**Recommendations**

You have a few decisions to make on how to implement ERC-4337. Here's what we usually recommend.

[Suggest Edits](https://docs.stackup.sh/edit/recommendations)

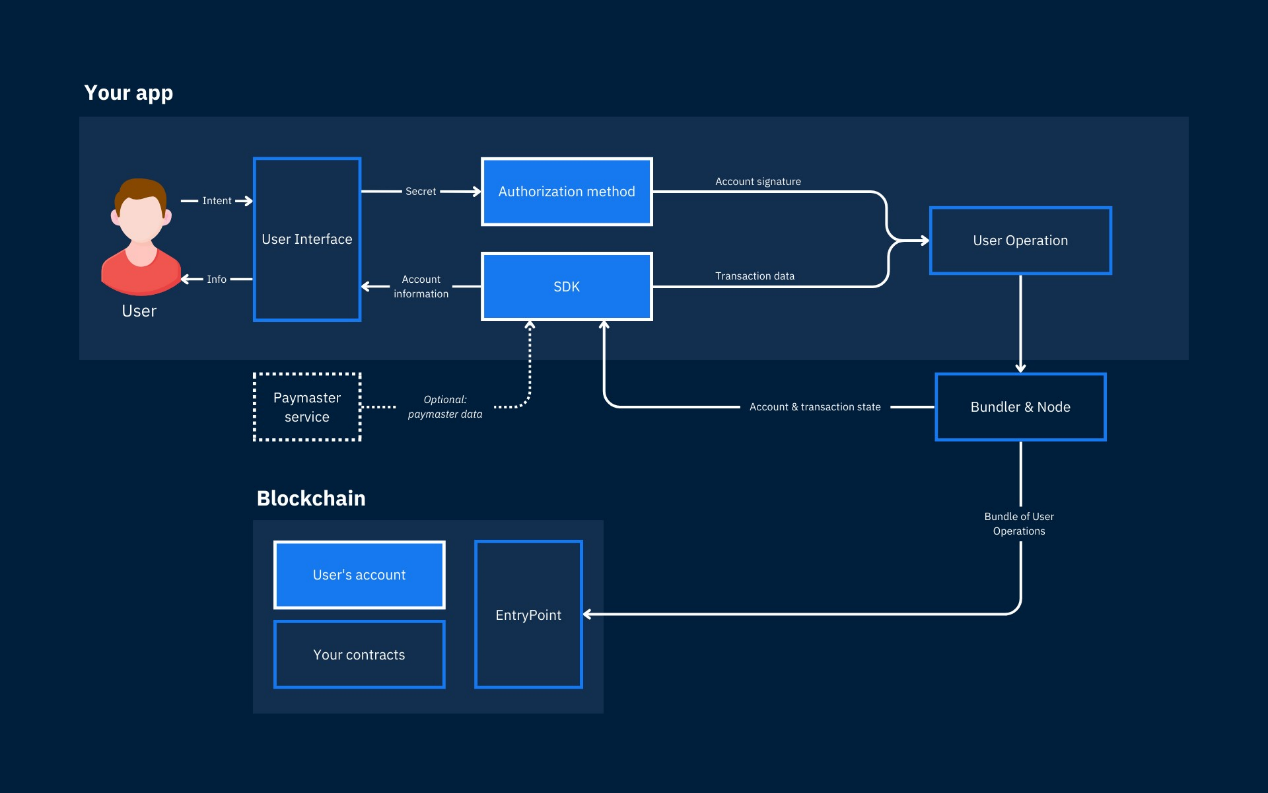
Although Stackup takes care of account infrastructure, you still have a number of decisions to make about which pieces you want to use if you are developing an application with ERC-4337.

These decisions can be overwhelming, so here's the advice we find ourselves giving 80% of projects.

👍

No endorsements

**We have no endorsement agreements with any third parties listed here.** These are based purely off of our experience developing applications with ERC-4337, and will change with time.



High-level interactions between an application and the blockchain. Blue boxes indicate major elements you will need to choose, and dashed lines are optional.

**Architectural Decisions**

Most apps will need to make three major decisions: how users will authorize transactions, what account contracts to give users, and what SDK to use to build user operations.

[You can see a template application that uses all of these recommendations here](https://docs.stackup.sh/docs/web3auth-nextjs-app)

**Decision 1: Authorization method**

We recommend using [Web3Auth](https://www.web3auth.io/)or similar MPC methods for basic applications. It has a plug-and-play modal that you can insert directly into your app, and users can always still connect with a traditional wallet.

More advanced authorization methods include session keys, biometrics, and device keys. You can explore some common authorization services [here](https://docs.stackup.sh/docs/erc-4337-key-management-tools).

**Decision 2: Smart account contracts**

We recommend using [ZeroDev's Kernel smart account](https://docs.stackup.sh/docs/erc-4337-examples-zerodev-kernel" \t "_self) for most applications. It is the most mature ERC-4337 Smart Account specific implementation, and includes modules for session keys and recovery methods.

It is also relatively future-proof. It is designed to comply with [ERC-6900](https://eips.ethereum.org/EIPS/eip-6900), a standard for making ERC-4337 smart accounts and their plugins interoperable.

See more options [here](https://docs.stackup.sh/docs/erc-4337-contract-account-libraries).

**Decision 3: SDK for building user operations**

We recommend using [userop.js](https://docs.stackup.sh/docs/useropjs) rather than using an ecosystem-dependent SDK for building transactions. Userop.js is more flexible, MIT open-source licensed, and likely underpins any application-specific SDK you are using.

See more options [here](https://docs.stackup.sh/docs/erc-4337-developer-sdks).

**Optional: Paymasters**

We recommend using a paymaster service rather than build one yourself. If you build one yourself, you need to manage on-chain liquidity.

We recommend the [Stackup Paymaster](https://docs.stackup.sh/docs/paymaster-api" \t "_self), not just because we built it, but because it is the most reliable and can be used to accept ERC-20 for gas payments or to sponsor transactions outright.

Regardless, in most instances it is better to use a paymaster that approves the gasless transaction off-chain than using one that is fully on-chain. This design allows a user to do any type of transaction without needing a blockchain's native token first - i.e., a user can start with just an ERC-20 token in their account.

If you use a paymaster to sponsor transactions for your app, we strongly recommend having a separate API key just for sponsoring transactions with your paymaster so that you can put business logic for which transactions you will accept server-side.

**Bonus: Reading user operation data**

ERC-4337 User Operations are routed through the EntryPoint contract, so they need to be parsed differently than regular transactions. We recommend using a data API that processes this data for you. Our [Data API](https://docs.stackup.sh/docs/data-api) does a lot of this parsing for you, but if you want more information you can use a service like [JiffyScan](https://app.jiffyscan.xyz/" \t "_self).

JiffyScan's block explorer is a bit slow, but has the most features.

Updated 22 days ago