Why permissionless.js

A summary of why we built permissionless.js

Problems

ERC-4337 has emerged as a popular standard to tackle some of the most pressing user experience issues in the Ethereum ecosystem, but the current state of tooling in the space has been lacking. Namely, no library we have seen has been able to succeed in all of the following areas:obsessed with developer experience, built closely on top of existing tools, flexible, and vendor-agnostic .

AtPimlico we have seen a lot of developers using our infrastructure struggle with having to string many SDKs together. Instead of getting them to learn a whole new framework or building a@pimlico/sdk -like library that would lock the developer to our infrastructure, we decided to build a tool that would allow developers to build on top of APIs they're already familiar with with any provider they wish, mixing-and-matching all of the different parts of the stack including bundlers, paymasters, smart account providers, and signers.

Great developer experience

Developer experience is the cornerstone of permissionless.js's design. We're committed to providing strongly-typed TypeScript APIs, comprehensive documentation on every function, and an intuitive and legible style that doesn't require you to constantly jump around files to understand the code that has been written.

Built on top of existing tools

Developers should not have to learn a completely new framework to make their apps leverage smart accounts.viem andwagmi have established themselves as the most popular libraries for Ethereum development, so permissionless.js is built as a thin layer on top of them reusing as much of their standards and styles as possible. This means less work for us, and less work for developers who want to build with smart accounts.

Flexible and extensible

Account abstraction is still an evolving space, and we want to make sure that developers can easily build and extend their own solutions for different parts of the stack as they see fit. If we don't support a bundler, a paymaster, a smart account provider, or a signer that you want to use, it should be trivial to swap it out for another one that implements the same interface or to build and connect your own without having to rewrite your entire app.

Avoiding lock-in

Crypto developers don't like lock-in. In a world where the infrastructure provider whose propreitary SDK you've spent the last year building your app on can disappear overnight, you want to make sure that you can easily swap out different parts of the stack as you see fit, including that of the original creator of the library. If we as Pimlico want to see smart accounts succeed, we're going to have to build tooling that benefits the entire ecosystem, not just the users of our own infrastructure.