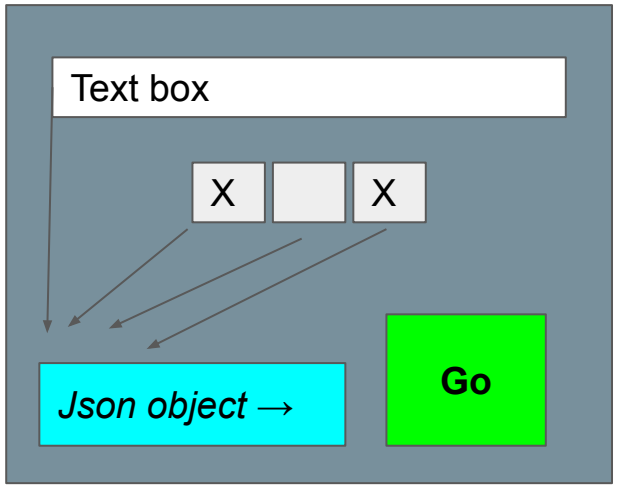
Advanced DApps

Agenda

- Designing Your Application
- Test Driven Development
- Continuous Integration
- Coding Guidelines
- VSCode Plugins
- Project Dependencies
- Homework - Continue

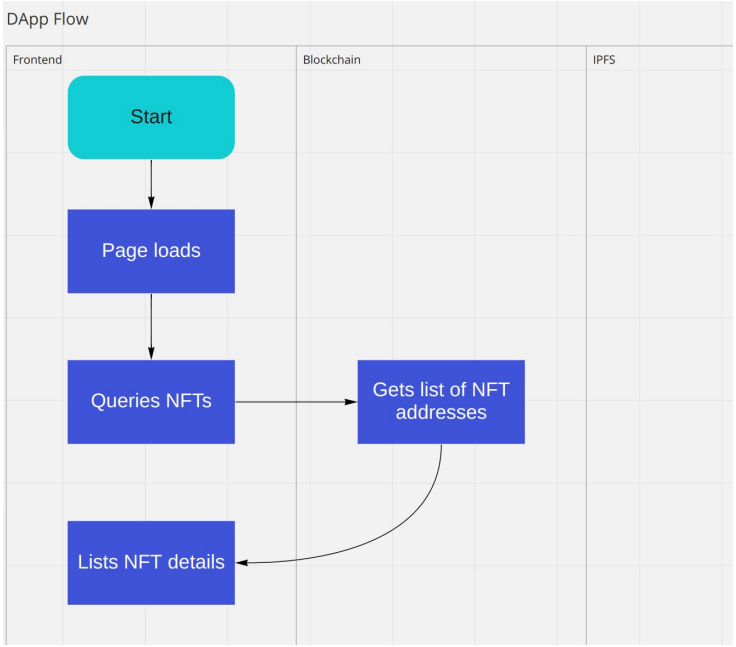
Designing Your Application

- Receive design spec
  - Outline of desired features e.g. **Input box, tickboxes, submit button**
- Sketch UI from design spec
  - Helps to visualise data sources and sinks



Designing Your Application

- Create flow charts / swim lanes https://miro.com/
  - Characterise data flows
  - Build up required functionality
  - Checks logic before writing any code
  - Share program flow to managers and other non-developers
- Maintain dialogue throughout design and development



EXTROPY.IO

Test Driven Development - Mocha

- Node.JS asynchronous testing framework
- Chai is a TDD assertion library for Mocha
- assert(expression, message)

```
assert('foo' !== 'bar', 'foo is not bar');
```

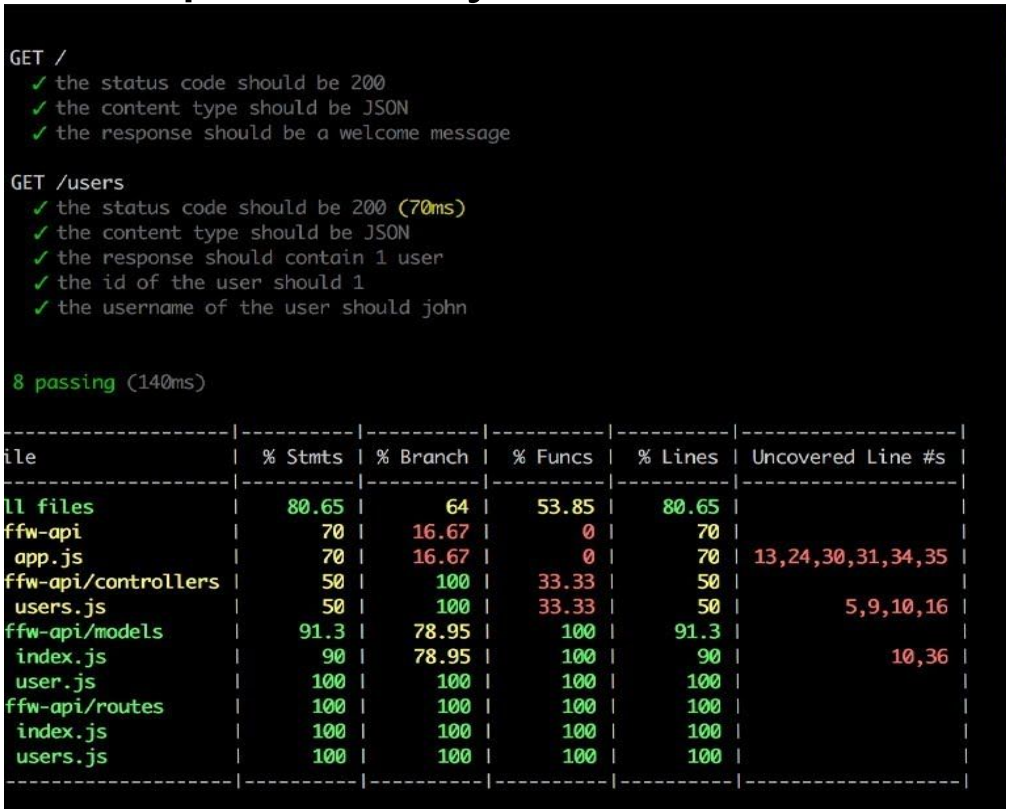
```
assert(Array.isArray([]), 'empty arrays are arrays');
```

```
assert.equal(x, y)
```

- https://mochajs.org/
- https://www.chaijs.com/api/assert/

EXTROPY.IO

Test Driven Development - nyc



```
GET /
  ✓ the status code should be 200
  ✓ the content type should be JSON
  ✓ the response should be a welcome message

GET /users
  ✓ the status code should be 200 (70ms)
  ✓ the content type should be JSON
  ✓ the response should contain 1 user
  ✓ the id of the user should 1
  ✓ the username of the user should john


 8 passing (140ms)

--------------------|----------|----------|----------|----------|-------------------|
ile                 |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
--------------------|----------|----------|----------|----------|-------------------|
ll files            |    80.65 |       64 |    53.85 |    80.65 |                   |
ffw-api             |       70 |    16.67 |        0 |       70 |                   |
 app.js             |       70 |    16.67 |        0 |       70 | 13,24,30,31,34,35 |
ffw-api/controllers |       50 |      100 |    33.33 |       50 |                   |
 users.js           |       50 |      100 |    33.33 |       50 |         5,9,10,16 |
ffw-api/models      |     91.3 |    78.95 |      100 |     91.3 |                   |
 index.js           |       90 |    78.95 |      100 |       90 |             10,36 |
 user.js            |      100 |      100 |      100 |      100 |                   |
ffw-api/routes      |      100 |      100 |      100 |      100 |                   |
 index.js           |      100 |      100 |      100 |      100 |                   |
 users.js           |      100 |      100 |      100 |      100 |                   |
--------------------|----------|----------|----------|----------|-------------------|
```

https://www.npmjs.com/package/nyc

Test Driven Development - Write the unit tests

```
const assert = require('chai').assert;
const myFuns = require('../src/someFuns.js');

// Failing test
it('should return something that is not an integer', () => {
      const input = "Hello";
      const newNumber = addOne(input);
      assert.isFalse(Number.isInteger(newNumber));
});

// Passing test
it('should return one more than the integer input', () => {
      const input = 1;
      const expected = 2;
      const actual = myFuns.addOne(input);
      assert.equal(actual, expected);
});
```

Test Driven Development - Write just enough code to pass

```
/**
* Adds 1 to the input
* @param {integer} input - Value to be incremented
* @return {integer} returnVal - The input value plus 1
*/
function addOne(input) {
  if (Number.isInteger(input)) {
      const plusOne = 1;
      const returnVal = input + plusOne;
      return returnVal;
  } else {
      return false;
  }
};
```

Test Driven Development - Refactor

```
/**
* Adds 1 to the input
* @param {integer} input - Value to be incremented
* @return {integer} - The input value plus 1
*/
function addOne(input) {
  if (Number.isInteger(input)) {
      return ++input;
  } else {
      return false;
  }
};
```

Test Driven Development - package.json Test Script

```
"scripts": {
  "test": "nyc --reporter=text mocha --exit"
},
```

```
[user]$ npm test
```

Test Driven Development - Code linting

- Linting is the automatic checking of code for programmatic and stylistic errors
- Performed as part of continuous development
- Promotes code readability
- Templates inform linting software to adhere to certain rules
  - Indentation
  - Variable names
  - Comment styles

EXTROPY.IO

Test Driven Development - .eslintrc.js

```
module.exports = {
 'env': {
   'browser': true,
   'commonjs': true,
   'es2021': true,
   'node': true,
 },
 'extends': ['google'],
 'parserOptions': {
   'ecmaVersion': 12,
 },
 'ignorePatterns': ['**/*.sol'],
 'rules': {
   'no-tabs': 0,
   'space-before-function-paren' : 0,
   'max-len': 0,
   'object-curly-spacing' : ['error', 'always'],
   'no-unused-vars' : ['warn', {}],};
```

```
   'new-cap': ['warn', {}],
   'indent': [2, 2],
   'require-jsdoc': [
     'warn',
     {
       'require': {
         'FunctionDeclaration' : true,
         'MethodDefinition' : true,
         'ClassDeclaration' : true,
         'ArrowFunctionExpression' : true,
         'FunctionExpression' : true,
       },
     },
   ],
 },
 'globals': {
   'it': true,
 },
};
```

Test Driven Development - .eslintignore

```
test/
truffle/
migrations/
```

Continuous Integration

- Production-safe multi-participant development environments
- Continuously deploy, test and monitor code
- Reduced gaps between functioning programs
- Catch issues early
- Prevent “integration hell”
- Automated workflow (reduced errors, standardisation and familiarity)
    - Runners as pseudo-production machines

Continuous Integration - Example NodeJS config file

```
image: node:latest

stages:
  - build
  - lint
  - test

Build:
  stage: build
  script:
    - echo "Building"
    - npm install
  artifacts:
    paths:
      - node_modules/
```

```
Code lint:
  stage: lint
  script:
    - echo "Linting NodeJS code"
    - |
      npm i eslint \
      eslint-config-google
    - node_modules/eslint/bin/eslint.js .

Test + Coverage:
  stage: test
  script:
    - echo "Testing"
    - npm test
```

Coding Guidelines - Overall

- Good code should (in priority order):
  a. Run all the tests.
  b. Contain no duplication.
  c. Express all the design ideas that are in the system.
  d. Minimise the number of entities such as classes, methods and functions.

Coding Guidelines - Filenames

- lowercase and can contain _ or -.

  `some_funs.js`

Coding Guidelines - Variables

- Use **const** over **let**. The use of **var** is actively prohibited when using a linter.
- No single character names.
- Global constant names should be UPPER_CASE.

```
const PI_APPROX = 3.14;
```

- Mutable variable names should be lowerCamelCase.

```
let timeNow = Date.now();
```

- Declare near point of use.

```
const someValue = "cheese string";

return someValue;
```

Coding Guidelines - Objects

Construct single instances of object literals instead of Object constructor:

```
const strArr = ["an", "array"];
```

or

```
let mutableJson = {"a": 1, "b": 2};
```

Instead of:

```
const strArrObj = new Array("an", "array");
```

EXTROPY.IO

Coding Guidelines - Loops

Prefer **for - of** and **Object.keys** over **for - in** where possible to:

- Gain direct access to the property

```
const numStrArr = ["first", "second", "third"];
for (let numStr of numStrArr) {
     console.log("This is the", numStr, "loop".)
}
```

```
const numStrArr = ["first", "second", "third"];
for (let numStr of Object.keys(numStrArr)) {
     console.log("This is the", numStr, "loop".)
}
```

- And avoid accessing non-iterable properties

```
const numStrArr = ["first", "second", "third"];
numStrArr.newProperty = "NaN";
// Prints "first", "second", "third", "NaN"
for (let numStr in numStrArr) {
     console.log(numStrArr[numStr])
}
```

Coding Guidelines - Error handling

- Translate exception to human-readable format.
- Include exception type.
- Raw error message shouldn't be handled.
- Catch exceptions in all cases where an exception may occur.

Coding Guidelines - Comments

JSDoc Comment

```
**/ Prints "Hello World" to console /*
function smallFunction() {
      console.log("Hello World");
}
```

JSDoc Comment with tags

```
**/
* Receives two integers and returns the sum.
* @param {integer} firstNumber - First number.
* @param {integer} secondNumber - Second number.
* @return {integer} The sum of both input numbers.
*/
function sumNumbers(firstNumber, secondNumber) {
    const sum = firstNumber + secondNumber;
    return sum;
}
```

Coding Guidelines - Bottlenecks / Scalability

- Database read / write
- Smart contract / RPC calls
- Backend API calls
- Application code
  - Smart contract calls can define nonce for quick succession transactions
- Hosting platforms offer auto-scaling options
- Blocktime
- Integrate layer 2 / 3 / 4… solutions to reduce congestion

Coding Guidelines - Upgradability

- Breaking changes
  - Backwards / forwards compatibility
  - Objects (e.g. option / variable files) generally cause problems
  - Functions can be overridden to prevent breaking existing applications
- Proxy smart-contract
  - Intermediate smart contract holding pointers to live smart contract
  - State might be lost - consider using other storage mechanisms

Homework Assignment: Recommended Software

- VSCode https://code.visualstudio.com/
- Node / NPM recommend use https://github.com/nvm-sh/nvm
- Ganache https://www.trufflesuite.com/ganache
- Postman https://www.postman.com/
- Browser (for contract development in Remix)

VSCode Plugins

- Git graph [mhutchie.git-graph]
- Prettier [esbenp.prettier-vscode]

Project Dependencies

- chai
- eslint
- mocha
- nyc

Exercise

- Interact with an ERC721 contract - see Homework 8 .pdf