Django ORM for Online Payment Systems?

2009

1 min read • 283 words

Themes: Programming

I've been spending an increasingly large amount of time with some rapid development frameworks, primarily Django (Python!), Grails (Groovy / Java), and Symfony (PHP)

> Kenneth's framework experimentation in 2009 reflects the polyglot programming movement of the era—before the current dominance of React and Node.js, developers explored diverse languages and paradigms to find the most productive development experience.

. I've been enjoying it. A lot. Life has never been better.

DRY tactics. Code portability. Who likes to repeat themselves anyway? It's a great idea.

My favorite concept to date is the Object Relational Model (ORM). Database-agnosticity is fantastic

> Kenneth's enthusiasm for ORMs and abstraction layers would become a defining characteristic of his library design philosophy—from Requests hiding HTTP complexity to Pipenv abstracting dependency management, he consistently sought to eliminate boilerplate.

. Not sure what database you want to use? Worry about it later. A client wants to switch to MySQL because SQLServer is costing too much? No problem. How much of my codebase will I have to change? About six characters. Wow.

So why not take this concept, and apply it elsewhere? I'm currently doing some work for a startup, and we are having trouble deciding which online payment service to use/support: PayPal, Amazon Payments, or Google Checkout.

My solution is to write a webPaySystem module that integrates all of these payment systems into one single class

> This early conceptualization of payment system abstraction anticipated the modern fintech API ecosystem—services like Stripe later succeeded by providing exactly this kind of unified, developer-friendly interface that Kenneth envisioned.

. But, before I spend the time to write this, I'd like to extend this question to the Python / Django community:

Would you find this useful in your web (and business desktop) applications?
