COMP 431/531: Web Development

Lecture 14: Unit Testing

Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp431

Announcements & Reminders

HW #4 (Draft Front-end) due Tue. Oct. 21st at 11:59pm

classroom hw4 repo: https://classroom.github.com/a/hfLD_40C

In-Class Exercise 13: Angular REST

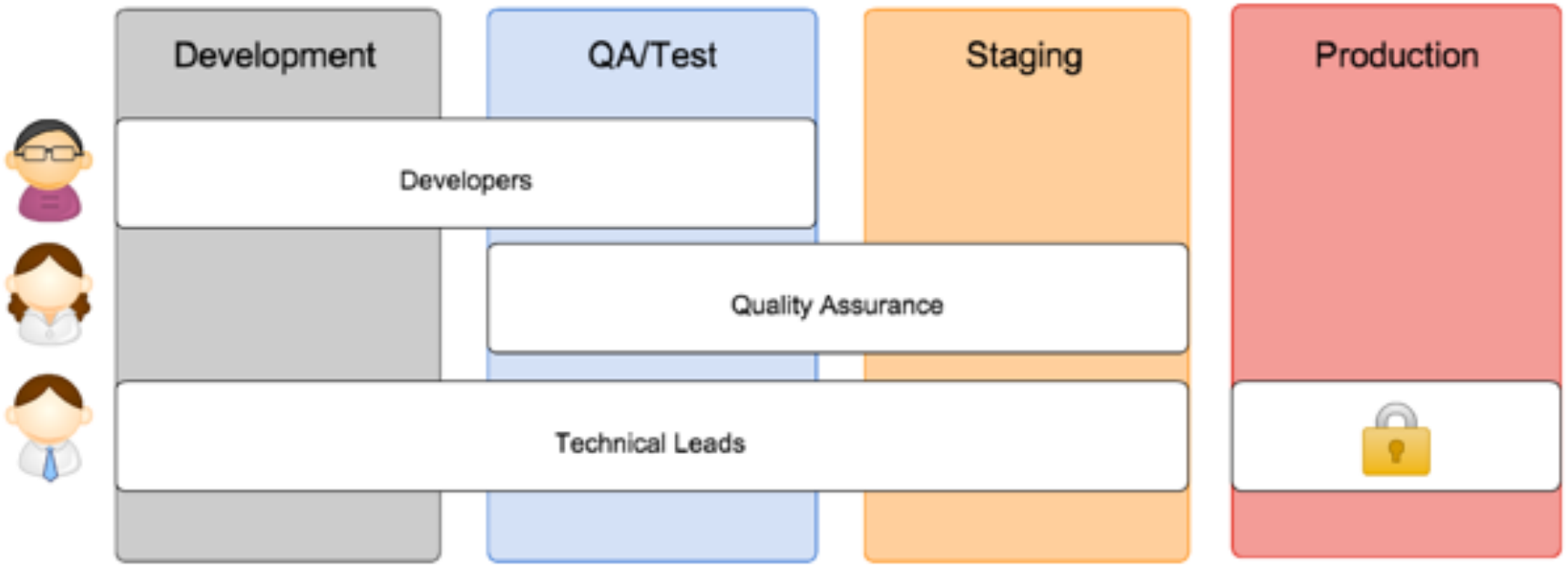
1. Create a new tic-tac-toe new component called *player*
2. Create new service for player component (call it *player*) in same directory as player component
3. Setup the route: “players” should use the PlayerComponent
4. Have player component employ service to get users array from https://jsonplaceholder.typicode.com/users
5. Use @for in template (player.component.html) to print ordered list of users’ first and last name.
6. Build and serve the application (view on http://localhost:4200)

   >> cd tictactoe

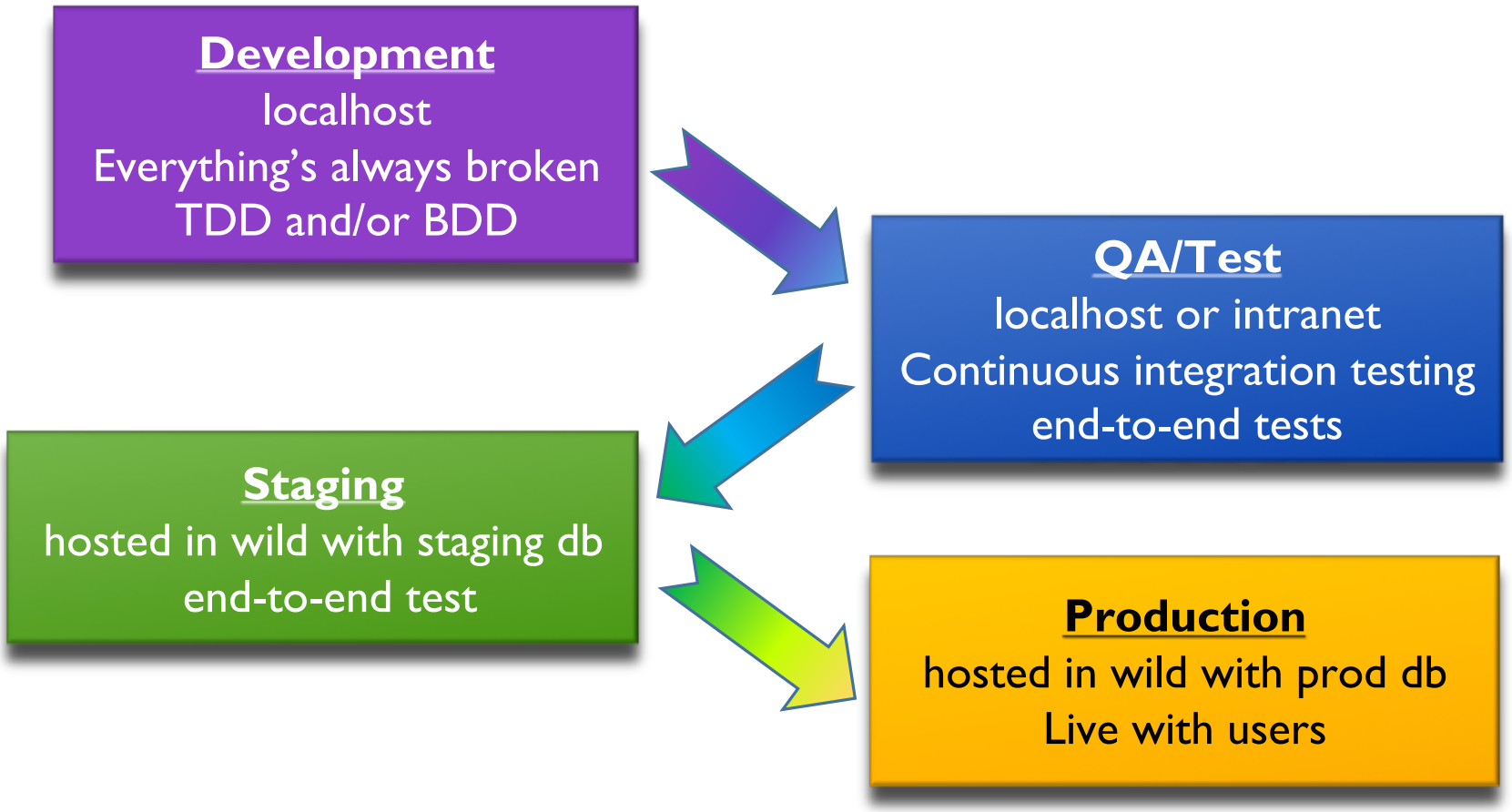
   >> ng serve --open
7. Navigate to *http://localhost:4200/players*

Submit player.component.html and player.service.ts to IC13 in Canvas

Development to Production



Development to Production



React Redux (Global Store) Tests

```
1     import tictactoeReducer, { selectSquare, requestPlayers } from "./tictactoeSlice";
2
3     test('test player winning game', () => {
4
5         let currState = tictactoeReducer(undefined, selectSquare({id: 1}));
6         expect(currState.board).toEqual(['', 'X', '', '', '', '', '', '', '']);
...
22    });
```

Unit test description

State initially undefined

Call reducer function to update state

React Testing with Redux

```
sudo npm install --save-dev vitest
```

```
tictactoe % sudo npm run test

> tictactoe@0.0.0 test
> vitest


 DEV  v3.2.4 /Users/mjoyner/comp431/F25/exercises/react/tictactoe

 ✓ src/features/game/tictactoeSlice.test.js (1 test) 2ms
   ✓ test player winning game 2ms

 Test Files  1 passed (1)
      Tests  1 passed (1)
   Start at  14:40:43
   Duration  214ms (transform 24ms, setup 0ms, collect 27ms, tests
vironment 0ms, prepare 49ms)

 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

Add test script to package.json

package.json

```
6       "scripts": {
7         "dev": "vite",
8         "build": "tsc -b && vite build",
9         "lint": "eslint .",
10        "test": "vitest",
11        "preview": "vite preview"
```


Code Coverage Testing in React

- Code coverage testing provides insight into what percentage of code is exercised by at least one unit test
  - if a function/line is not tested, you have no confidence it works!

- Run coverage testing:

```
tictactoe % sudo npm install -D @vitest/coverage-v8

tictactoe % sudo npm run test -- --coverage --run --reporter verbose
```

- Results are placed in a coverage directory

Code Coverage Testing in React

All files **src/features/game**

61.64% Statements `45/73` **61.53%** Branches `8/13` **60%** Functions `3/5` **61.64%** Lines `45/73`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
tictactoeSlice.tsx		86.53%	45/52	66.66%	8/12	75%	3/4	86.53%	45/52

Code Coverage Testing in React

All files / src/features/game tictactoeSlice.tsx

86.53% Statements 45/52 **66.66%** Branches 8/12 **75%** Functions 3/4 **86.53%** Lines 45/52

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1  1x  import { createSlice } from '@reduxjs/toolkit';
2
3  1x  export const tictactoeSlice = createSlice({
4  1x      name: 'game',
5  1x      initialState: {
6  1x          players: ['Joseph', 'Mary'],
7  1x          playerTurn: 'X',
8  1x          series: {xWins: 0, oWins: 0},
9  1x          status: 'Player turn: ',
10 1x          board: Array(9).fill("")
11 1x      },
```

Lines covered by test

Code Coverage Testing in React

```
12 1x      reducers: {
13 1x          selectSquare: (state, action) => {
14 1x              const id = action.payload.id;
15
16 1x              if (!state.board[id]) {
17 1x                  state.board[id] = state.playerTurn;
18
19                     // check if a player won the game
20 1x                  if (wonGame(state.board, state.playerTurn)) {
21                         (state.playerTurn === 'X') ? state.series.xWins++ : state.series.oWins++;
22                         state.board = Array(9).fill("");
23                     }
```

Lines not covered by test

Code Coverage Testing in React

```
24
25                  // check if the game is a draw
26 1x               else if (tieGame(state.board))
27 1x                   state.board = Array(9).fill("");
28
29 1x               state.playerTurn = (state.playerTurn === "X") ? "O" : "X";
```

Condition partially covered by test

Condition partially covered by test

Unit Testing in Angular with Jasmine

- TestBed simulates module environment
  - add modules, components, services needed by component
- Enables testing of individual components
  - test Angular framework behavior
- Could use Jasmine for React but there's other options
  - Jest/Enzyme, Mocha

Angular Root Component Unit Tests

app.component.spec.ts

```
1   import { TestBed } from '@angular/core/testing';
2   import { AppComponent } from './app.component';
3
4   describe('AppComponent', () => {
5     beforeEach(async () => {
6       await TestBed.configureTestingModule({
7         imports: [AppComponent],
8       }).compileComponents();
9     });
10
11    it('should create the app', () => {
12      const fixture = TestBed.createComponent(AppComponent);
13      const app = fixture.componentInstance;
14      expect(app).toBeTruthy();
15    });
16
17    it(`should have the 'tictactoe' title`, () => {
18      const fixture = TestBed.createComponent(AppComponent);
19      const app = fixture.componentInstance;
20      expect(app.title).toEqual('tictactoe');
21    });
22
23    it('should render title', () => {
24      const fixture = TestBed.createComponent(AppComponent);
25      fixture.detectChanges();
26      const compiled = fixture.nativeElement as HTMLElement;
27      expect(compiled.querySelector('h1')?.textContent).toContain('Hello, tictactoe');
28    });
29  });
```

Create test module that uses standard options (declarations, imports, providers)

A fixture is a wrapper for a component and its template

Dynamically detect fixture changes

Native element provides direct access to the DOM

KARMA

- **Karma is a test runner used to run tests in a browser environment**

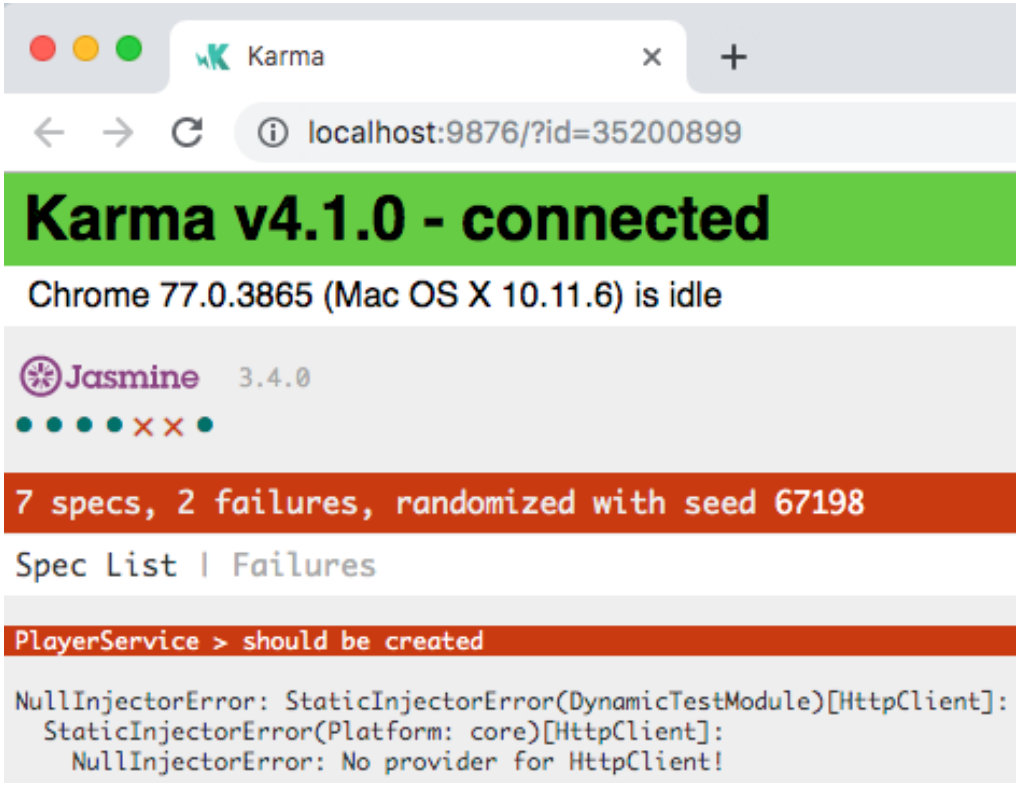
>> ng test

```
27 09 2015 21:44:21.127:WARN [karma]: No captured browser, open http://localhost:9876/
27 09 2015 21:44:21.158:INFO [karma]: Karma v0.13.10 server started at http://localhost:9876/
27 09 2015 21:44:21.221:INFO [launcher]: Starting browser Chrome
27 09 2015 21:44:29.937:INFO [Chrome 45.0.2454 (Windows 8.1 0.0.0)]: Connected on socket Z_nU9i_rt8KCs7
QCAAAA with id 86007040
Chrome 45.0.2454 (Windows 8.1 0.0.0): Executed 5 of 5 SUCCESS (0.164 secs / 0.121 secs)
```

- Opens a browser
- Use browser to execute code
  - run all tests
- Collect results
- Present results in terminal

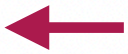
Karma / Jasmine Unit Testing in Angular

>> ng test



Player Service Unit Test

```
1  import { TestBed } from '@angular/core/testing';
2
3  import { PlayerService } from './player.service';
4
5  describe('PlayerService', () => {
6    let service: PlayerService;
7
8    beforeEach(() => {
9      TestBed.configureTestingModule({});
10     service = TestBed.inject(PlayerService);
11   });
12
13   it('should be created', () => {
14     expect(service).toBeTruthy();
15   });
16 });
```



Need to add providers for HttpClient

Player Service Unit Test Demo

Code Coverage Testing in Angular

- Code coverage testing provides insight into what percentage of code is exercised by at least one unit test
  - if a function/line is not tested, you have no confidence it works!

- Run coverage testing:

>> *ng test --code-coverage --source-map*

- Results are placed in a coverage directory

Code Coverage Testing in Angular

All files

74.47% Statements `35/47` **28.57%** Branches `4/14` **75%** Functions `12/16` **71.79%** Lines `28/39`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File ▲		Statements		Branches		Functions		Lines	
src		100%	3/3	100%	0/0	100%	0/0	100%	3/3
src/app		100%	3/3	100%	0/0	100%	1/1	100%	2/2
src/app/board		67.86%	19/28	33.33%	4/12	75%	6/8	62.5%	15/24
src/app/player		76.92%	10/13	0%	0/2	71.43%	5/7	80%	8/10

Code Coverage Testing in Angular

All files src/app/board

67.86% Statements 19/28 **33.33%** Branches 4/12 **75%** Functions 6/8 **62.5%** Lines 15/24

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
board.component.ts	47.06%	8/17	0%	0/8	50%	2/4	43.75%	7/16
board.service.ts	100%	11/11	100%	4/4	100%	4/4	100%	8/8

Code Coverage Testing in Angular

- Shows lines covered/not covered
- Indicates number of times line covered
- Can move *changePlayerTurn* to board service
- Function *handleBtnClick* should stay in component
- Need way to test handler for user event (e.g. button click)

```
 9 1x  export class BoardComponent implements OnInit {
10 1x    boardStatus: Array<string> = new Array<string>(9);
11       headline: string;
12       playerTurn: string;
13       winner: boolean;
14
15 1x    constructor(private bServ: BoardService) {
16 1x      this.boardStatus.fill("",0, 8);
17 1x      this.headline = "Player's turn: ";
18 1x      this.playerTurn = "X";
19 1x      this.winner = false;
20       }
21
22       ngOnInit() {
23       }
24
25       /**
26        * Handle the user selecting a square to put an X or an O.
27        * @param id  The square id
28        */
29       handleBtnClick(id: number) {
30         // check if player can select this square
31         if (!this.boardStatus[id] && !this.winner) {
32           this.boardStatus[id] = this.playerTurn;
33           this.winner = this.bServ.wonGame(this.playerTurn, this.boardStatus);
34
35           // check if the player made a winning move
36           if (this.winner)
37             this.headline = "Winner is ";
38           // alternate player turns
39           else
40             this.changePlayerTurn();
41         }
42       }
43
44       /**
45        *  Alternate player turns.
46        */
47       changePlayerTurn() {
48         if (this.playerTurn === "X")
49           this.playerTurn = "O";
50         else
51           this.playerTurn = "X";
52       }
```


Manual Application Testing

- Manual testing user interactions with the application can be tedious
  - Validating the registration information

- So far, we used unit tests to avoid manual testing
  - Karma is a test runner that runs all unit tests
  - Focus on individual component or function