
prize-worthy
an Ethereum Python hackathon guide
Marc Garreau
Developer, EF Python

intro + goals

- intros

marc, *wolovim*

github
github
github
github
github
github.com/ethereum/eth-utils
github.com/ethereum/py-trie
github.com/ethereum/py-geth
github.com/ethereum/eth-abi
github.com/ethereum/hexbytes
...

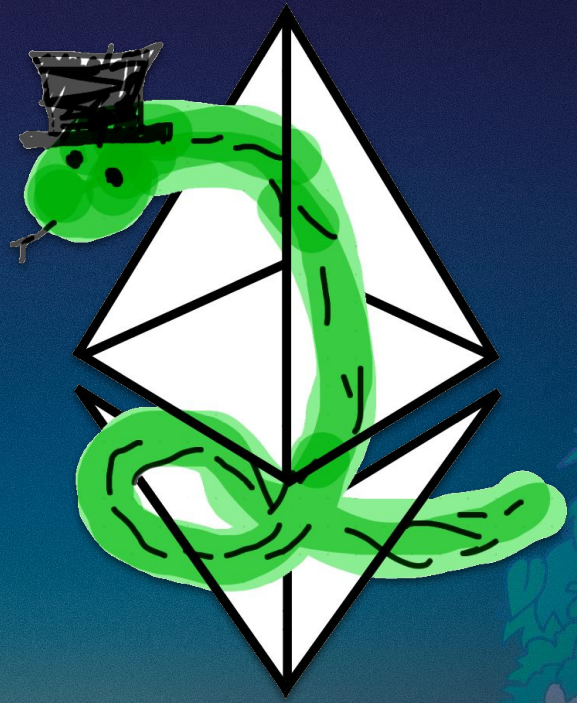
stu, *reedsa*

paul, *pacrob*

felipe, *fselmo*

intro + goals

- intros
- team “north star” goals
  - *aid in EIP implementation and testing*
  - *reduce UX friction*
  - ***educate and inspire builders***
- a hypothesis
- my goal for you



why hackath

- prizes, obviously
- mentorship
- meet potential employers and/or investors
- meet teammates, make friends, have fun
- grow your personal brand and credibility
- level up skills, learn new tech
- practice presenting your ideas
- find out if your idea has legs
- ...all within a week!
- ...in low risk environment!

tl;dr: career or company launchpad ✨🚀

game plan

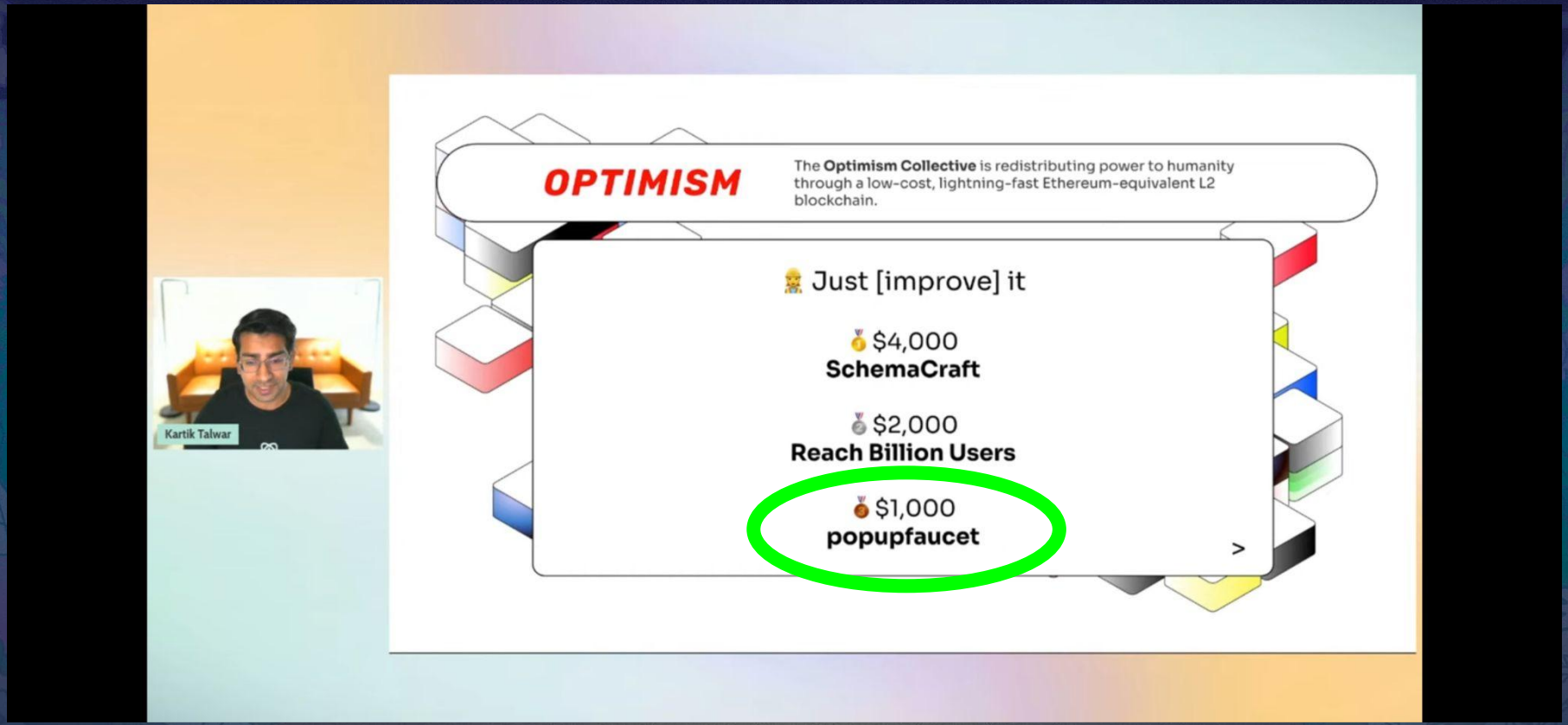
DappaDan
@DAppaDanDev

They call it a hackathon because you are literally hacking your way around a dark forest of deprecated code, unreleased features and out of date docs.

And honestly, I wouldn't have it any other way ❤️

12:44 PM · Mar 3, 2024 · **1,460** Views

game plan


Kartik Talwar
OPTIMISM
The Optimism Collective is redistributing power to humanity through a low-cost, lightning-fast Ethereum-equivalent L2 blockchain.
Just [improve] it
$4,000
SchemaCraft
$2,000
Reach Billion Users
$1,000
popupfaucet
>

chapter 0. **idea**

pro-tip: sign up before you're "ready"

- tons of participants start without an idea nailed down
- tl;dr: you'll figure one out or join someone else's team

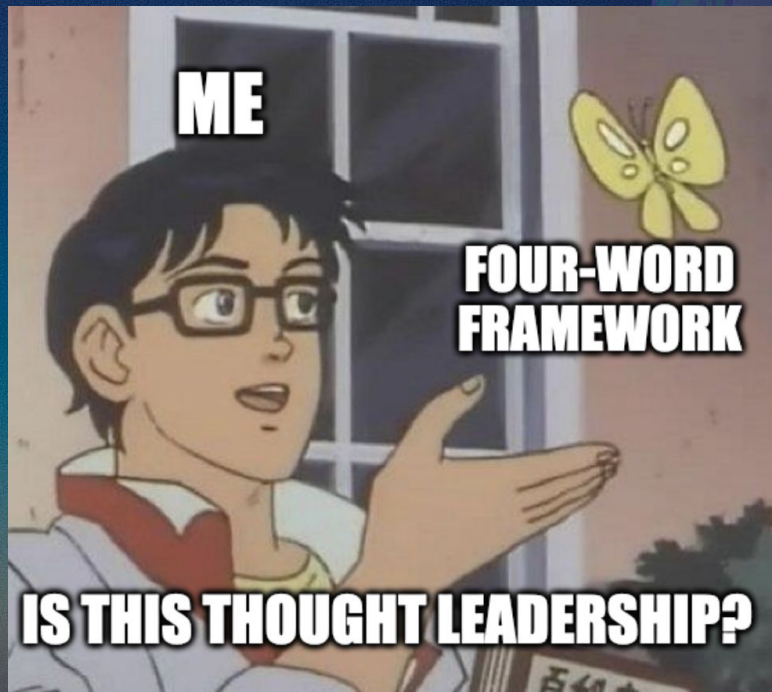
Q: how do i find an idea?

- categories: devtools, consumer apps, data projects
- mindset - be on the lookout for:
  - apps/tools you wish existed
  - pain points in your workflows
  - complaints you see on social media
- reference past winners, e.g., ETHGlobal project showcase
- join idea brainstorming/team formation events
- bounty lists from hackathon sponsors
- <what else?>

Q: how do i evaluate an idea?

1. **who** is it for?
2. **what** is their goal?
3. **how** do current options fall short?
4. **now**… (what solution can you offer?)

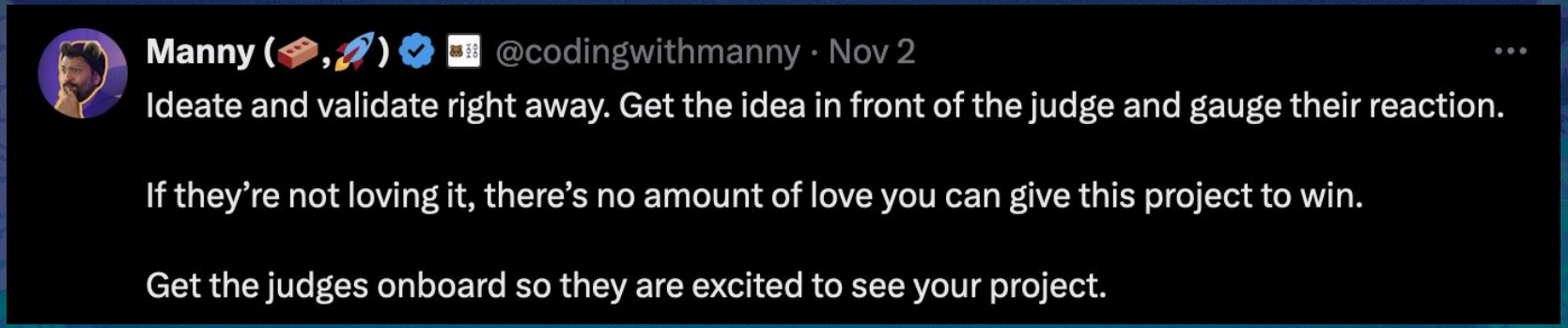
- *does the story you're telling make sense?*
  - *does the user exist?*
  - *does the problem exist?*
  - *is your solution any better than existing options?*



pro-tip: share your idea

- execution is everything
- feedback sharpens your idea
- rally support and/or team members

pro-tip: share your idea



Q: how do i evaluate an idea? (example)

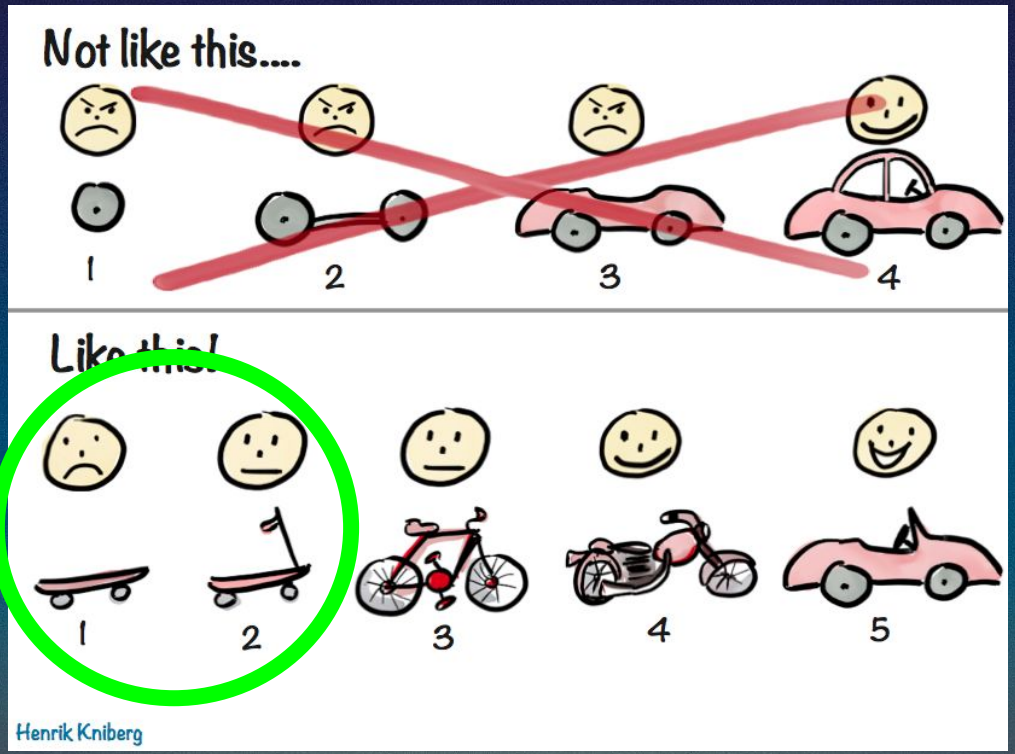
1. **who** is it for?
2. **what** is their goal?
3. **how** do current options fall short?
4. **now...** (what solution can you offer?)

Q: how do i evaluate an idea? (popupfaucet)

1. **who** is it for?
   a. workshop hosts
   b. workshop attendees
2. **what** is their goal?
   a. efficiently distribute testnet ether to attendees to enable participation
3. **how** do current options fall short?
   a. create and pre-seed accounts, print out and distribute QR codes
      i. cons: physical/in-person, manual process, doesn't scale well
   b. send attendees to 3rd-party faucets
      i. cons: external point of failure, sales pitches, account creation, daily limits
   c. collect addresses then use a script to distribute funds
      i. cons: manual process, late-arrival headaches
4. **now...** (what solution can you offer?)

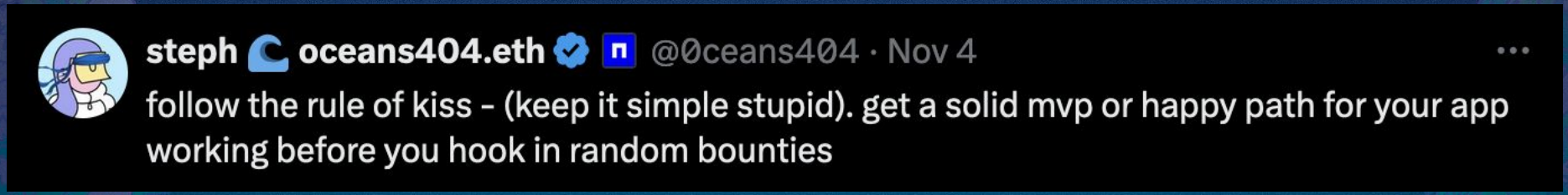
chapter 1. **dapp architecture**

pro-tip: think mvp



(obligatory, sorry)

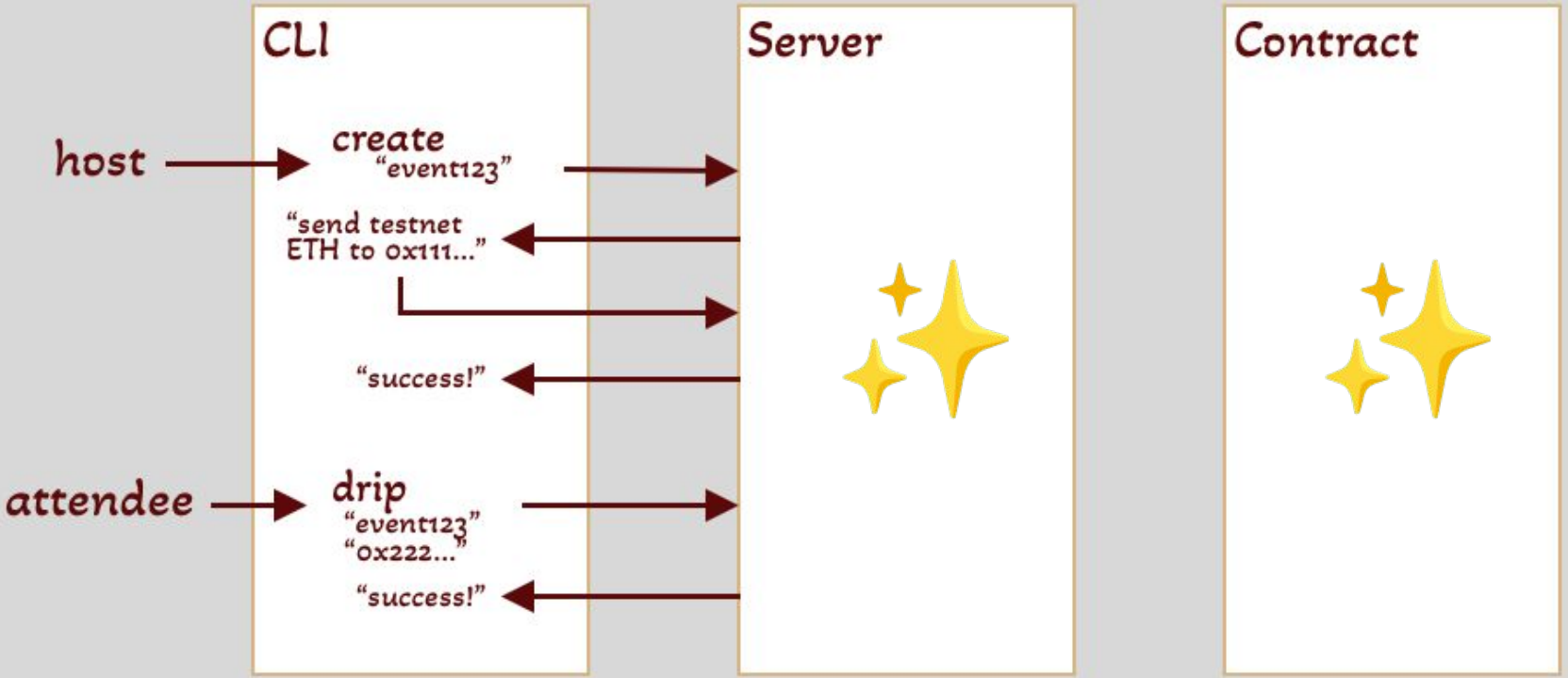
pro-tip: think mvp



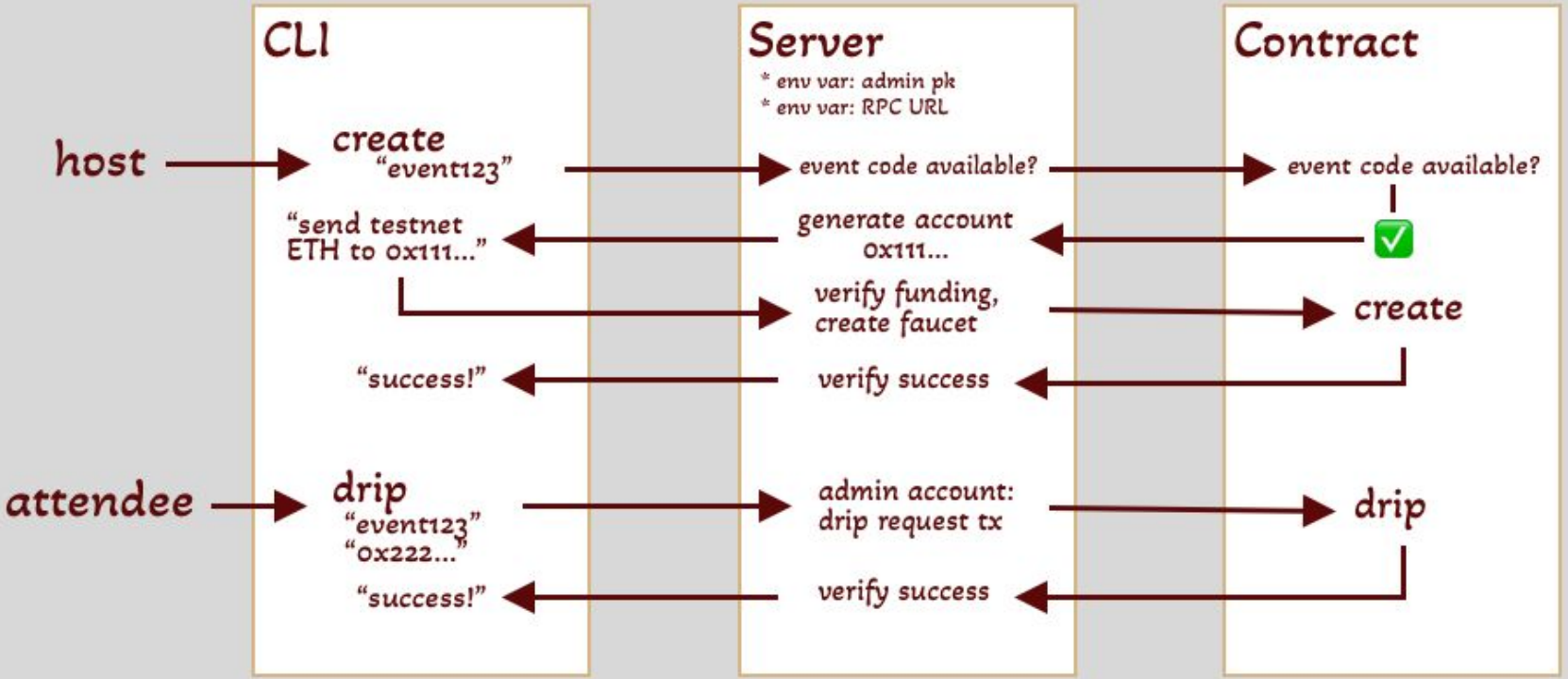
dapp architecture: mvp definition

- UI
  - least possible friction?
  - **host:** choose an event code + send funds to an address
  - **attendee:** specify event code + address you want funds sent to
  - mode: CLI; distribution: PyPI
  - *not included: duration, rate limiting, drip amount*
- Contract
  - one contract, arbitrary number of faucets
  - functions: create, drip
  - state: event codes and their available funds
  - *not included: security or gas efficiency audits*
- Server
  - contract interaction has to happen somewhere
  - secrets: admin private key, RPC URL
  - **note*: anyone can still use the contract without this server!

dapp architecture: mvp definition



dapp architecture: mvp definition

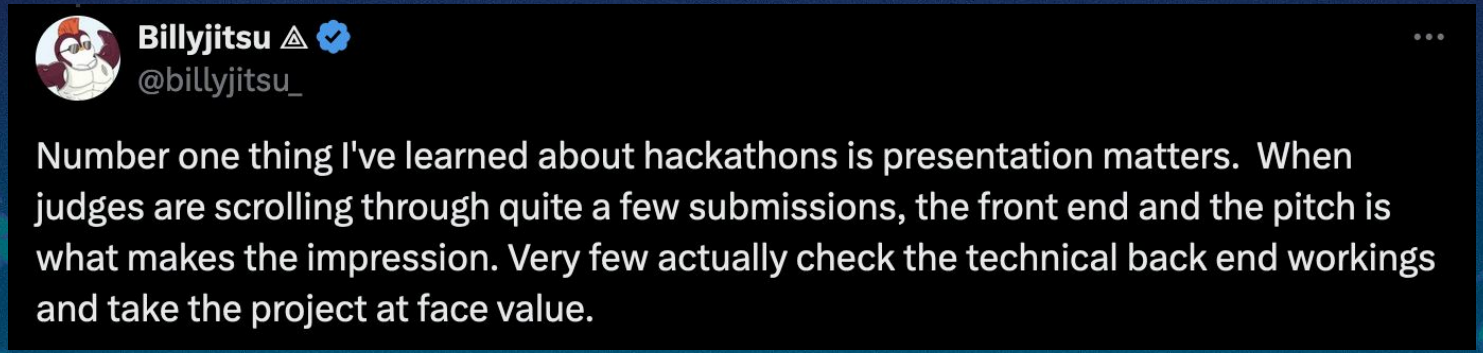


chapter 2. **contract**

pro-tip: don’t overthink stack choices

- (in the context of a hackathon)
- for consumer apps, most judges don’t care*
- tl;dr – be clear about your goals

pro-tip: don’t overthink stack choices



pro-tip: take shortcuts, use building blocks

- Solidity
  - OpenZeppelin contract wizard: https://wizard.openzeppelin.com/
  - Solady https://github.com/Vectorized/solady
  - Solmate https://github.com/transmissions11/solmate
  - https://solidity-by-example.org/
- Vyper
  - Snekmate https://github.com/pcaversaccio/snekmate
  - https://vyper-by-example.org/
- reference material
  - https://www.cookbook.dev/
  - https://github.com/marcelc63/popular-contract-templates
- prototyping editors
  - Remix: https://remix.ethereum.org/
  - sneko: https://github.com/wolovim/sneko 🐍

~ demo ~

- prerequisite: solidity or vyper fundamentals
- intro **sneko**, prototyping framework
  - textual
  - popupfaucet contract
  - local testnet
- intro **ape**, smart contract development framework
  - config
  - testing
  - scripting, e.g., deploying

~ demo ~

chapter 3. **contract interaction**

ch 3. game plan

1. brief tour of web3.py
2. highlight a few intermediate features
3. put the relevant concepts into practice

1. web3.py speedrun

web3.py speedrun

core concepts:

- provider config
- read
- write

web3.py speedrun

core concepts:

- **provider config**

```python
from web3 import Web3, HTTPProvider

w3 = Web3(HTTPProvider("https:// ... "))

w3.is_connected()
# True
```

```python
from web3 import AsyncWeb3, AsyncHTTPProvider

w3 = AsyncWeb3(AsyncHTTPProvider("https:// ... "))

await w3.is_connected
# True
```

web3.py speedrun

core concepts:

- **provider config**

```python
from web3 import Web3, HTTPProvider

RPC_URL = "https://<your-rpc>/mainnet"
RPC_URL = "https://<your-rpc>/sepolia"
RPC_URL = "https://<your-rpc>/arbitrum"
RPC_URL = "https://<your-rpc>/base-sepolia"

w3 = Web3(HTTPProvider(RPC_URL))

w3.is_connected()
# True
```

web3.py speedrun

core concepts:

- **provider config**

```python
from web3 import Web3, EthereumTesterProvider

w3 = Web3(EthereumTesterProvider())

w3.is_connected()
# True

acct = w3.eth.accounts[0]
w3.eth.get_balance(acct)
# 100000000000000000000000
```

web3.py speedrun

core concepts:

- **provider config**

```python
from web3 import AsyncWeb3, WebSocketProvider

async with AsyncWeb3(WebSocketProvider("wss//...")) as w3:
    await w3.is_connected()

    # eth_subscribe to new blocks
    await w3.eth.subscribe("newHeads")

    # parse blocks as they're added
    async for message in w3.socket.process_subscriptions():
        block = message["result"]
        print(f"Block: {block['number']}"}
        # Block: 21036998
```

web3.py speedrun

core concepts:

- provider config
- **read:**

web3.py speedrun

core concepts:

- provider config
- read:
  - **account balances**

```
w3.eth.get_balance('0×1a2b3c ... ')
# 2664118544717027 42
```

web3.py speedrun

core concepts:

- provider config
- read:
  - account balances
  - **block data**

```
w3.eth.get_block(15537393)
# AttributeDict({'baseFeePerGas': 48811794595, ...

w3.eth.get_transaction("0xabc123...")
# AttributeDict({'blockHash': HexBytes('0x56a...'), ...
```

web3.py speedrun

core concepts:

- provider config
- read:
  - account balances
  - block data
  - **transaction data**

web3.py speedrun

core concepts:

- provider config
- read:
  - account balances
  - block data
  - transaction data
  - **from contract**

```
example_contract = w3.eth.contract(
    abi=...,
    address=...
)
example_contract.functions.mult(4,5).call()
# 20
```

web3.py speedrun

core concepts:

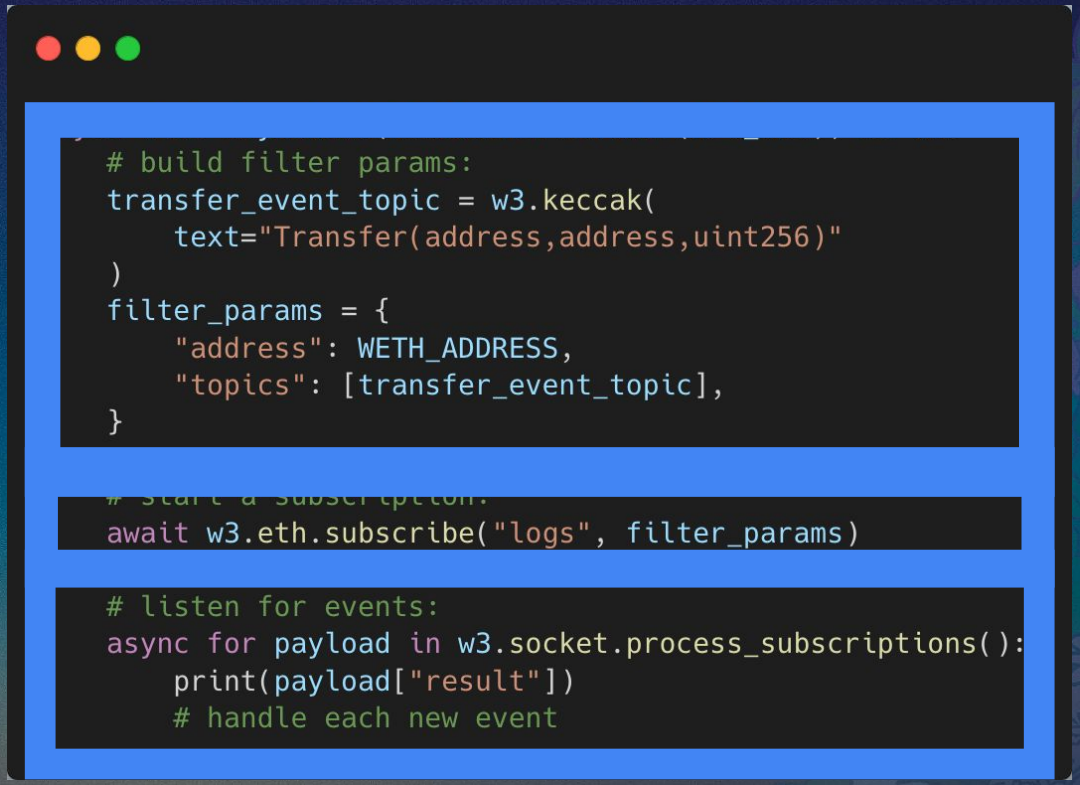
- provider config
- read:
  - account balances
  - block data
  - transaction data
  - from contract
  - **event logs**

```
my_contract = w3.eth.contract(address=ADDRESS, abi=ABI)

my_contract.events.myEvent().get_logs(
    argument_filters={"eventArg1": [1, 2, 3]},
    from_block=1337,
    to_block=1437,
)
```

web3.py speedrun

core concepts:

- provider config
- read:
  - account balances
  - block data
  - transaction data
  - from contract
  - **event logs**



```
# build filter params:
transfer_event_topic = w3.keccak(
    text="Transfer(address,address,uint256)"
)
filter_params = {
    "address": WETH_ADDRESS,
    "topics": [transfer_event_topic],
}
```

```
await w3.eth.subscribe("logs", filter_params)
```

```
# listen for events:
async for payload in w3.socket.process_subscriptions():
    print(payload["result"])
    # handle each new event
```

web3.py speedrun

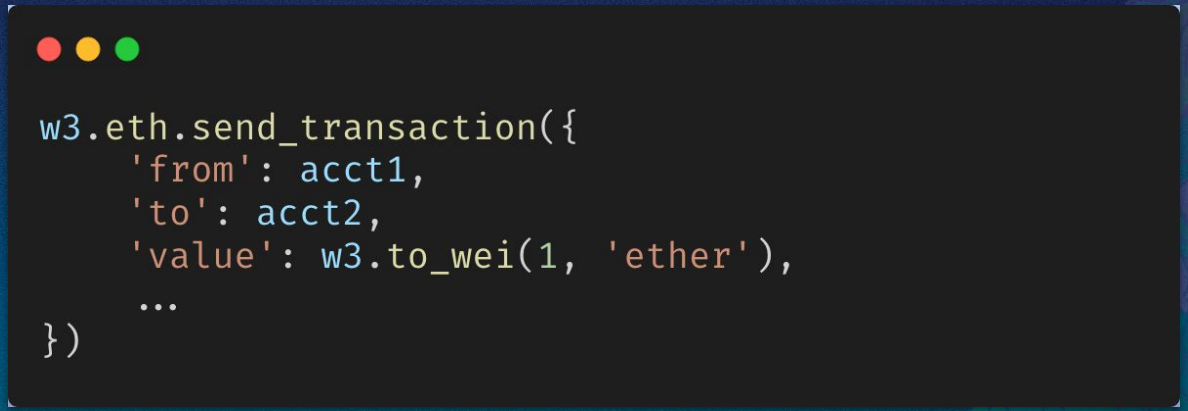
core concepts:

- provider config
- read:
  - account balances
  - block data
  - transaction data
  - from contract
  - event logs
- **write:**

web3.py speedrun

core concepts:

- provider config
- read:
  - account balances
  - block data
  - transaction data
  - from contract
  - event logs
- write:
  - **transfer ether**



```
w3.eth.send_transaction({
    'from': acct1,
    'to': acct2,
    'value': w3.to_wei(1, 'ether'),
    ...
})
```

web3.py speedrun

core concepts:

- provider config
- read:
  - account balances
  - block data
  - transaction data
  - from contract
  - event logs
- write:
  - transfer ether
  - **write to contract**

```
example_contract = w3.eth.contract(
    abi=...,
    address=...
)
example_contract.functions.updateThing().transact()
```

web3.py speedrun

core concepts:

- provider config
- read:
  - account balances
  - block data
  - transaction data
  - from contract
  - event logs
- write:
  - transfer ether
  - write to contract
  - **deploy**

```
ExampleContract = w3.eth.contract(
    abi=...,
    bytecode=...
)
ExampleContract.constructor().transact()
```

2. web3.py, bonus round

web3.py speedrun

web3.py fine-tuning:

- middleware
- custom methods
- custom modules
- custom providers
- caching, retries, etc.
- convenience config,
  e.g., default signer account

web3.py speedrun

notable addons:

- **ENS support**
- eth-account
- eth-utils

```
w3.ens.address('ethereum.eth')
# '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'

w3.ens.name('0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045')
# 'vitalik.eth'

w3.ens.get_text('nick.eth', 'url')
# 'https://ens.domains/'
```

web3.py speedrun

notable addons:

- ENS support
- **eth-account**
- eth-utils

```python
from eth_account import Account

# Account creation:
account = Account.create()

# Account management:
account = Account.from_key('0xabc...')
account = Account.from_mnemonic("abandon agent cactus...")
pk = Account.decrypt(keyfile_json, password)

# Transaction signing:
Account.sign_transaction(tx, pk)

# EIP-712 message signing:
signed_message = Account.sign_typed_data(
    pk,
    domain_data,
    message_types,
    message_data
)
```

web3.py speedrun

notable addons:

- ENS support
- eth-account
- **eth-utils**

Utilities

ABI Utilities

```
abi_to_signature()
collapse_if_tuple()
event_abi_to_log_topic()
event_signature_to_log_topic()
filter_abi_by_name()
filter_abi_by_type()
function_abi_to_4byte_selector()
function_signature_to_4byte_selector()
get_abi_input_names()
get_abi_input_types()
get_abi_output_names()
get_abi_output_types()
get_aligned_abi_inputs()
get_all_event_abis()
get_all_function_abis()
get_normalized_abi_inputs()
```

web3.py speedrun

notable addons:

- ENS support
- eth-account
- **eth-utils**

`humanize_wei(int)` -> string

Returns a human-friendly form of units given an amount of wei.

```
>>> from eth_utils import humanize_wei
>>> humanize_wei(0)
'0 wei'
>>> humanize_wei(1000000000000000000)
'1000 ether'
>>> humanize_wei(9876543)
'0.009876543 gwei'
```

3. apply the concepts

server

```python
from flask import Flask, request, jsonify

app = Flask(__name__)
load_dotenv()


@app.route("/status", methods=["GET"])
def status():
  ...

@app.route("/create", methods=["POST"])
def create():
  ...

@app.route("/drip", methods=["POST"])
def drip():
  ...
```

server: setup

```
# securely stored .env variables:
ADMIN_PK = os.getenv("POPUPFAUCET_ADMIN_PK")
RPC_URL = os.getenv("OP_SEPOLIA_URL"))
```

```
# load contract compilation artifacts:
with open("artifacts.json") as f:
    artifacts = json.load(f)
```

```
w3 = Web3(HTTPProvider(RPC_URL)
contract = w3.eth.contract(
    address=CONTRACT_ADDRESS,
    abi=artifacts["abi"],
)
```

server: multi-network support

```
# init Web3:
w3_op_sepolia = Web3(HTTPProvider(os.getenv("OP_SEPOLIA_URL")))
w3_base_sepolia = Web3(HTTPProvider(os.getenv("BASE_SEPOLIA_URL")))
```

```
o_contract = w3_op_sepolia.eth.contract(address=ADDRESS, abi=ABI)
b_contract = w3_base_sepolia.eth.contract(address=ADDRESS, abi=ABI)
```

```
    "OP Sepolia": {"w3": w3_op_sepolia, "contract": o_contract},
    "Base Sepolia": {"w3": w3_base_sepolia, "contract": b_contract},
    "Sepolia": {"w3": w3_sepolia, "contract": s_contract},
}
```

```
def get_w3_and_contract(network):
    return networks[network]["w3"], networks[network]["contract"]

# usage:
w3, contract = get_w3_and_contract("Sepolia")

# w3.eth.get_block("latest")
# contract.functions.codeAvailable("devcon").call()
```

server: contract reads

```python
@app.route("/availability", methods=["GET"])
def check_availability():
    event_code = request.args.get("event_code")
    network = request.args.get("network")
    w3, contract = get_w3_and_contract(network)

    t
        is_available = contract.functions.codeAvailable(event_code).call()
                                                                    e}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

server: contract writes

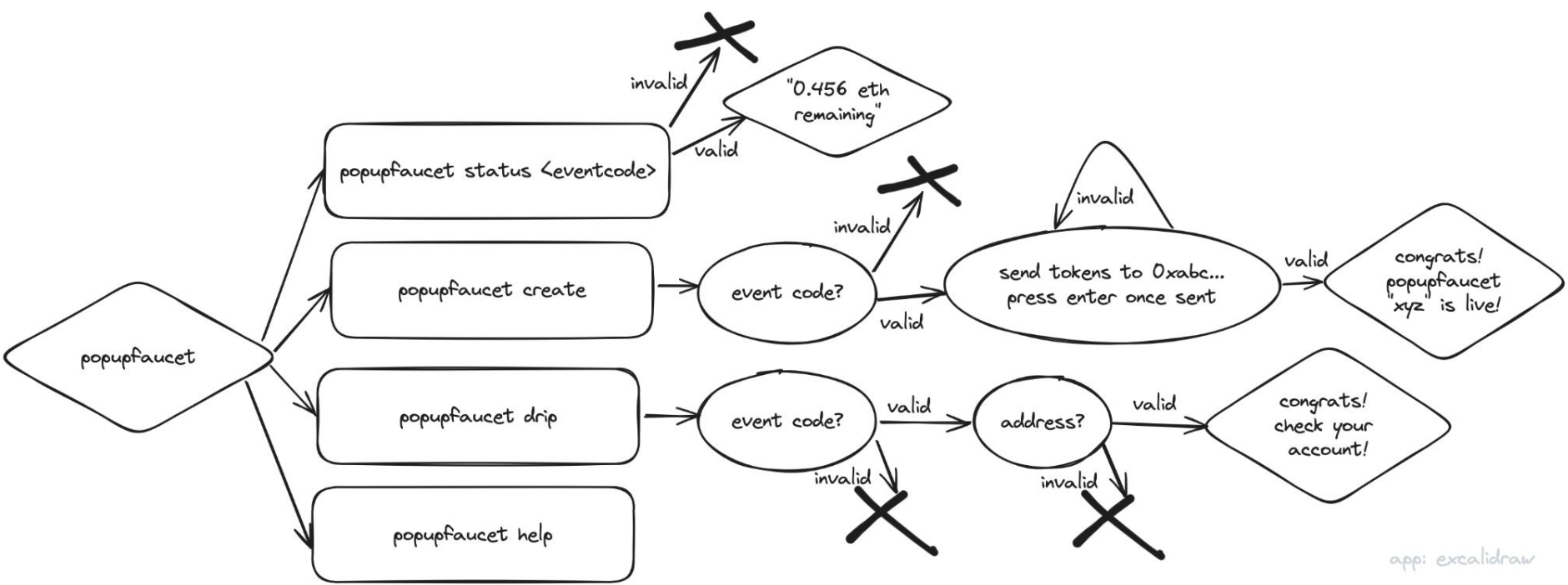
```python
@app.route("/drip", methods=["POST"])
def drip():
    ...
    w3, contract = get_w3_and_contract(network)

    t
        # build the transaction
        tx = contract.functions.drip(address, event_code).build_transaction(
            {"nonce": w3.eth.get_transaction_count(admin_account.address)}
        )

        # sign the transaction
        signed_tx = w3.eth.account.sign_transaction(
            tx, private_key=admin_account.key
        )

        # send the transaction & wait for receipt
        tx_hash = w3.eth.send_raw_transaction(signed_tx.raw_transaction)
        tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
        receipt_hash = tx_receipt["transactionHash"].to_0x_hex()
        return jsonify({"tx_hash": receipt_hash}), 200

    ex
        return jsonify({"error": str(e)}), 500
```

chapter 4. **UI**

Python UI options

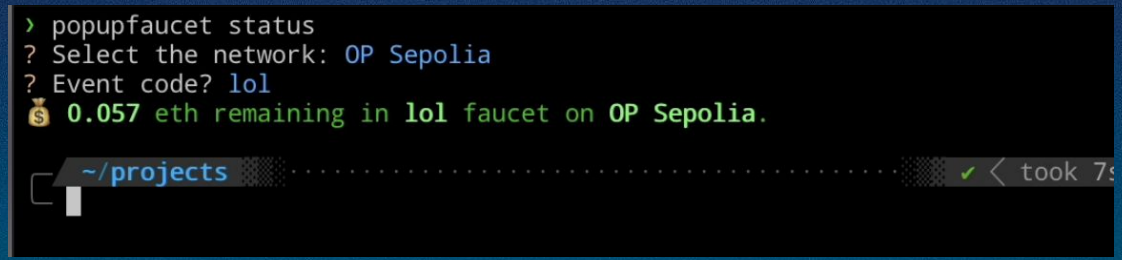
- CLI: click, rich
- TUI: textual
- bot (Discord, Telegram, etc.)
- Jupyter notebooks
- data viz tools
- webapp (via Flask, Django, FastAPI, etc.)
- web3.py (via custom module)
- distribution:
  - web/mobile app
  - PyPI package
  - chat app
  - NYC billboard

pro-tip: sketch user experience

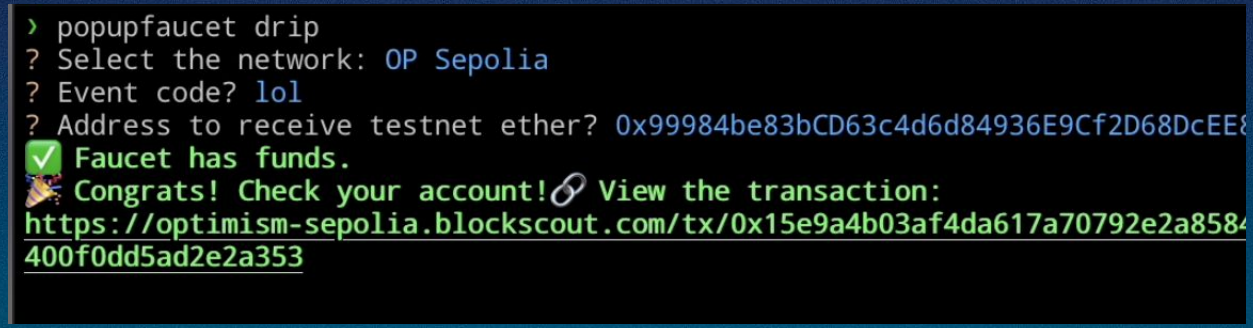


UI: faucet creation

UI: faucet status



UI: faucet drip



UI: ephemeral accounts

```
from eth_account import Account

@popupfaucet.command()
def create():
    answers = prompt([...])
    network = answers["network"]
    event_code = answers["event_code"]

    # confirm event code available:
    with console.status("Checking availability...", spinner="moon"):
        response = session.get(f"{SERVER_URL}/availability", params={...})
        is_available = response.json().get("is_available")
        if not is_available:
            console.print("❌")
            return
        else:
            console.print("✅")

    # use an ephemeral account:
    acct = Account.create()
    console.print(f"Send {network} testnet ether to: {acct.address}")
    input()

    # to be continued...
```

UI: ephemeral accounts

```
    # every 5 seconds, check if ephemeral account was funded:
    with console.status("Waiting for confirmation...", spinner="moon"):
        payload = {"network": network, "address": acct.address}
        waiting_for_confirmation = True
        while waiting_for_confirmation:
            response = requests.post(f"{SERVER_URL}/seeder-funded", json=payload)
            if response.status_code == 200:
                console.print(f"✅ Account funded!")
                waiting_for_confirmation = False
            else:
                console.print("❌ ")
                sleep(5)

    # initiate the contract write:
    with console.status("Deploying faucet...", spinner="moon"):
        payload = {"event_code": event_code, "network": network, "pk": acct.key.hex()}
        response = requests.post(f"{SERVER_URL}/create-faucet", json=payload)
        if response.status_code == 200:
            console.print("🎉 Congrats! Your popupfaucet is live!")
        else:
            console.print("❌ ")
```

last code sample 😤

chapter 5. **presentation**

presentation

- ETHGlobal: 2-4 minute recorded video
- template: **who, what, how, now…**
  - “We’re solving a problem for workshop hosts that want to…”
- factor in: script-writing, slide creation, rehearsal, redos
- **pro-tip:** start max. 75% of the way through your timeline.

presentation

tonyolendo.eth
@tonyolendo

Do the video and presentation first. Less is more, focus on the bounties that best line up with your skills. Talk to the product people on-site about what you're building before the deadline, they should be looking out for you when reviewing submissions. Have fun and experiment.

tonyolendo.eth
@tonyolendo

Immediately after you settle on an MVP, use a mockup

chapter 6. **polish**

polish

once MVP is done, iterate:

- product
  - chase another bounty
  - styling, graphics, logo
  - context and usage instructions
  - loading experiences
  - error messages
  - testing edge cases
- presentation
  - polish script
  - improve the slides or demo
  - rehearse
  - re-record video

✨ good luck at ETHGlobal Bangkok! ✨

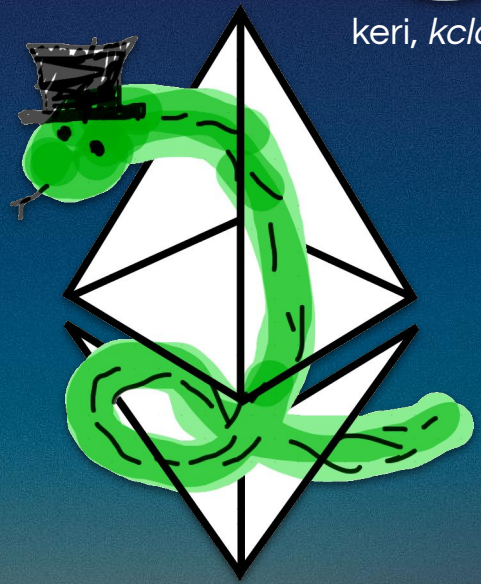
⚡ open forum ⚡

- share a hackathon pro-tip
- pitch an idea via the *“who, what, how, now”* framework
- intro yourself, looking for a team
- intro your team/idea, looking for members
- other questions?

thanks! ✨

@wolovim
@EthereumPython

marc, *wolovim*



keri, *kclowes*

stu, *reedsa*

paul, *pacrob*

felipe, *fselmo*

workshop resources: