TEAM 4

ETH BROWNIE

The Hitchhiker guide to the MOON and BEYOND!!

POINTS TO DISCUSS

Let's talk about this yummy brown thing.

01

WHAT IS BROWNIE?

02

BROWNIE'S ARCHITECTURE

03

QUALITY ATTRIBUTES

04

BROWNIE'S DESIGN PATTERNS

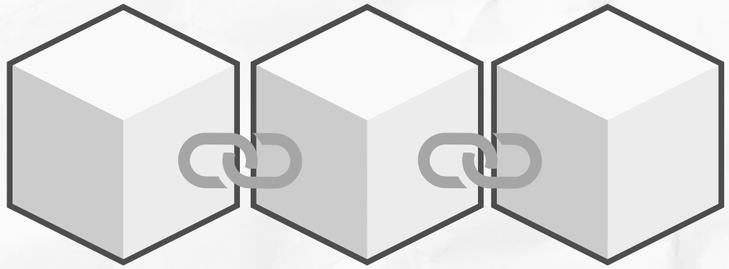
01

WHAT IS BROWNIE?

WHAT IS ETHEREUM?

Ethereum is a decentralized, open-source blockchain with **smart contract** functionality. Ether (ETH or Ξ) is the native cryptocurrency of the platform.

Ethereum provides a virtual environment to execute code on and we'll called it **Ethereum Virtual Machine (EVM).**



WHAT IS BROWNIE?

Brownie is a Python-based development and testing framework for smart contracts targeting the **Ethereum Virtual Machine.**

WHAT BROWNIE SUPPORT?

List of EVM Networks

Ethereum Mainnet
Ethereum Testnet (ROP, RIN, GOR, KOV)
Expanse Network
Ethereum Classic
Ubiq Network
Metadium Mainnet / Testnet
ThaiChain
ELA-ETH Sidechain
RSK Network Mainnet / Testnet
GoodData
Telos EVM Mainnet
Telos EVM Testnet
XinFin
CoinEx Smart Chain
Zyx Mainnet
Binance Smart Chain
Ontology Mainnet

BitTorrent Chain
Arbitrum One
Permission
Energy Web Chain
Fantom Opera
Boba Network
KCC Mainnet / Testnet
Shiden
Theta Mainnet / Testnet
PulseChain Mainnet
Rupaya
Tao Network
cheapETH
Callisto Mainnet / Testnet
Acala Network
Wanchain Mainnet / Testnet
Ambros Chain
PulseChain
Lucky Network

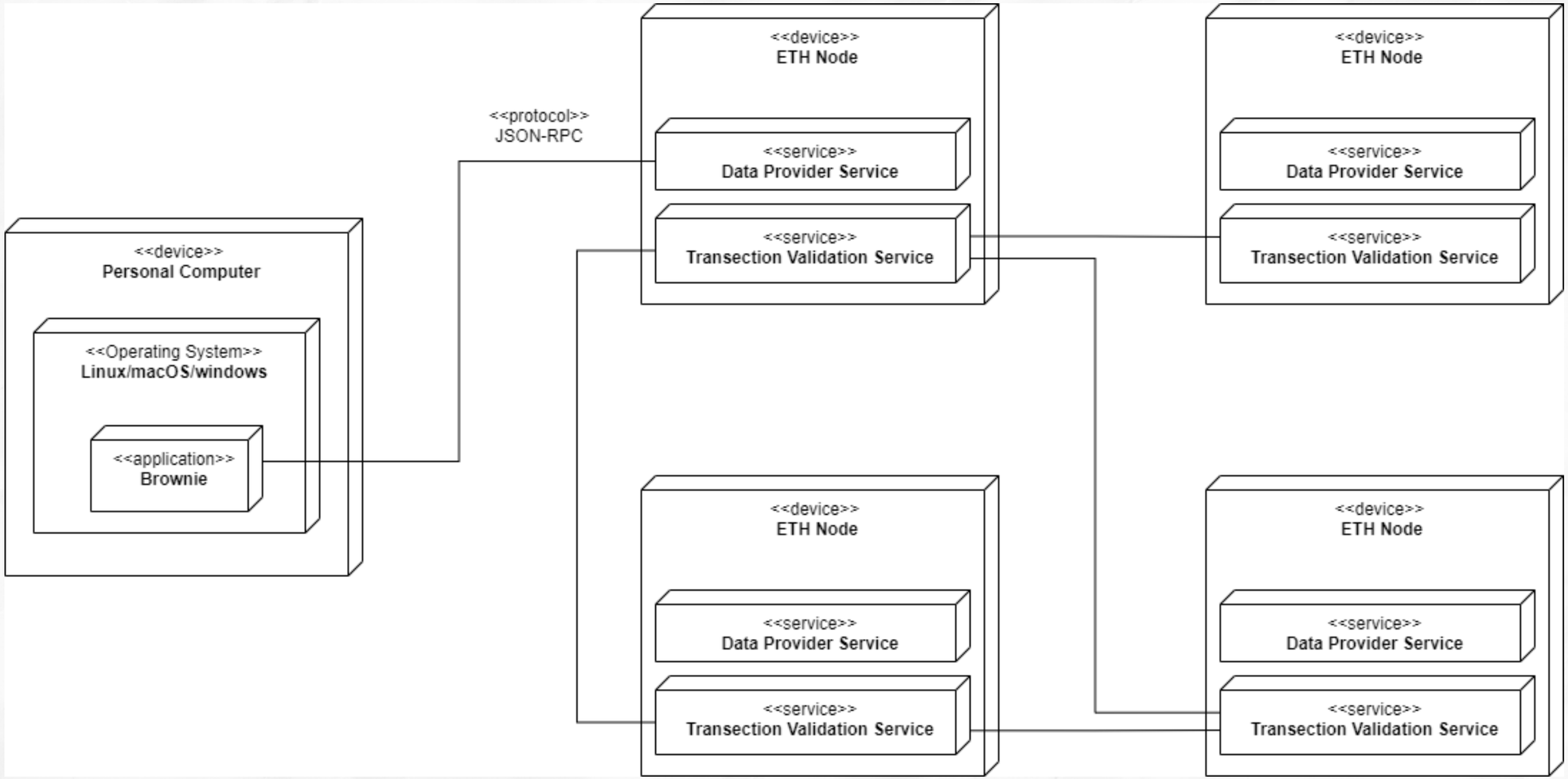
EraSwap
Smart Bitcoin Cash
MetaDot
Ethersocial Network
Celo Mainnet
Athereum
Avalanche Mainnet / Fuji Testnet
Polyjuice
QuarkChain Mainnet / Devnet
Akroma
ARTIS
Polis Mainnet / Testnet
Etho Protocol
Xerom
Auxilium Network
Aquachain
Joys Digital Mainnet
Neon EVM

And the list goes on...

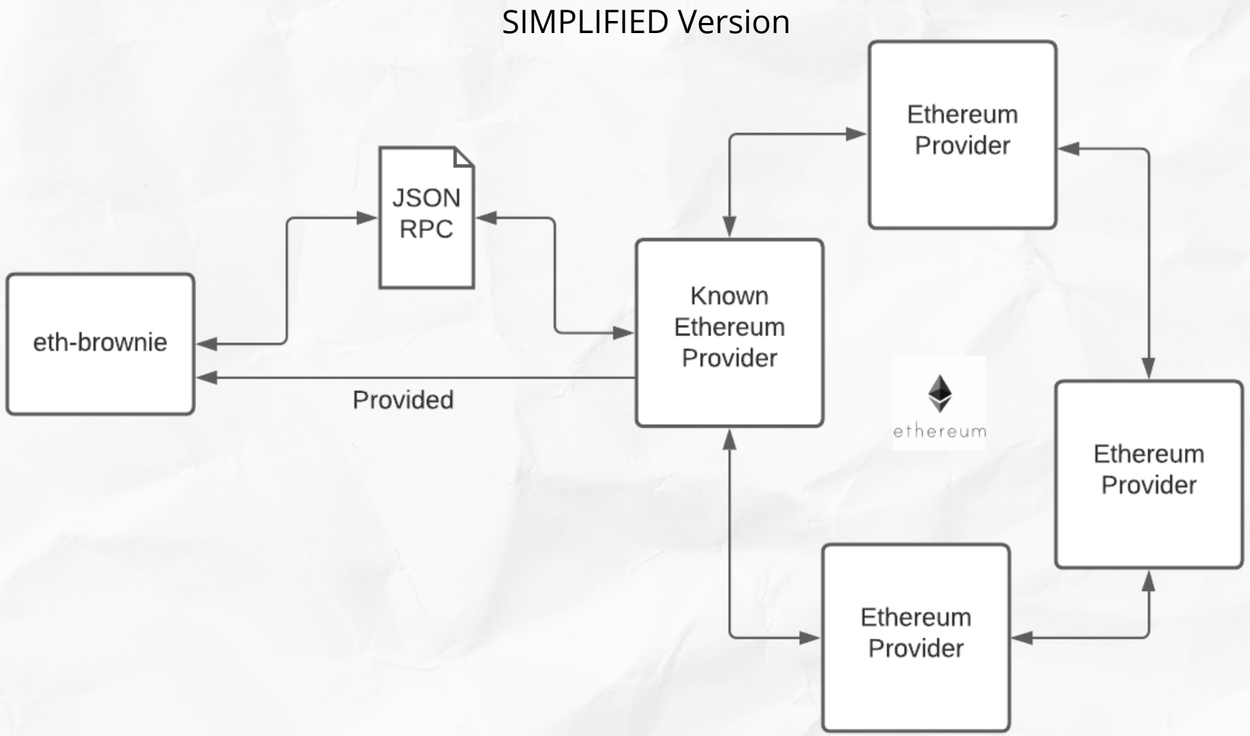
02

BROWNIE'S ARCHITECTURE

CLIENT-SERVER



CLIENT-SERVER

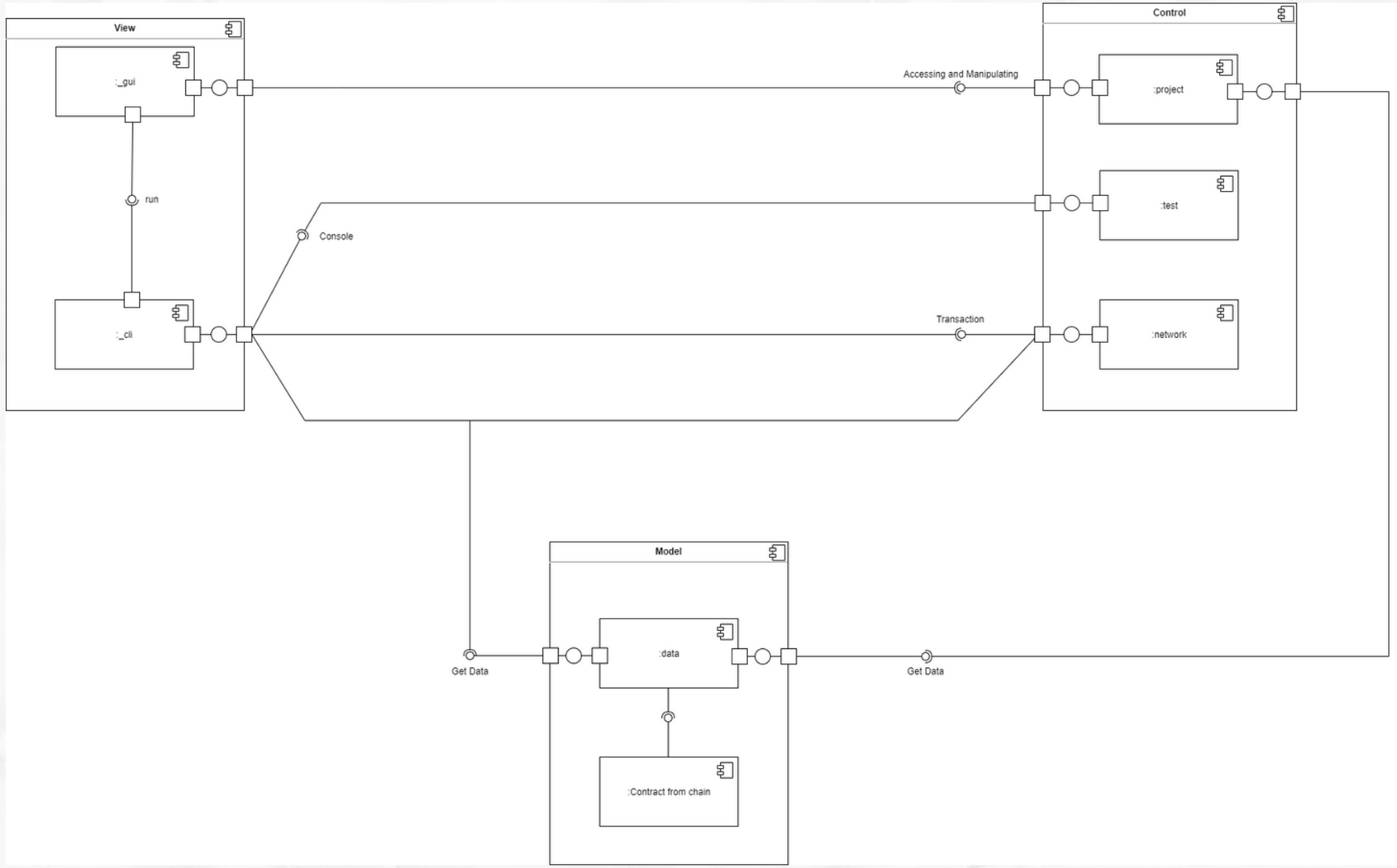


EXPLAIN

eth-brownie architecture can be viewed as a client-server.

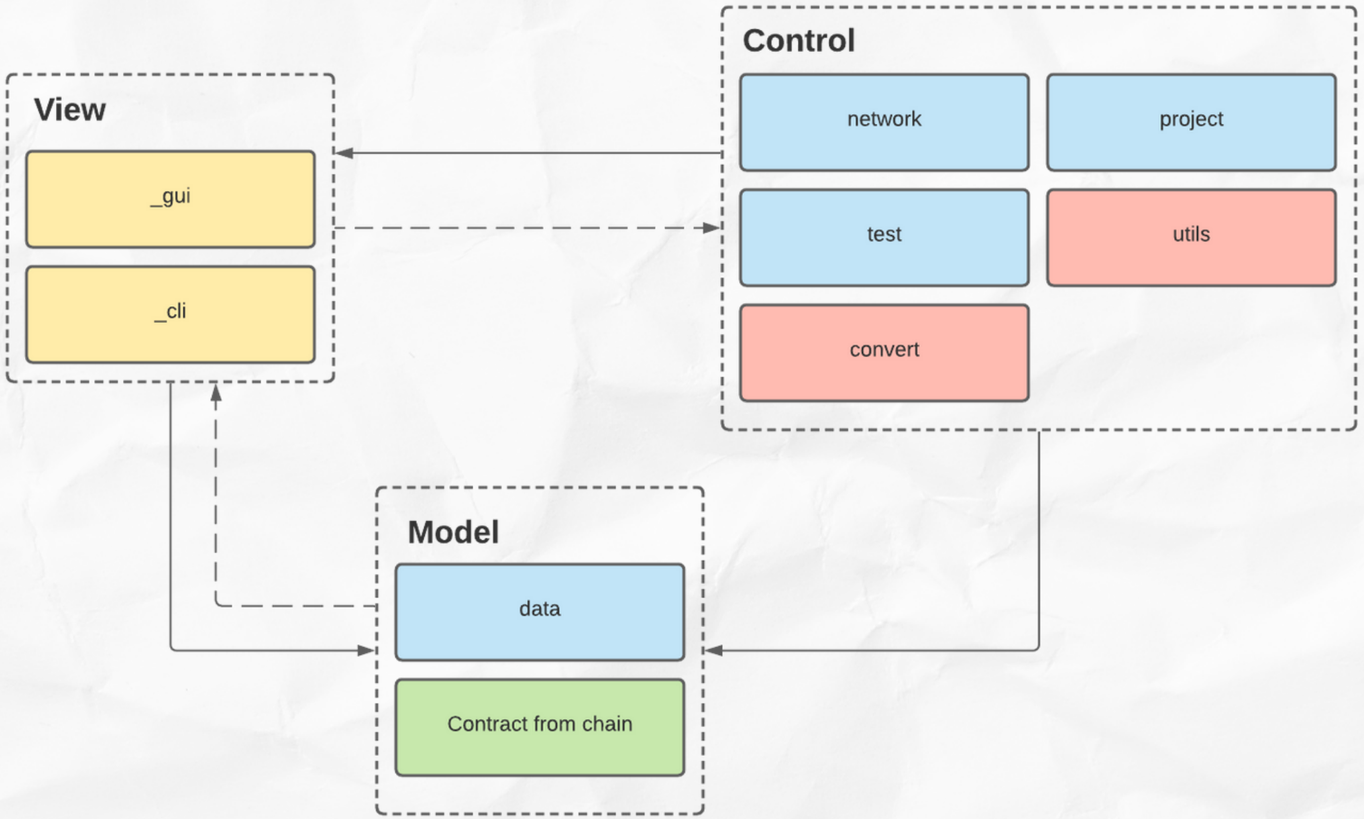
Think of eth-brownie as a client that connects with the server (In this case some node inside the Ethereum network)

MODEL-VIEW-CONTROLLER (MVC)



MVC

SIMPLIFIED Version



EXPLAIN

eth-brownie architecture can also be interpreted as MVC architecture.

The GUI, CLI can be categorized as View, Control is the part that does the operation with the data. The Model is data and other contracts on the chain.

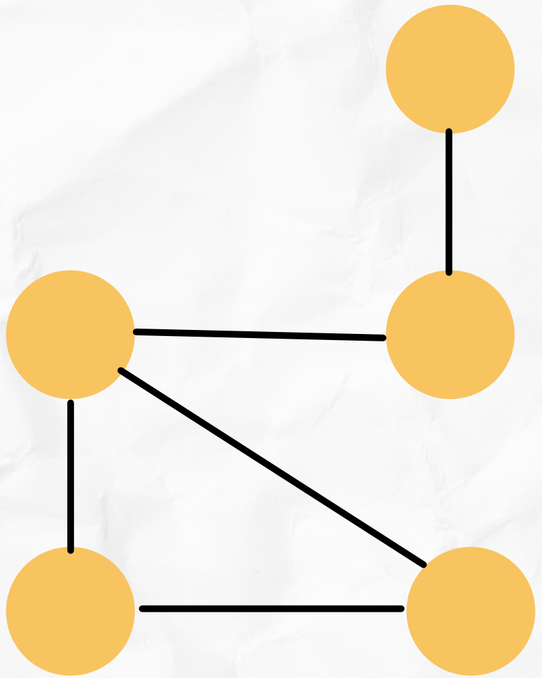
ARCHITECTURE

PROS

eth-brownie architecture is client-server and somewhat MVC this makes eth-brownie have loose coupling property. In which making eth-brownie really extensible, robust, easy to maintain, and really emphasize on its composability

CONS

eth-brownie architecture is really composable due to its nature as a library, Every module was really small. But it has a drawback that in order todo some task it require a lot of Boilerplate code.



03

QUALITY ARTRIBUTES

Interopability

Brownie **provide a lot of ways** to interact with (Graphical User Interface, Command Line Interface, Library)

Which promotes its interoperability a lot.

Also, You can just use pip to install eth-brownie into your project.
(this implies that eth-brownie has complied to the PIP standard)

Availability

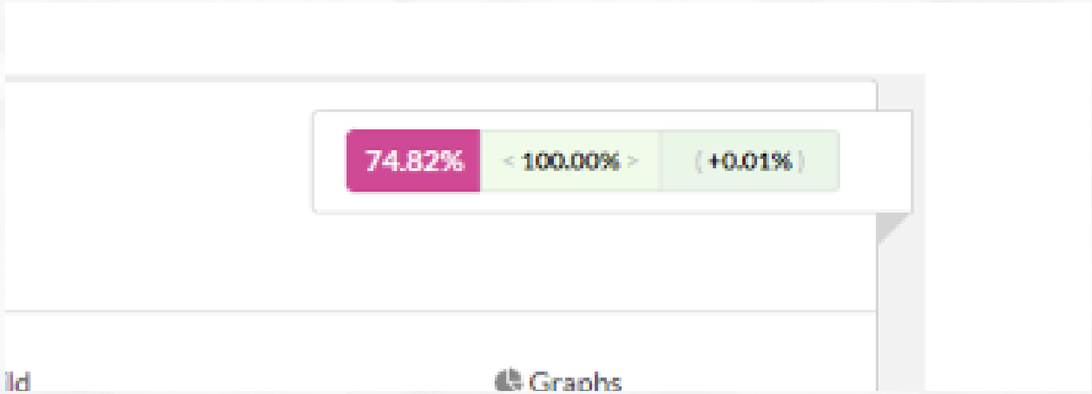
Brownie provides ways to pick **multiple RPC** thus increasing availability in case some RPC is down.

Modifiability

Brownie uses **Strategy Design patterns that increase modifiability** when some components want to change. Strategy Pattern is used in many ways for example in gas calculations.

⚗ Testability

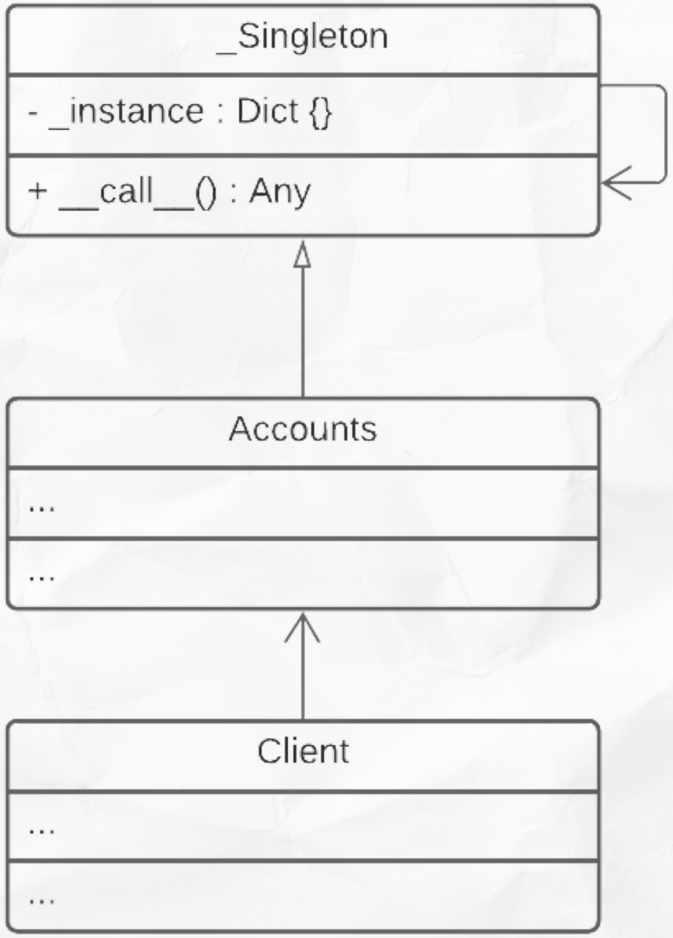
eth-brownie has a **high coverage score (74.82%)** indicating their ease of testing.



04

BROWNIE'S DESIGN PATTERN

CREATIONAL: SINGLETON




```python
#!/usr/bin/python3
from typing import Any, Dict, Tuple


class _Singleton(type):

    _instances: Dict = {}

    def __call__(cls, *args: Tuple, **kwargs: Dict) -> Any:
        if cls not in cls._instances:
            cls._instances[cls] = super(_Singleton, cls).__call__(*args, **kwargs)
        return cls._instances[cls]
```

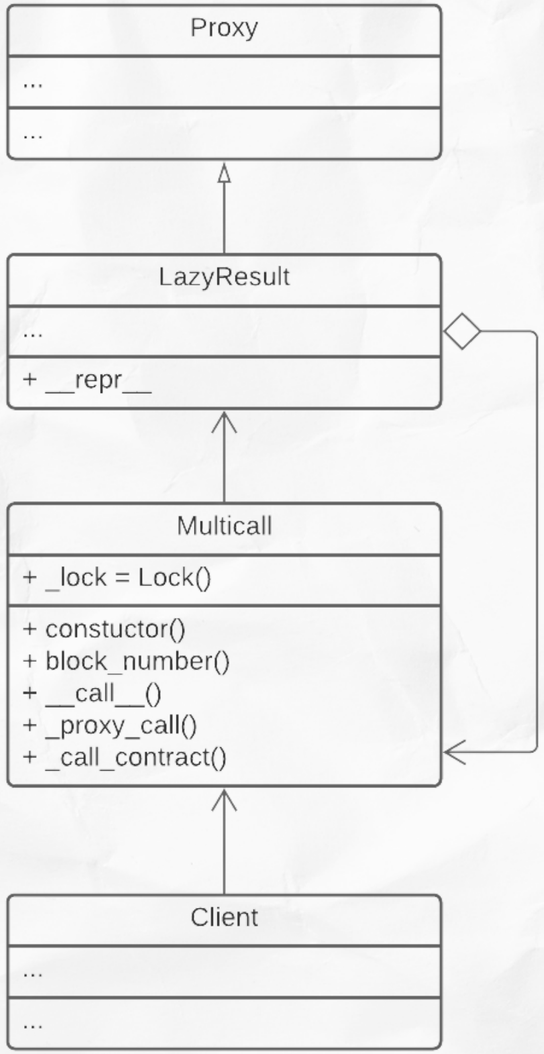
CREATIONAL: SINGLETON

WHY

ETH-Brownie uses a singleton pattern to make sure it has only one instance of some type of object.

In the source code, we found that eth-brownie use Singleton with various classes such as Config, TxHistory, Accounts, etc.,

STRUCTURAL : PROXY



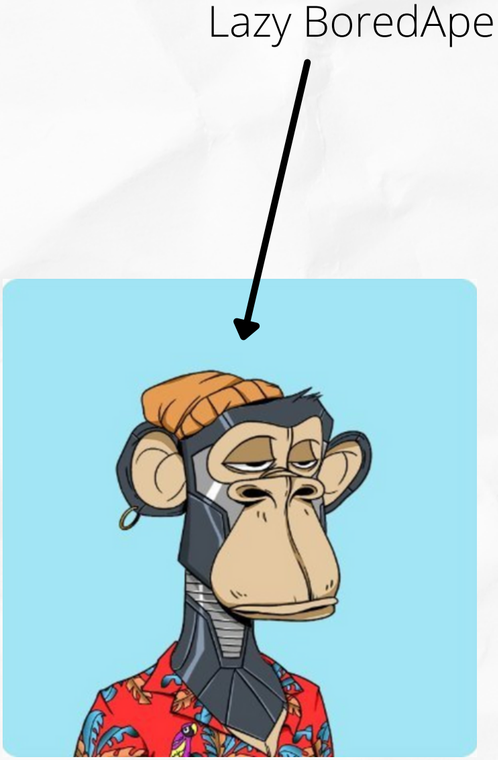

```python
class LazyResult(Proxy):
    """A proxy object to be updated with the result of a multicall."""

    def __repr__(self) -> str:
        return repr(self.__wrapped__)


class Multicall:
    """Context manager for batching multiple calls to constant contract functions."""

        ...

    def _call_contract(self, call: ContractCall, *args: Tuple, **kwargs: Dict[str, Any]) -> Proxy:
        """Add a call to the buffer of calls to be made"""
        calldata = (call._address, call.encode_input(*args, **kwargs))  # type: ignore
        call_obj = Call(calldata, call.decode_output)  # type: ignore
        # future result
        result = Result(call_obj)
        self._pending_calls[get_ident()].append(result)

        return LazyResult(lambda: self._flush(result))
```

STRUCTURAL: PROXY

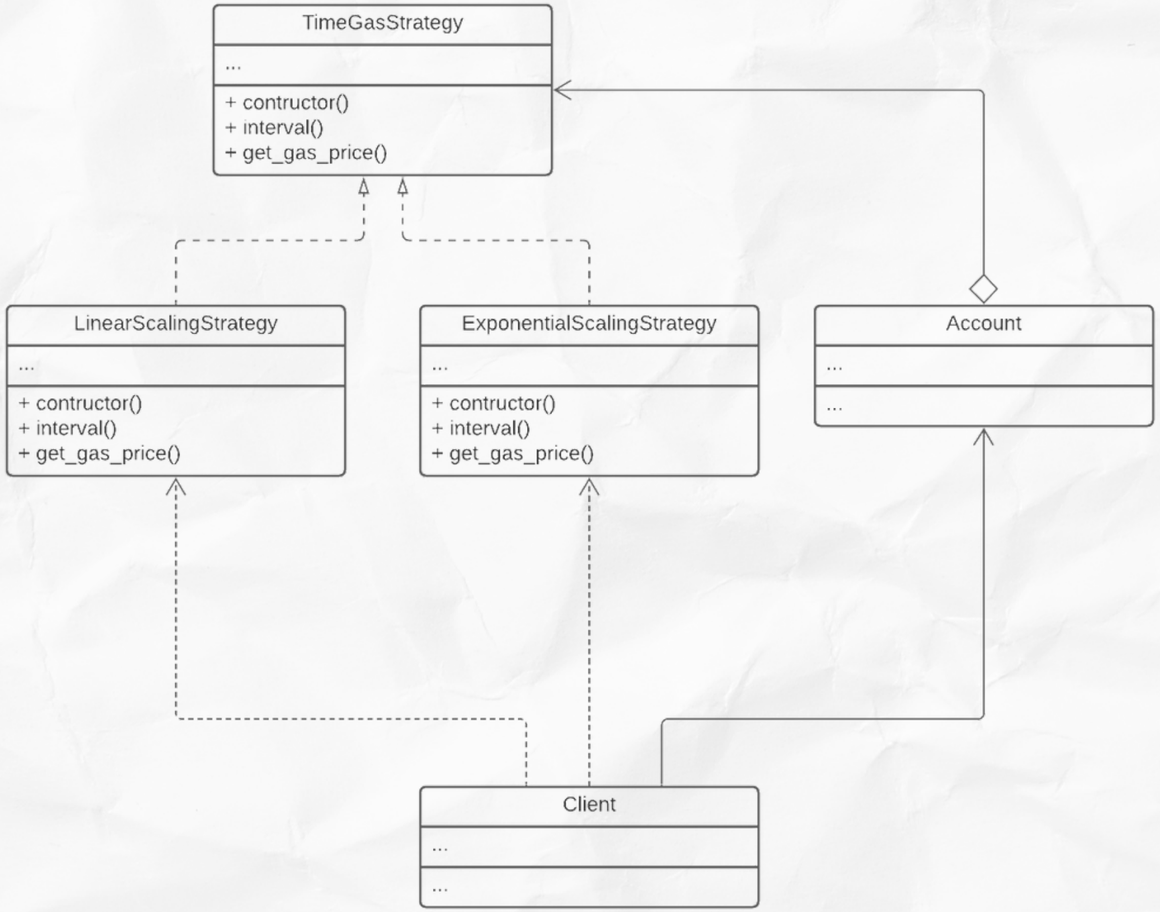
WHY

ETH-Brownie uses a proxy pattern to allow some operation to happen before really accessing it.

In the source code, we found that eth-brownie use **Lazy Load proxy** when making a request to blockchain in which is a common use case for the proxy pattern.



BEHAVIOR: STRATEGY




```python
class TimeGasStrategy(ScalingGasABC):
    """
    Abstract base class for time gas strategies.
    """
    duration = 30

    def __init__(self, time_duration: int = 30) -> None:
        self.duration = time_duration

    def interval(self) -> int:
        return int(time.time())

    @abstractmethod
    def get_gas_price(self) -> Generator[int, None, None]:
        raise NotImplementedError

        ...

class LinearScalingStrategy(TimeGasStrategy):
    """
    Gas strategy for linear gas price increase.
    """

    def __init__(
        self,
        initial_gas_price: Wei,
        max_gas_price: Wei,
        increment: float = 1.125,
        time_duration: int = 30,
    ):
        super().__init__(time_duration)
        self.initial_gas_price = Wei(initial_gas_price)
        self.max_gas_price = Wei(max_gas_price)
        self.increment = increment

    def get_gas_price(self) -> Generator[Wei, None, None]:
        last_gas_price = self.initial_gas_price
        yield last_gas_price

        while True:
            last_gas_price = min(Wei(last_gas_price * self.increment), self.max_gas_price)
            yield last_gas_price
```

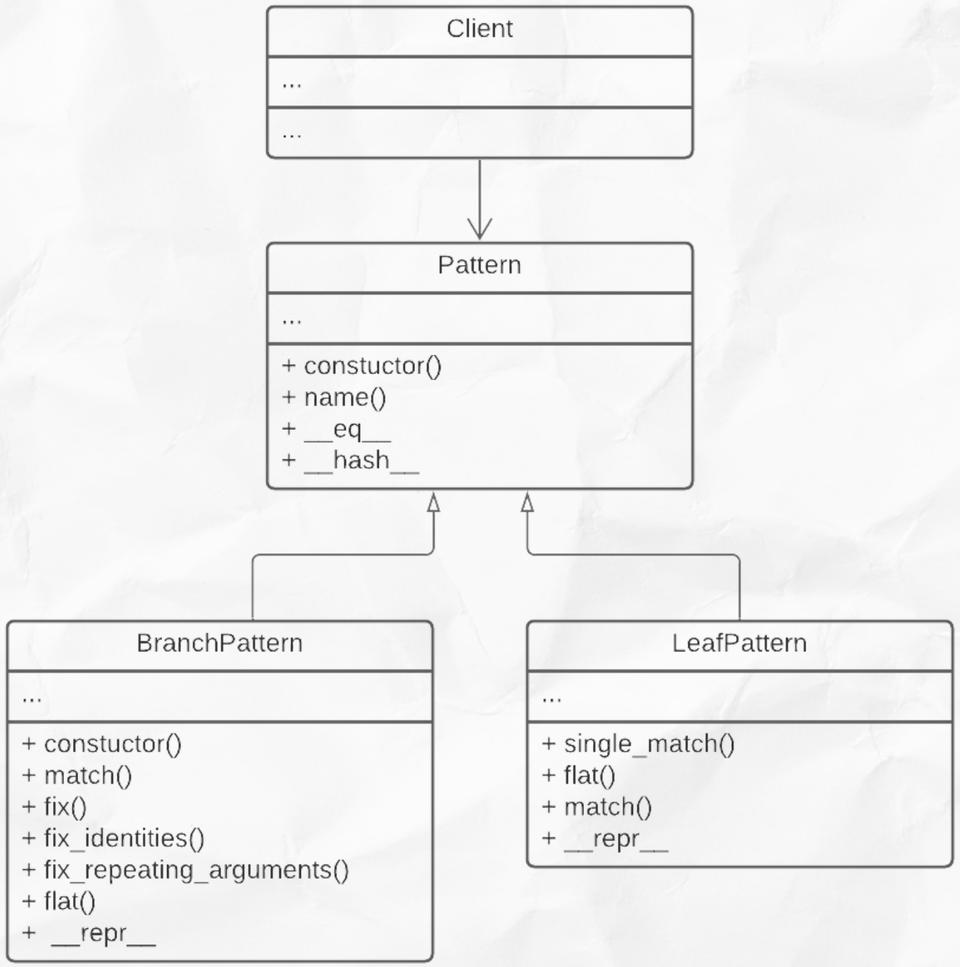
BEHAVIOR: STRATEGY

WHY

Brownie used a strategy pattern so it could provide multiple algorithms to do some tasks.

In the source code, we found that eth-brownie use **strategy pattern for Gas Price Approximation** that will be used in which have different ways to approximate.

BEHAVIOR: INTERPRETER




```python
class LeafPattern(Pattern):

    """Leaf/terminal node of a pattern tree."""

    def __repr__(self) -> str:
        return "%s(%r, %r)" % (self.__class__.__name__, self.name, self.value)

    def single_match(self, left: List["LeafPattern"]) -> TSingleMatch:
        raise NotImplementedError  # pragma: no cover

    def flat(self, *types) -> List["LeafPattern"]:
        return [self] if not types or type(self) in types else []

    ...


class BranchPattern(Pattern):

    """Branch/inner node of a pattern tree."""

    def __init__(self, *children) -> None:
        self.children = list(children)

    def match(self, left: List["Pattern"], collected: List["Pattern"] = None) -> Any:
        raise NotImplementedError  # pragma: no cover

    def fix(self) -> "BranchPattern":
        self.fix_identities()
        self.fix_repeating_arguments()
        return self

    def __repr__(self) -> str:
        return "%s(%s)" % (self.__class__.__name__, ", ".join(repr(a) for a in self.children))

    def flat(self, *types) -> Any:
        if type(self) in types:
            return [self]
        return sum([child.flat(*types) for child in self.children], [])
    ...
```

BEHAVIOR: INTERPRETER

Moon

WHY

Brownie used an interpreter pattern so it could provide **parsing capability** to use in its CLI.

In the source code, we found that eth-brownie uses an interpreter pattern to parse the command. It provides multiple grammars so it could work on CLI function well.

QR Slide

THANK YOU !

62010609
พักตร์ภูมิ ตาแพร่

62010293
ณัฐวุฒิ ครองอารีธรรม

62010452
นนทกร จิตรชิรานันท์

62010497
นิธิ น้อมประวัติ

62010278
ณัฐพนธ์ สุขถาวร

62010284
ณัฐภูมิ เพ็ชรชนะ

WE'RE OPEN TO QUESTIONS.