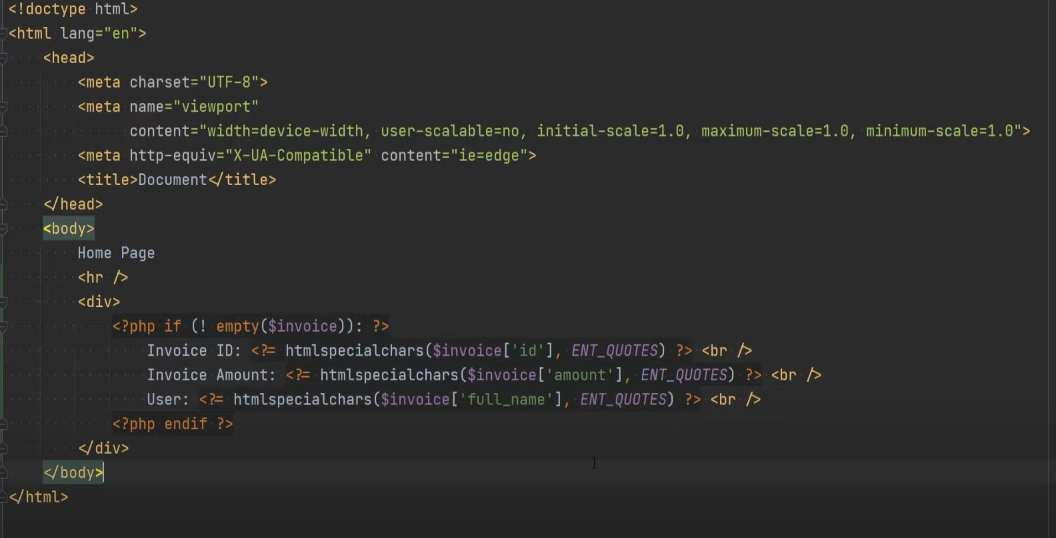








* Kết quả:
  + Các class được tách riêng chức năng, dễ quản lý và bảo trì hơn.
  + Home controller được tinh gọn, không còn làm nhiều việc như trước nữa.



Cuối cùng, controller của chúng ta trông sạch sẽ hơn nhiều. Trong view, chúng ta chỉ cần hiển thị một số thông tin về hóa đơn được truyền qua từ controller. Như vậy là hết cho video này, cảm ơn bạn đã xem. Hãy đánh giá video này nếu bạn thấy hữu ích, chia sẻ và đăng ký kênh của tôi để xem thêm nhiều video hơn.