Manticore (PiggyBank, Lockbox, Lottery)

Portland State
Computer Science

PiggyBank level

```
contract PiggyBank{
    using SafeMath for uint256;
    uint256 public piggyBalance;
    string public name;
    address public owner;
    constructor(address _ctfLauncher, address _player, string _name) public payable {
        name=_name;
        owner=msg.sender;
        piggyBalance=piggyBalance.add(msg.value);
    }
    function() external payable {
        piggyBalance=piggyBalance.add(msg.value);
    }
    modifier onlyOwner() {
        require(msg.sender == owner, "Unauthorized: Not Owner");
        _;
    }

    function withdraw(uint256 amount) internal {
        piggyBalance = piggyBalance.sub(amount);
        msg.sender.transfer(amount);
    }

    function collectFunds(uint256 amount) public onlyOwner {
        require(amount<=piggyBalance, "Insufficient Funds in Contract");
        withdraw(amount);
    }
}
```

```
contract CharliesPiggyBank is PiggyBank{

    uint256 public withdrawlCount;

    constructor(address _ctfLauncher, address _player) public payable PiggyBank
(_ctfLauncher, _player, "Charlie") {
        withdrawlCount = 0;
    }

    function collectFunds(uint256 amount) public {
        require(amount<=piggyBalance, "Insufficient Funds in Contract");
        withdrawlCount = withdrawlCount.add(1);
        withdraw(amount);
    }
}
```

Not protected

Have Manticore automatically generate the transaction that wipes out the contract's balance (via calculation of `msg.data`).

Note the parameter must also be symbolic.

Solution script

```
# Set the amount of ETH you want to obtain from the contract
contract_balance = ???

# Create the PiggyBank CTF level contract on the EVM using wallet
contract_account = m.solidity_create_contract(
    contract_src,
    contract_name="CharliesPiggyBank",
    owner=user_account,
    balance=contract_balance,
    args=(0,0)
)

# Make symbolic buffer to hold msg.data and ask Manticore to calculate the "winning" value
# 4 bytes for the function signature hash and ??? more for a uint256
sym_args = m.make_symbolic_buffer(4+???)

# Issue a symbolic transaction to the EVM by setting msg.data to symbolic buffer
m.transaction(
    caller=user_account,
    address=contract_account.address,
    data=sym_args,
    value=0,
    gas=gas
)
```

Lockbox level

```
pragma solidity 0.4.24;

contract Lockbox1 {
    uint256 private pin;

    constructor(address _ctfLauncher, address _player) public payable {
        pin = now%10000;
    }

    function unlock(uint256 _pin) external{
        require(pin == _pin, "Incorrect PIN");
        msg.sender.transfer(address(this).balance);
    }
}
```

Have Manticore automatically generate the transaction that wipes out the contract's balance (via calculation of `msg.data`)

Solution script

```
from manticore.ethereum import ManticoreEVM
from MEVMCustomState import MEVMCustomState
from manticore.core.smtlib import ConstraintSet
from manticore.platforms import evm
from manticore.ethereum.state import State

# Create the initial Ethereum world state at the time the Lockbox
#   CTF level is created
#   1. Start with no constraints
initial_constraints = ConstraintSet()
#   2. Then, create custom world state with specified Unix timestamp
initial_world = evm.EVMWorld(initial_constraints, timestamp=???)
initial_state = State(initial_constraints, initial_world)

# Instantiate Manticore's Symbolic Ethereum Virtual Machine
#   and specify its initial_state as the one created
m = MEVMCustomState(initial_state=initial_state)

# Create an account for your wallet address on the EVM.
user_account = m.create_account(address=from_address, balance=contract_balance)

# Create the Lockbox CTF level contract on the EVM using wallet
contract_account = m.solidity_create_contract(
    contract_src,
    contract_name="Lockbox1",
    owner=user_account,
    balance=contract_balance,
    args=(0,0)
)
```

- …as before

```
# Make symbolic buffer to hold msg.data and ask Manticore to calculate the "winning" value
# 4 bytes for the function signature hash and ??? more for a uint256
sym_args = m.make_symbolic_buffer(4+???)

# Issue a symbolic transaction to the EVM by setting msg.data to symbolic buffer
m.transaction(
    caller=user_account,
    address=contract_account.address,
    data=sym_args,
    value=0,
    gas=gas
)

# Symbolically execute program to find an exploit that obtains our funds back.
for state in m.running_states:
    world = state.platform
    # Check if funds can be retrieved
    if state.can_be_true(world.get_balance(user_account.address) == contract_balance):
      # If so, add constraint
      #   Then concretize symbolic buffer to provide one solution
      state.constraints.add(world.get_balance(user_account.address) == contract_balance)
      conc_args = state.solve_one(sym_args)
      # Print out our transaction to send to win
      print(f'''eth.sendTransaction({{data:"0x{conc_args.hex()}", from:"0x{from_address:
040x}", to:"0x{si_level_address:040x}", gas:{gas}}})''')
      sys.exit(0)
```

Lottery level

```
pragma solidity 0.4.24;
contract Lottery {
    using SafeMath for uint256;
    uint256 public totalPot;
    constructor(address _ctfLauncher, address _player) public payable {
        totalPot = totalPot.add(msg.value);
    }

    function() external payable{
        totalPot = totalPot.add(msg.value);
    }

    function play(uint256 _seed) external payable {
        require(msg.value >= 1 finney, "Insufficient Transaction Value");
        totalPot = totalPot.add(msg.value);
        bytes32 entropy = blockhash(block.number);
        bytes32 entropy2 = keccak256(abi.encodePacked(msg.sender));
        bytes32 target = keccak256(abi.encodePacked(entropy^entropy2));
        bytes32 guess = keccak256(abi.encodePacked(_seed));
        if(guess==target){
            //winner
            uint256 payout = totalPot;
            totalPot = 0;
            msg.sender.transfer(payout);
        }
    }
}
```

Solution script

```
from manticore.ethereum import ABI, ManticoreEVM

# Set the amount of ETH you want to obtain from the contract
contract_balance = ???

# Set the amount of ETH we need to send in our transaction (msg.value) to play.
msg_value = ???

# Create an account for your wallet address on the EVM with funds to
# both deploy the contract and to play the lottery
user_account = m.create_account(address=from_address, balance=contract_balance+msg_value)

# Create the Lottery CTF level contract on the EVM using wallet
contract_account = m.solidity_create_contract(
    contract_src,
    contract_name="Lottery",
    owner=user_account,
    balance=contract_balance,
    args=(0,0)
)

# Make symbolic buffer to hold msg.data and ask Manticore to calculate the "winning" value
# 4 bytes for the function signature hash and ??? more for a uint256
sym_args = m.make_symbolic_buffer(???)
```

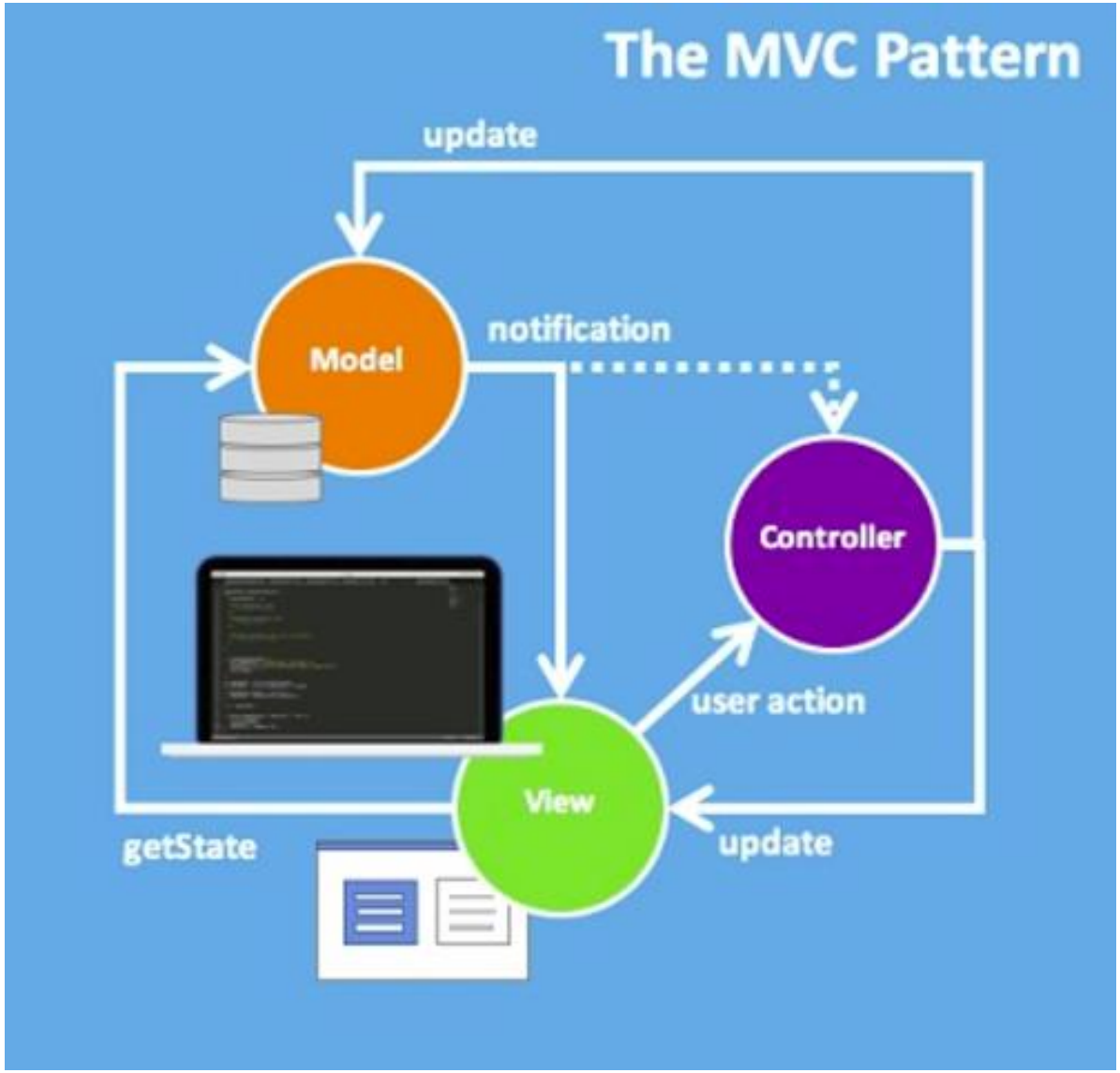
```
# Issue a symbolic transaction to the EVM by setting msg.data to symbolic buffer
#   as well as msg.value to the amount needed to play
m.transaction(
    caller=user_account,
    address=contract_account.address,
    data=sym_args,
    value=msg_value,
    gas=gas
)

# Symbolically execute program to find an exploit that obtains our funds back.
for state in m.running_states:
    world = state.platform
    # Check if funds can be retrieved
    if state.can_be_true(world.get_balance(user_account.address) == contract_balance +
 msg_value):
      # If so, add constraint
      #   Then concretize symbolic buffer to provide one solution
      state.constraints.add(world.get_balance(user_account.address) == contract_balanc
e + msg_value)
      conc_args = state.solve_one(sym_args)
      # Print out our transaction to send to win
      print(f'''eth.sendTransaction({{data:"0x{conc_args.hex()}", from:"0x{from_addres
s:040x}", to:"0x{si_level_address:040x}", value:{msg_value}, gas:{gas}}})''')
      sys.exit(0)
    print('No state found')
```

Vyper Guestbook

Portland State
Computer Science

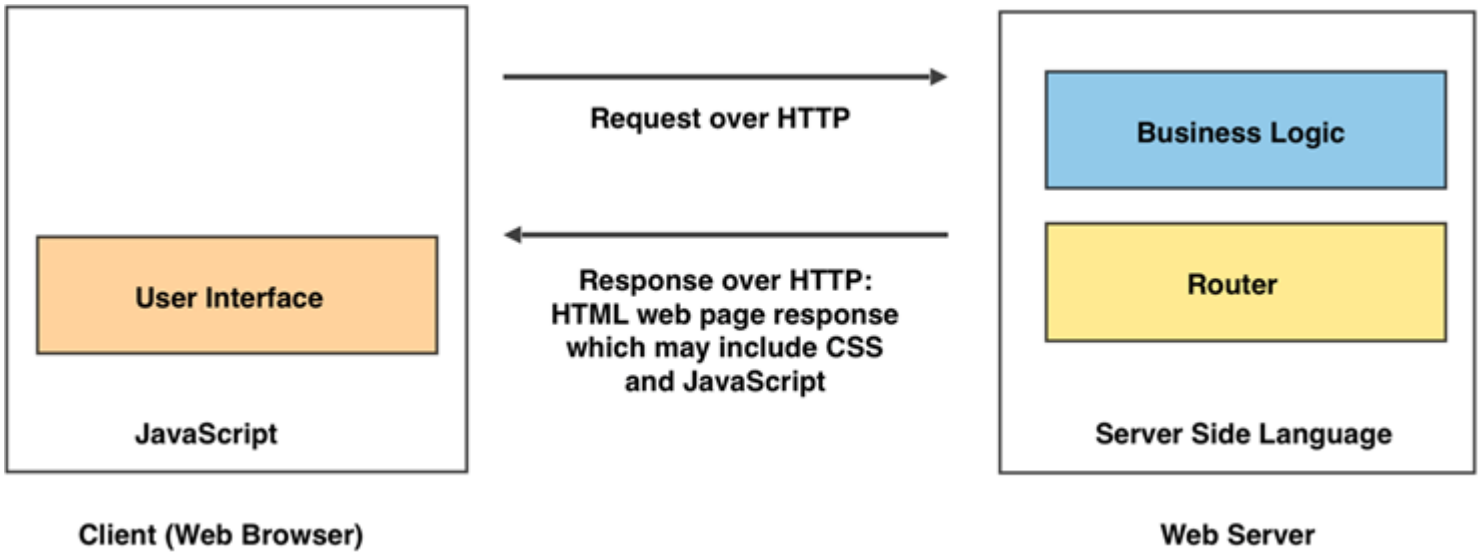
MVC (Model-View-Controller) architecture

- Developed in 1970s (Smalltalk), adapted everywhere
- First used in web applications in late 1990s via Struts
- Model
  - Code that encapsulates backend application data
  - Data representation and storage
- View
  - Code for rendering that generates HTML output and presents functionality to user (UI)
- Controller
  - Code connecting model and view
    - Takes in user requests to update model
    - Pulls in model data to supply the view
- Separation of concerns
  - Simplifies swapping out database backend or the UI frontend


The MVC Pattern
update
Model
notification
Controller
user action
View
getState
update

Server rendering with MVC

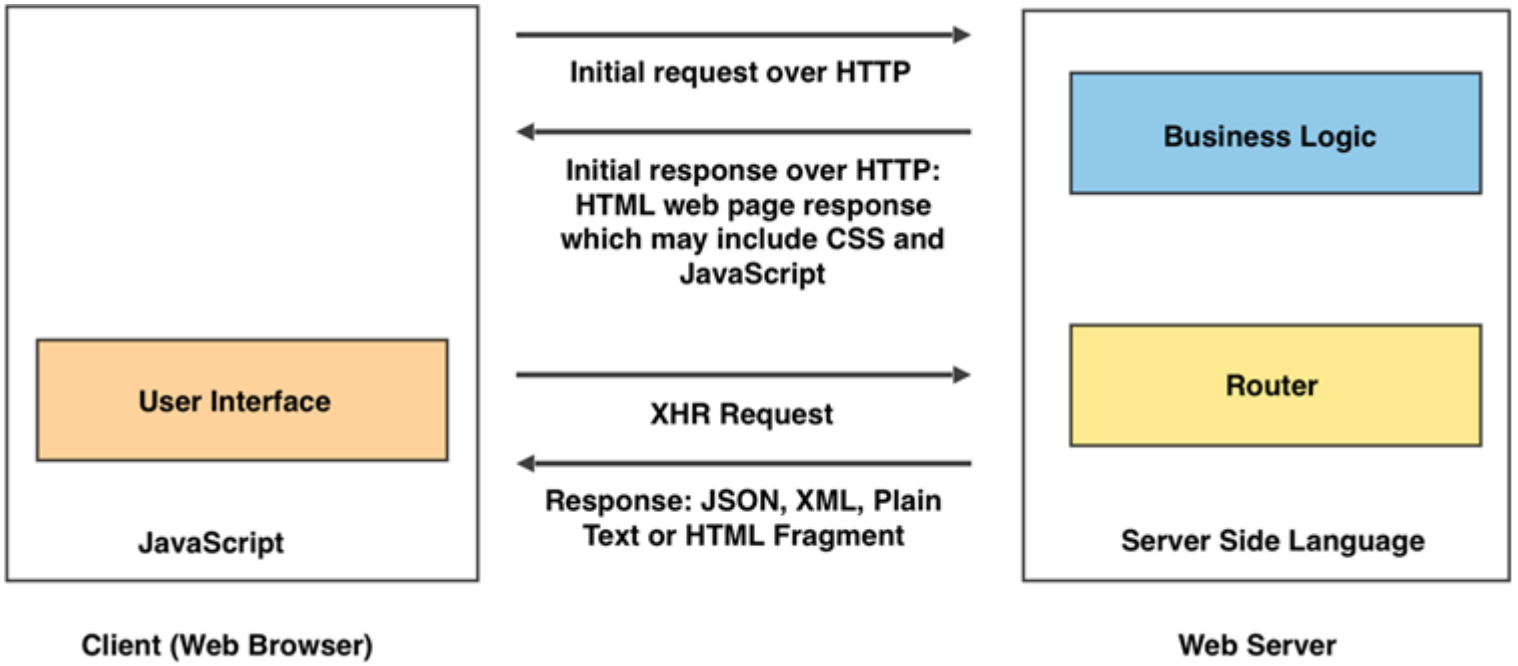
- Two rendering steps
  - Web application renders server data into HTML/CSS for web browser
  - Web browser renders HTML/CSS for user
  - Repetitive re-rendering at server
  - Connection overhead and latency issues



(Figures used from K. Balasubramanian, "Isomorphic Go")

Client-side rendering

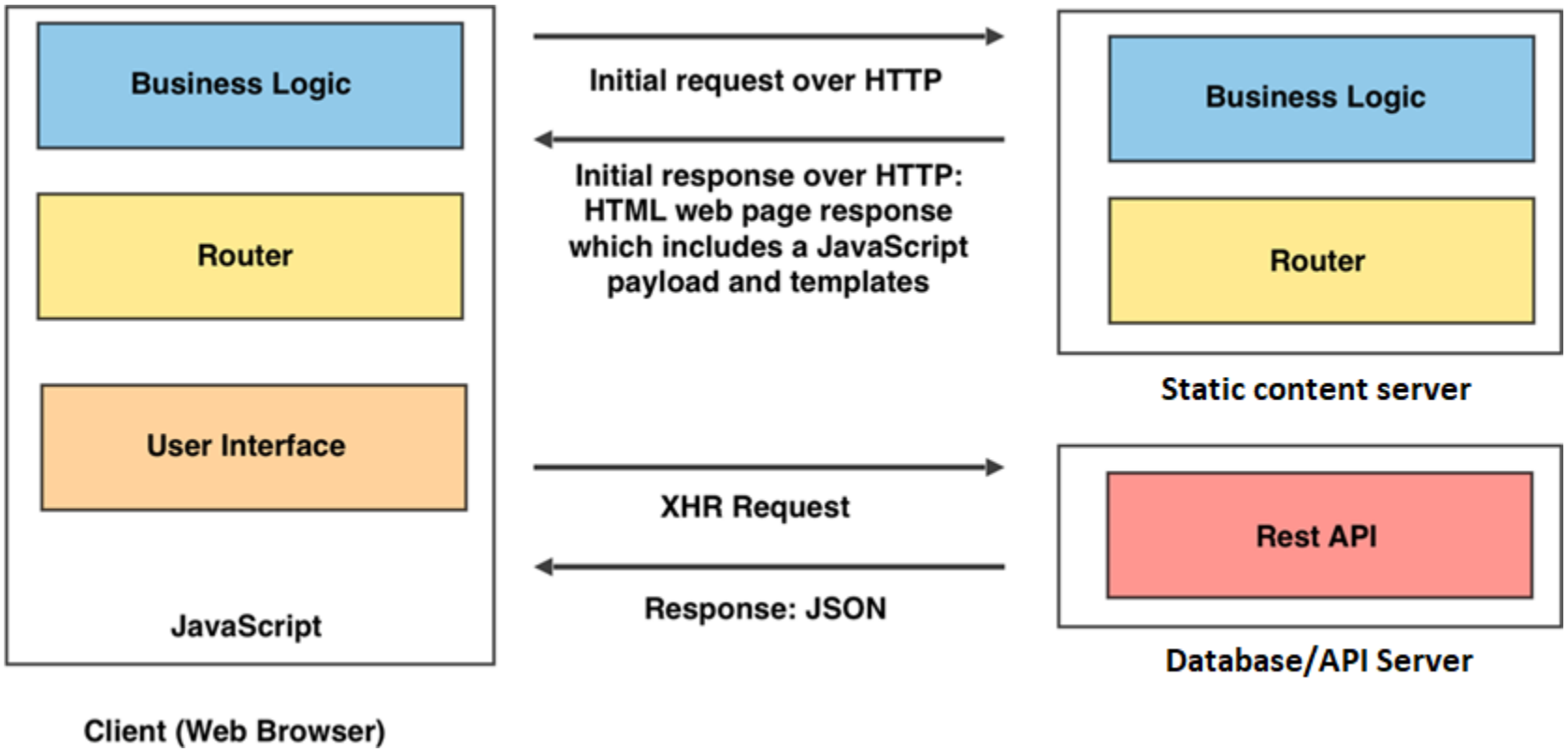
- Allow client to retrieve/update page elements via AJAX (Asynchronous JavaScript and XML)
  - Reduces amount of rendering performed on server-side
  - Client-side operations more responsive (reduce page reloads)
  - Can have just the changed portions of page downloaded
  - Fewer connections and decreased load on server
  - Can one render and ship the entire application and push it to the client?



Single-page applications

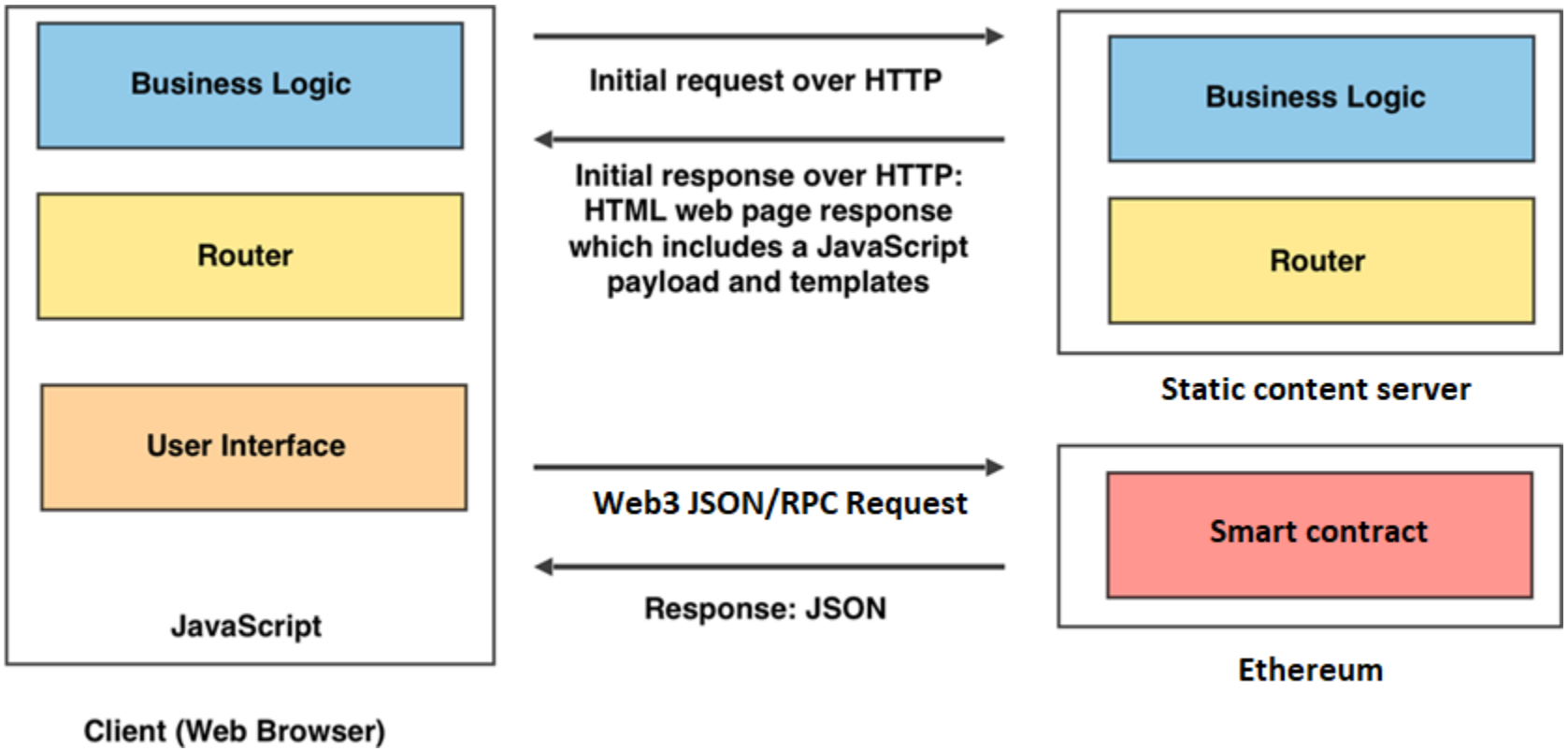
- Ship entire web application to client in a single package
  - Controller, View, and page rendering all on client
  - Typically, a single `index.html` file, a CSS bundle, and a JavaScript bundle
  - Examples: GMail, Google Maps, Facebook, Github
  - Backend only supplies model via an API and database server
    - e.g via REST/gRPC or GraphQL (React)
  - Web app view directly interacts with API
- Advantages
  - Essential static content delivered ahead of time with no need to refresh
  - Faster UI since network calls minimized

- JAM applications (JavaScript, REST-ful APIs, static Markup)
  - Web applications without web servers



Vyper Guestbook

- REST API replaced with Smart contract backend



Vyper code

```
# pragma @version ^0.2.4

# Guestbook entry structure containing a single entry
struct gbentry:
    signer: address
    name: String[32]
    email: String[32]
    message: String[100]
    date: uint256
    bounty_entry: uint256

# Size of the guestbook
ENTRIES: constant(uint256) = 3

# Owner of the guestbook contract to send funds to upon selfdestruct
owner: public(address)

# List of guestbook entries
gb: public(gbentry[ENTRIES])

# Current minimum bounty.  Calls to sign guestbook must exceed this
# value in order to be added.
bounty: public(uint256)

# Event emitted to web3 front-end when the guestbook changes. Sends
# the new bounty value.
event Entry:
    value: uint256
```

```
# Constructor that initializes guestbook and its initial entries
@external
def __init__():
    self.owner = msg.sender
    self.bounty = 0
    for i in range(ENTRIES):
        self.gb[i].signer = msg.sender
        self.gb[i].name = "Owner of Contract"
        self.gb[i].email = "owner@pdx.edu"
        self.gb[i].message = "Hello!"
        self.gb[i].date = block.timestamp
        self.gb[i].bounty_entry = convert(i*10,uint256)


# Destroy contract and return funds to the contract owner
@external
def cashOut():
    selfdestruct(self.owner)

# Contract accepts any ETH someone wants to send us!
@external
@payable
def __default__():
    pass
```

```
# Finds the minimum bounty value and updates the storage variable for it
@internal
def update_bounty():
    minimum: uint256 = 0
    for i in range(ENTRIES):
        if (minimum == 0) or (self.gb[i].bounty_entry < minimum):
            minimum = self.gb[i].bounty_entry
    self.bounty = minimum

# Implement insertion of new entry. Signer must supply sufficient funds
# in excess of the bounty in order for call to succeed.  Upon success,
# emit Event to front-end
@external
@payable
def sign(name: String[32], email: String[32], message: String[100]):
    assert msg.value > self.bounty
    for i in range(ENTRIES):
        if self.gb[i].bounty_entry == self.bounty:
            self.gb[i].signer = msg.sender
            self.gb[i].name = name
            self.gb[i].email = email
            self.gb[i].message = message
            self.gb[i].date = block.timestamp
            self.gb[i].bounty_entry = msg.value
            break
    self.update_bounty()
    log Entry(self.bounty)
```

Vyper front-end (index.html)

- Bounty and account elements (filled in via JavaScript)
- Form elements and submission UI (submitted via JavaScript)
- Entries elements (filled in via JavaScript)

```
<div class=page>
<h2>Guestbook</h2>
  <div>Bounty: <span id="bounty"></span></div>
  <div>Account: <span id="account"></span></div>
  <div>
      <div> <label for="name">Name: </label>
            <input id='name' type='text' name='name'>
      </div>
      <div> <label for="email">Email: </label>
            <input id='email' type='text' name='email'>
      </div>
      <div> <label for="message">Message: </label><br>
        <textarea id="message" rows=2 cols=50 name="message"></textarea>
      </div>
      <button id="sign">Sign</button>
  </div>
<h2>Entries</h2>
  <div id="entries"></div>
```

```
    <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/web3@l
atest/dist/web3.min.js"></script>
    <script type="text/javascript" src="guestbook.js"></script>
    <script type="text/javascript">
      updateBounty();
      updateAccount();
      updateEntries();
    </script>
    </div>
  </body>
</html>
```

Vyper front-end (guestbook.js)

```
/* ================================================================================*/
/* Javascript code for Guestbook DApp
/* ================================================================================*/

/* Check if Metamask is installed. */
if (typeof window.ethereum !== 'undefined') {
    console.log('MetaMask is installed!');
} else {
    console.log('Please install MetaMask or another browser-based wallet');
}

/* Instantiate a Web3 client that uses Metamask for transactions.  Then,
 * enable it for the site so user can grant permissions to the wallet */
const web3 = new Web3(window.ethereum);
window.ethereum.enable();

/* Grab ABI from compiled contract (e.g. in Remix) and fill it in.
 * Grab address of contract on the blockchain and fill it in.
 * Use the web3 client to instantiate the contract within program */
var GuestbookABI = <FMI>;

var Guestbook = new web3.eth.Contract(GuestbookABI,'<FMI>');
```

```
/* ================================================================================*/
/* Update the UI with current wallet account address when called */
async function updateAccount() {
  const accounts = await window.ethereum.request({ method: 'eth_requestAccounts' });
  const account = accounts[0];
  const accountNode = document.getElementById("account");
  if (accountNode.firstChild)
    accountNode.firstChild.remove();
  var textnode = document.createTextNode(account);
  accountNode.appendChild(textnode);
}

/* ================================================================================*/
/* Update the UI with current minimum bounty when called */
async function updateBounty(){
  const bounty = await Guestbook.methods.bounty().call();
  updateBountyUI(bounty);
}

function updateBountyUI(value){
  const bountyNode = document.getElementById("bounty");
  if (bountyNode.firstChild)
    bountyNode.firstChild.remove();
  var textnode = document.createTextNode(value + " Wei");
  bountyNode.appendChild(textnode);
}
```

```
/* ======================================================================*/
/* Update the UI with Guestbook entries from contract when called */
async function updateEntries(){
  const entriesNode = document.getElementById("entries");
  while (entriesNode.firstChild) {
    entriesNode.firstChild.remove();
  }
  for (var i = 0 ; i < 3; i++) {
      var entry = await Guestbook.methods.gb(i).call();
      const nameAndEmail = document.createTextNode(
                              entry.name + " <" + entry.email + ">"  );
      const wallet = document.createTextNode(entry.signer);
      const entrydate = new Date(parseInt(entry.date)*1000);
      const signedOn = document.createTextNode("signed on " + entrydate.toUTCString()
+ " for " + entry.bounty_entry + " Wei");
      const message = document.createTextNode(entry.message);
      const br1 = document.createElement("br");
      const br2 = document.createElement("br");
      const br3 = document.createElement("br");
      const p = document.createElement("p");
      p.classList.add("entry");
      p.appendChild(nameAndEmail);
      p.appendChild(br1);
      p.appendChild(wallet);
      p.appendChild(br2);
      p.appendChild(signedOn);
      p.appendChild(br3);
      p.appendChild(message);
      entriesNode.appendChild(p);
  }
}
```

```
/* Issue a transaction to sign the guestbook based on form field values */
async function sign() {
  const accounts = await window.ethereum.request({ method: 'eth_requestAccounts' });
  const account = accounts[0];
  const name = document.getElementById("name").value;
  const email = document.getElementById("email").value;
  const message = document.getElementById("message").value;
  const bounty = await Guestbook.methods.bounty().call();

  bid = parseInt(bounty) + 10;
  const transactionParameters = {
    from: account,
    gasPrice: 0x1D91CA3600,
    value: bid
  };
  await Guestbook.methods.sign(name, email, message).send(transactionParameters);
};
```

```
/* Register a handler for when contract emits an Entry event after Guestbook is
 * signed to reload the page */
Guestbook.events.Entry().on("data", function(event) {
  updateBountyUI(event.returnValues.value);
  updateEntries();
});

/* Register an event listener on sign button and invoke sign
 * transaction when clicked */
const button = document.getElementById('sign');
button.addEventListener('click', () => {
  sign();
});
```