Re:DOM

Starting Front Dev with RE:DOM

으로 시작하는 프론트개발

2020-01-18

이인용@UPnL

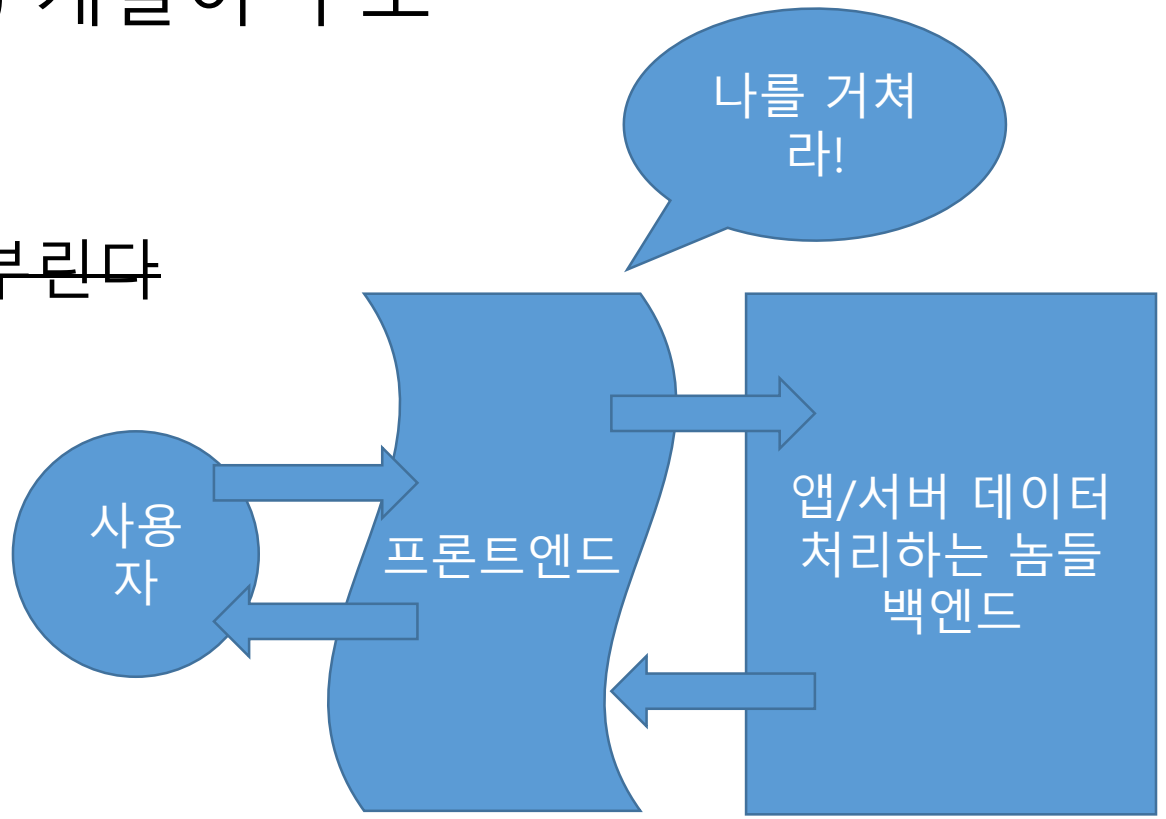
발표자 소개

- 이인용
- 컴퓨터공학부 14학번
- 14.09 정회원
- 유피넬 홈페이지 개발자(프론트 담당)
- 전전전전 와장
- 19.09 소집해제

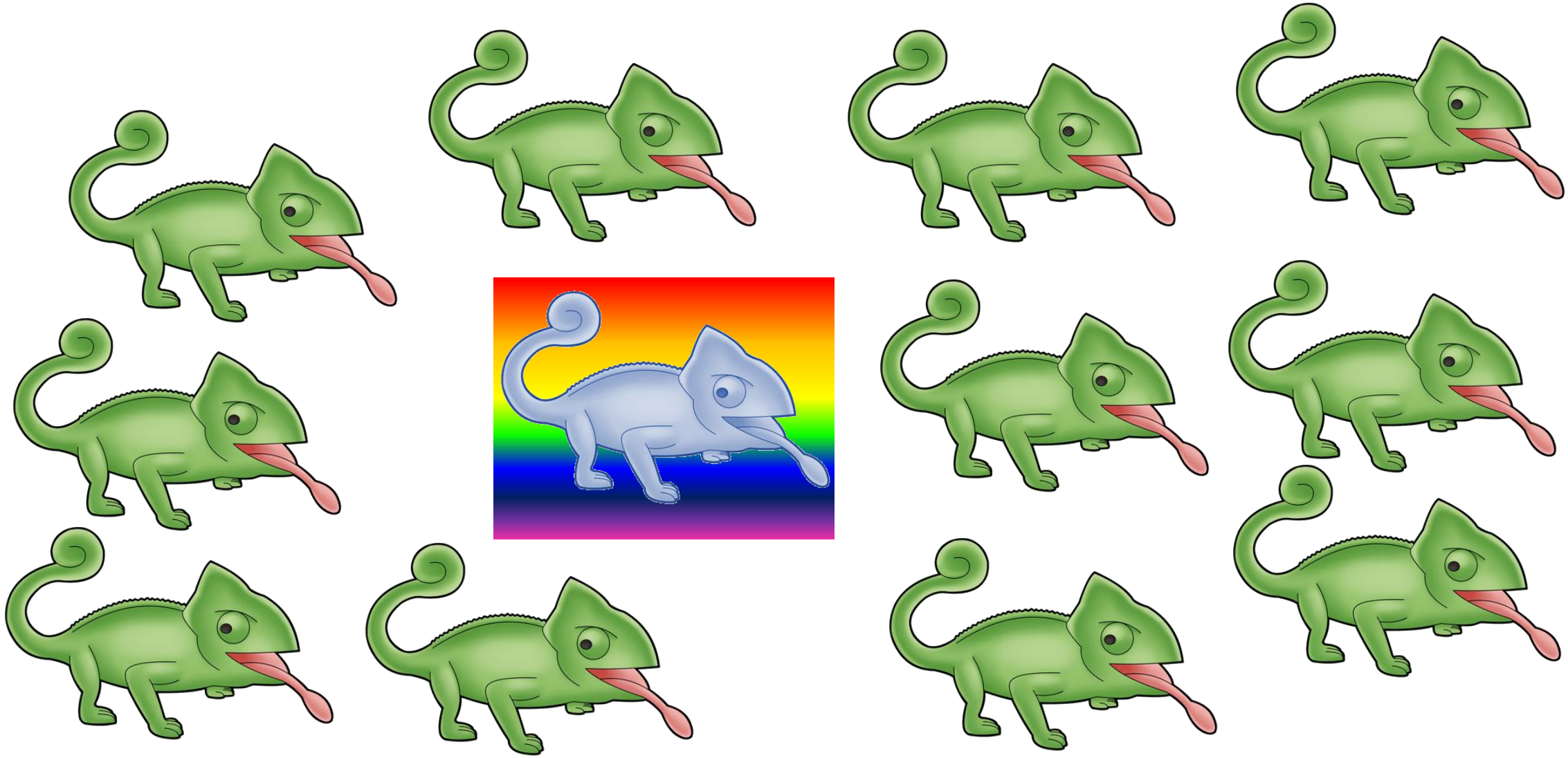
- 리제로는 뭔지 모름

프론트엔드!

- 프론트엔드란 무엇인가
- 주로 사용자의 인터페이스(UI) 개발이 주로
- 사용자랑 붙어있다?
  - 예쁘면 좋다……
  - 고객들을 상대한다 ~~→ 진상을 부린다~~
  - 쓰기 편해야 한다
  - 쉬워야 한다
  - ……



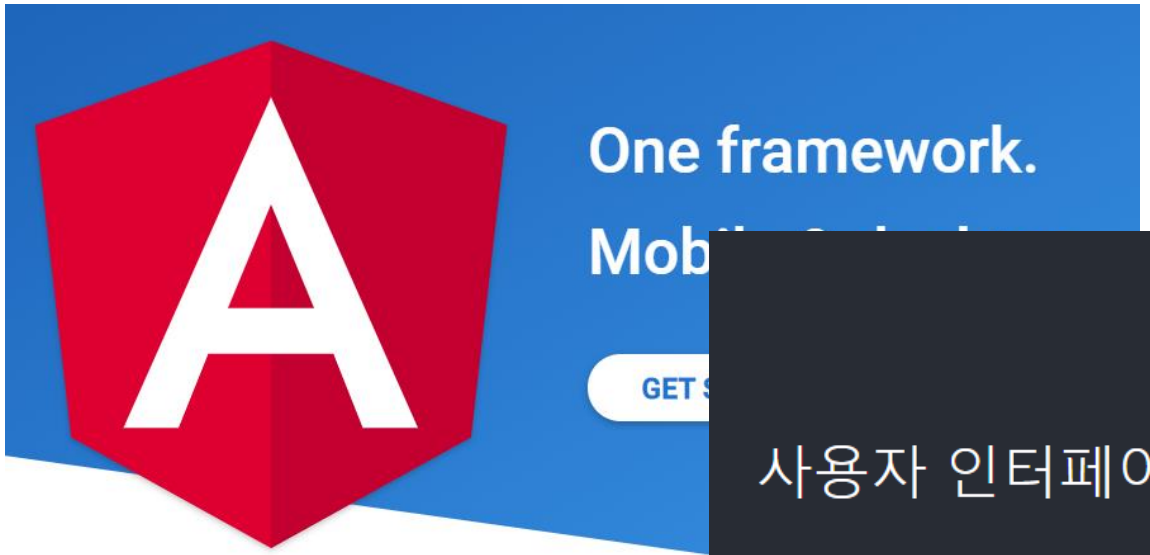
카멜레온을 골라보자



프레임워크란?

Vue.js가 무엇인가요?

Vue(/vju:/ 로 발음, **view** 와 발음이 같습니다.)는 사용자 인터페이스를 만들기 위한 **프로그레시브 프레임워크** 입니다. 다른 단일형 프레임워크와 달리 Vue는 점진적으로 채택할 수 있도록 설계하였습니



React

사용자 인터페이스를 만들기 위한 JavaScript 라이브러리

프레임워크란?

- 제어의 역전을 활용하여 사용자가 작성할 코드를 줄여준다
- 제어의 역전?
  - 몰라 찾아봐 용어는 중요하지 않아
  - 내가 짠 코드를 다른 프로그램/함수에게 줘서 다른 프로그램이 내 코드 돌리게 하는 거
  - 아무 코드나 주고 돌리게는 안 해주고, 정해진 인터페이스 등을 상속해야 돌리게 해줌
    - Runnable 넣어주면 돌려줄게
    - addEventListener('click', clickHandler); 클릭오면 돌려주겠음

프레임워크 예시

HTTP 서버를 작성하려면
포트 정해서 TCP 바인드
클라이언트 소켓 기다리고
연결되면
HTTP 프로토콜 파싱하고
세션도 가져오고
URL에 따라서
할 일을 한다
뭐 리턴하고
소켓 닫음

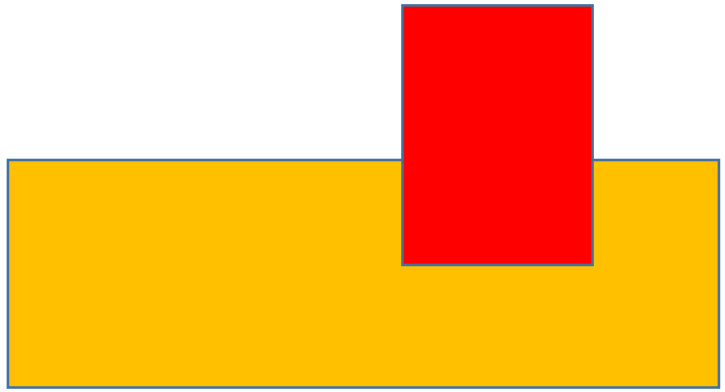
HTTP 서버 프레임워크로는
포트 정해서 TCP 바인드
클라이언트 소켓 기다리고
연결되면
HTTP 프로토콜 파싱하고
세션도 가져오고
URL에 따라서
할 일을 한다
뭐 리턴하고
소켓 닫음

계속 나오니까 우리가 해줄게요!

프레임워크와 라이브러리의 차이

- 내 코드 실행해주세요
- 내 코드를 큰 틀(프레임워크)에 끼워넣음
- 주객전도
- 언제 호출되는지?는 쟤가 정함

- 단순 함수 모음
- 결과 내놔라
- 호출 시점은 내 마음
- 함수니까...



프레임워크에 등록된 코드의 예시

```
@homepage.route("/next_season", methods=['POST'])
def next_season():
    if not session.__contains__('user_info'):
        return redirect(url_for('.login'))

    elif session['user_info'][0] == 'admin':
        seasons = [u"겨울", u"봄", u"여름", u"가을"]
        season = dao.query(Admin).filter_by(name='season').first()
        index = seasons.index(season.semester)
        if index == 0:
            season.year = season.year + 1
        season.semester = seasons[(index + 1) % 4]
        dao.commit()
        return redirect(url_for('.admin'))

    else:
        return redirect(url_for('.index'))
```

다음 시즌으로 가는 "그" 소스 코드

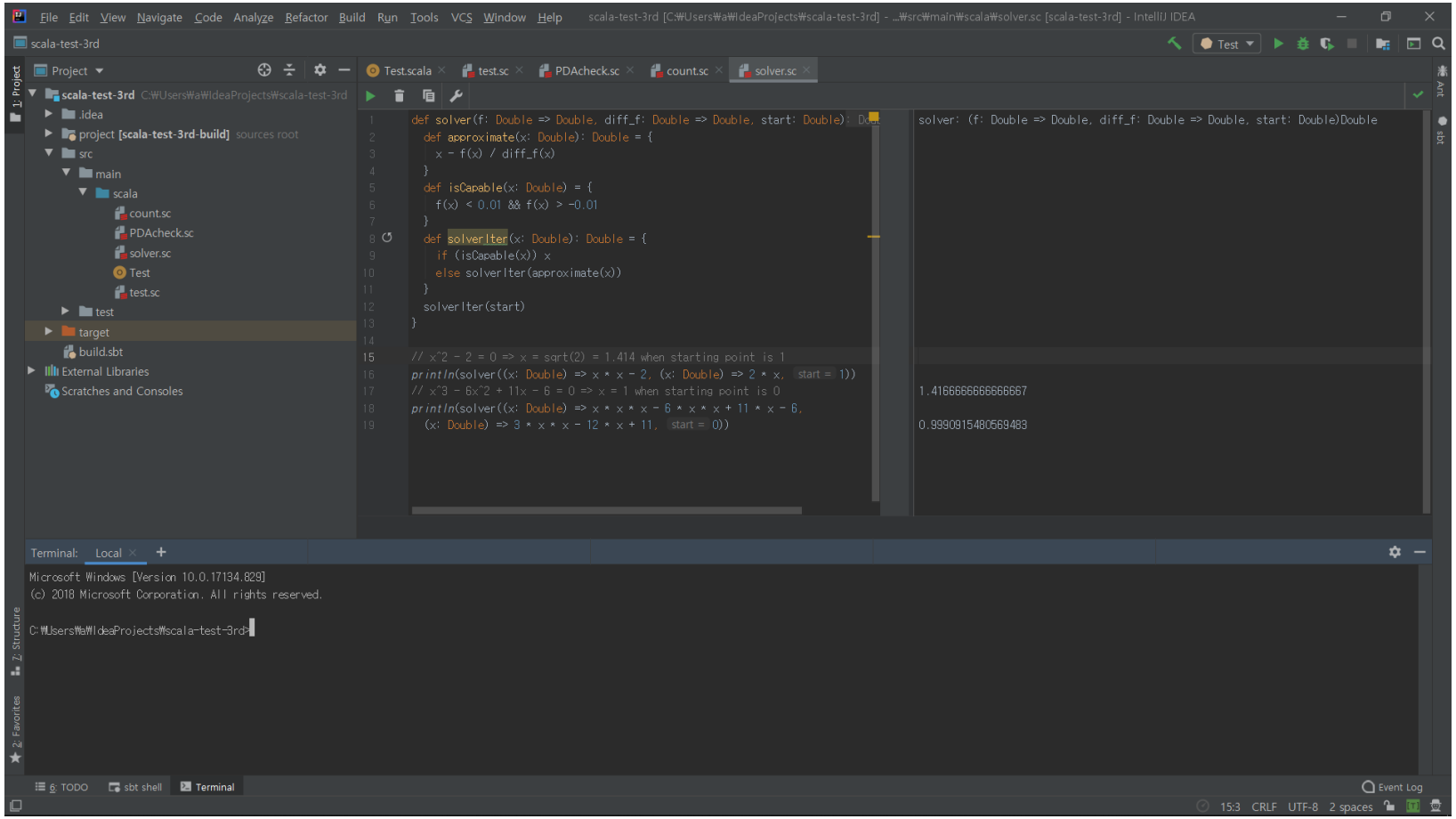
프론트엔드 개발이 하고 싶어요!

- 뭘 배워야 하나요?
  - 몰라
  - 개념 원리
  - HTML, CSS, Javascript
- 사실 뭘 만들고 싶느냐가 주가 되겠지만
- 발표가 삼천포로 빠지니까 대충 알려주고 넘어감

프론트엔드 구성 요소

- HTML
  - 뼈대, xml, <h1>큰 제목</h1> <button>버튼</button> <input/>
- CSS
  - button { width: 80px; height: 30px; background: red; }
- Javascript
  - button.onclick = function (event) { console.log('버튼 눌림'); };

오늘 만들 것


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
scala-test-3rd [C:¥Users¥a¥IdeaProjects¥scala-test-3rd] - ...¥src¥main¥scala¥solver.sc [scala-test-3rd] - IntelliJ IDEA
scala-test-3rd
Test
Project
Test.scala
test.sc
PDAcheck.sc
count.sc
solver.sc
scala-test-3rd C:¥Users¥a¥IdeaProjects¥scala-test-3rd
.idea
project [scala-test-3rd-build] sources root
src
main
scala
count.sc
PDAcheck.sc
solver.sc
Test
test.sc
test
target
build.sbt
External Libraries
Scratches and Consoles
def solver(f: Double => Double, diff_f: Double => Double, start: Double): Do
def approximate(x: Double): Double = {
x - f(x) / diff_f(x)
}
def isCapable(x: Double) = {
f(x) < 0.01 && f(x) > -0.01
}
def solverIter(x: Double): Double = {
if (isCapable(x)) x
else solverIter(approximate(x))
}
solverIter(start)
}
// x^2 - 2 = 0 => x = sqrt(2) = 1.414 when starting point is 1
println(solver((x: Double) => x * x - 2, (x: Double) => 2 * x, start = 1))
// x^3 - 6x^2 + 11x - 6 = 0 => x = 1 when starting point is 0
println(solver((x: Double) => x * x * x - 6 * x * x + 11 * x - 6,
(x: Double) => 3 * x * x - 12 * x + 11, start = 0))
solver: (f: Double => Double, diff_f: Double => Double, start: Double)Double
1.4166666666666667
0.9990915480669483
Terminal: Local
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.
C:¥Users¥a¥IdeaProjects¥scala-test-3rd>
6: TODO
sbt shell
Terminal
Event Log
15:3 CRLF UTF-8 2 spaces

오늘 만들 것

- IDE를 만들어보자
- 뻥이고 IDE같이 여러 패널이 있고, 윈도우가 있고, 서로 옮길 수 있는 것을 만들기

뭐부터 해야할까?

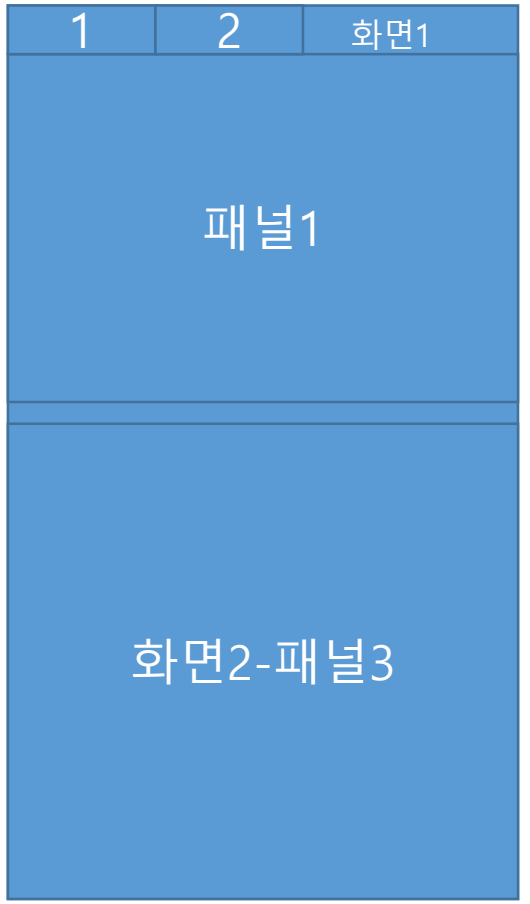
- 레이아웃부터 짜봐요!
- 디자인 모양을 CSS로 만들어봐요!

데이터 구조부터 짜라!

- 레이아웃 짜거나 CSS를 먹이는 건
  - 그런 일을 하는 사람이 있고(?) 퍼블리셔?라고 하던가
  - 아랫사람을 시키든가(??)
    - 물론 없으면 해야함ㅋㅋ
  - 디자인 라이브러리가 해주기도 하고
- 아무튼 일단 신경 쓰지말고
- 프론트엔드라고 들입다 HTML부터 짜지 말고 데이터 구조를 짜보세요
  - 뭐 만들지 작게 만들어보는 건 해야할 수도...

데이터 구조부터 짜라!

- 헤더 - 뭐 둘 수 없음
- 사이드바, 푸터
  - 어떤 화면을 넣어둘 수 있음, 열고 접을 수 있음, 사이즈 조절 가능
- 화면
  - 화면 안에 패널 여러 개가 탭으로 구성될 수 있음, 사이즈 조절 가능
- 패널
  - 화면 안의 도구



사이드바 예시

데이터 구조부터 짜라!

```
{
  right: {
    isFold: false, // 열렸니?
    windowHeights: [ // 고민 필요
      300,
    ],
    windows: [
      { type: 'window', name: 'window1',
        selectedPannelIndex: 0, // 무슨 패널 선택됐는지
        pannels: [
          { type: 'pannel', name: 'pannel1' },
          { type: 'pannel', name: 'pannel2' }
        ]
      },
      { type: 'window', name: 'window2',
        selectedPannelIndex: 0,
        pannels: [
          { type: 'pannel', name: 'pannel3' }
        ]
      }
    ]
  }
}
```

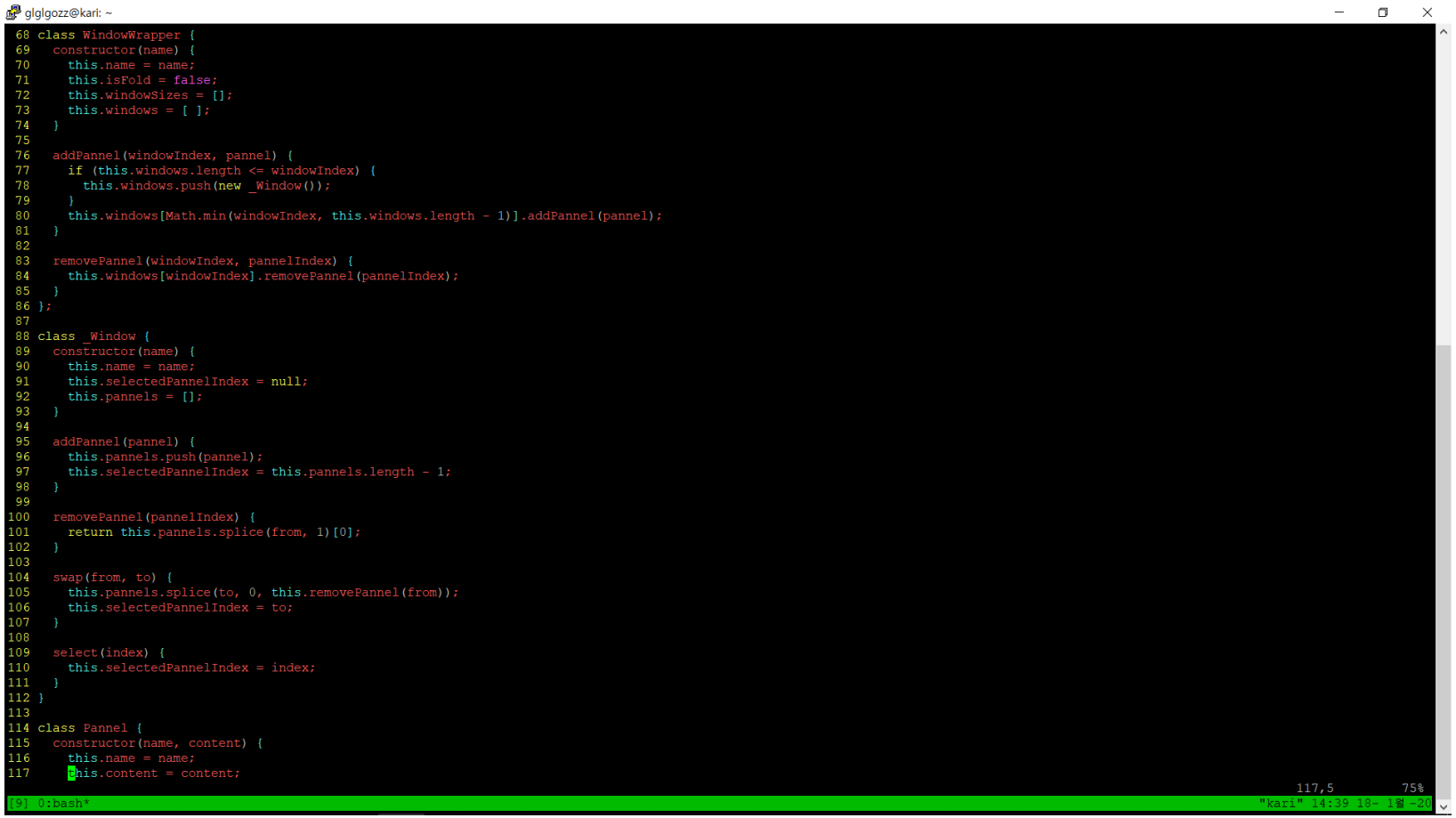
이 데이터를 변경하는 걸 짜야 함

- 화면 구조 변경하는 건 뭐가 있을까요?
  - 다른 패널 선택
  - 패널 → 다른 윈도우에 놓기
  - 패널 → 제 3의 윈도우로 놓기
  - 패널 → 다른 사이드바나 푸터로 옮기기
  - 열기 접기
  - 각 컴포넌트 사이즞 ㅗ절

데이터를 변경하는 방법을 기술해라!

- 일일이 함수로 만들어요
- 아 귀찮아
- 아랫사람 시키자
- 없잖아?
- ㅜ

데이터를 변경하는 방법을 기술해라!



시간이 없어ㅠㅠ TODO 리스트 하자

- 아까처럼 해주자
- 데이터 정의
  - { 할 일들: [{ 이름, 했음? }, { 이름, 했음? }] }
- 데이터 변경
  - n번째 했음, 안 했음 변경, 이름 변경, 할 일 추가

Todo 리스트 데이터 정의, 변경

```
class Todo {
  constructor(name) {
    this.name = name;
    this.done = false;
  }
  isDone() {
    return this.done;
  }
  toggleDone() {
    this.done = !this.done;
  }
  changeName(name) {
    this.name = name;
  }
}

class TodoList {
  constructor() {
    this.todos = [];
  }
  addTodo(name) {
    this.todos.push(new Todo(name));
  }
  removeTodo(index) {
    this.todos.splice(index, 1);
  }
}
```

```
var todoList = new TodoList();
function add(todoList, name) {
  todoList.addTodo(name);
}
function remove(todoList, index) {
  todoList.remove(index);
}
function done(todoList, index) {
  todoList.getTodo(index).toggleDone();
}
```

검증

```
var todoList = new TodoList();
add(todoList, '발표자료 만들기');
done(todoList, 0);
remove(todoList, 0);
```

- 잘 됨

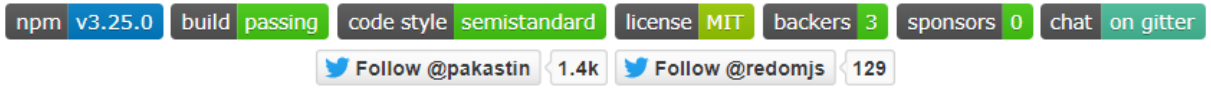
이제야!! UI를 만들 수 있어요

- 사실 UI는 만들었죠?
  - 함수 3개가 UI
  - 콘솔로 조작하면 됨
  - 일반 사람들이 쓰기 어려울 뿐
- UI는 껍데기일 뿐이다!
  - 반복 3번 하세요
- 불편하니까 GUI로 제공해주자

RE:DOM으로 만들자

- RE:DOM

Develop web applications with **100% JavaScript** and web standards. 🚀



RE:DOM is a tiny **(2 KB) UI library** by Juha Lindstedt and contributors, which adds some useful helpers to create DOM elements and keeping them in sync with the data.

Because RE:DOM is so close to the metal and **doesn't use virtual dom**, it's actually **faster** and uses **less memory** than almost all virtual dom based libraries, including React (benchmark).

It's also easy to create **reusable components** with RE:DOM.

Another great benefit is, that you can use just **pure JavaScript**, so no complicated templating languages to learn and hassle with.

RE:DOM - 기본적인 method

- method el
- el(tag이름, attr-object, 자식들) 온갖 것 넣기 가능

```
el('div#id',
   el('span.class', '안뇽'),
   el('input', { type: 'checkbox' })
); // 의 결과는 아래와 같이

<div id="id">
   <span class="class">안뇽</span>
   <input type="checkbox">
</div>
```

RE:DOM - component

- Node instance 를 el 프로퍼티로 가지는 Object
- el을 프로퍼티로 가지는 클래스를 만들어주자

```
function TodoView() {
  this.el = el('span', '아');
}
var todoView = new TodoView();
document.body.appendChild(todoView);
```

RE:DOM - mount

- 리돔에도 약간의 프레임워크 끼가 있어서 라이프사이클이 존재하긴 함
- append 를 직접하는 것보단 mount method를 사용해서 해주면 좋다

```
mount(document.body, todoView);
```

RE:DOM - list

- TodoList 처럼 데이터에 따라 동적으로 생성되는 것들을 위해 헬퍼로 list를 제공함. list 클래스는 update() 를 통해 data를 집어넣을 수 있으며 자세한 건 도큐먼트 참조
- 이 때 javascript의 assignment expression을 이용하여 재밌는 일을 할 수 있다

```
function TodoApp() {
  this.el = el('div',
    this.list = list('ul', TodoView), // this.list에 대입하면서 넘어감
  )
}
```

RE:DOM - update

- list 두번째 인자로 넘겨준 컴포넌트 클래스에 update method 를 적당히 추가해주면 데이터를 갖고 업데이트 가능
- 이벤트는 dom을 알고 있으니까 직접 핸들러 추가

```
const { el, list, mount } = redom;

function draw() {
  app.todoListView.update(todoList.getTodos());
}

function TodoView() {
  this.el = el('li',
    el('label',
      this.todoNameView = el('span.name'),
      this.checkbox = el('input', { type: 'checkbox' }),
    ),
    this.removeButton = el('button', '삭 제 ')
  );

  this.update = (todo, index) => {
    this.todo = todo;
    this.index = index;

    this.todoNameView.textContent = todo.name;
  };

  this.checkbox.onchange = event => {
    done(todoList, this.index);
  };

  this.removeButton.onclick = event => {
    remove(todoList, this.index);
    draw();
  };
}
```

```
const todoList = new TodoList();
function TodoApp() {
  this.el = el('div',
    el('h1', 'Todo List'),
    this.form = el('form',
      this.todoNameInput = el('input'),
      this.makeTodoButton = el('button', '추 가 ')
    ),
    this.todoListView = list('ul', TodoView)
  );

  this.form.onsubmit = event => {
    event.preventDefault();
    if (this.todoNameInput.value) {
      add(todoList, this.todoNameInput.value);
      this.todoNameInput.value = '';
      draw();
    }
  };
}

const app = new TodoApp();
mount(document.body, app);
```

마무리

- UI는 껍데기이다
- 프론트엔드에서도 데이터 구조를 잡는 것이 중요하다!
- 아까 만들었던 데이터 구조들은 이미 프레임워크들에 녹아있다
- Redux
  - Store -- 데이터 정의
  - Action & Reducer -- 데이터 조작 및 처리
- 처럼...
- 그래서... RE:DOM 써보실래요?