שבוע אורקל 2016

ברוכים הבאים
לשבוע אורקל 2016

ORACLE

JOHN BRYCE
תלמד הי-טק. זה עובד!
a matrix company

Maximize Your Oracle Experience

ישראל וג'ון ברייס הדרכה גאות להציג, זו השנה ה- 23, את האירוע המקצועי הגדול ביותר בישראל של קהילת ה- IT וההייטק בסביבת Oracle.

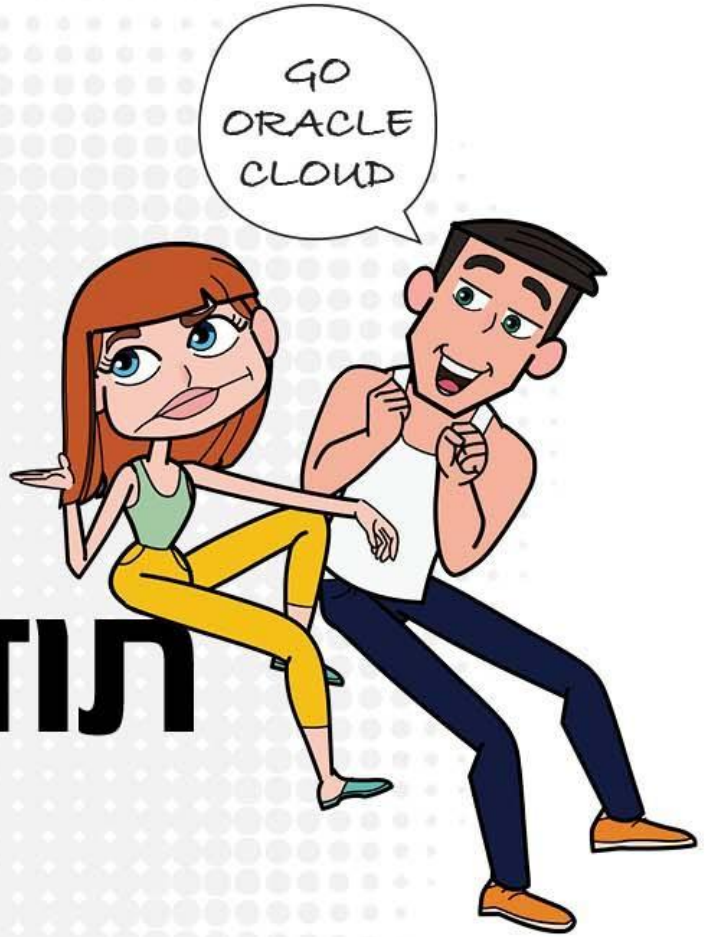
- ארוחת הצהריים מתקיימת בגן המלון.
- בשעה 15:00 תיערך הפסקה מתוקה במתחם קבלת הפנים.
- במהלך הסמינר נחלק משובים על השולחנות , נשמח לקבל את חוות דעתכם

בין ממלאי המשוב יוגרל מידי יום טאבלט!

אטרקציות במתחם ההתכנסות

- עמדת צילום: אתם רק צריכים לחייך, להצטלם ותהיה לכם תמונת פרופיל מקצועית.
- תערוכה ייחודית של צילומי ענגים.
- מדוושים, נהנים ומייצרים שייק פירות מרענן.
- משחקים בענן עם משקפיים וירטואליים.
- הגרלה יומית של מכשיר חכם לאימון יציבה. פרטים נוספים בדוכן UpRight.

שבוע אורקל 2016



תודה רבה!

שבוע אורקל 2016

The Future of Web Development

Angular2, ReactJS, NodeJS, BackBone

ORACLE

JOHN BRYCE
תלמד הי-טק. זה עובד!
a matrix company

Who am I?

Name: Yariv Katz
Email: yariv@nerdeez.com
Website: nerdeez.com
LinkedIn: https://www.linkedin.com/in/yariv-katz-38aaa023

What's cooking today?

- Web dev Past VS Future
  - Webpack - Setting our environment
  - ES6 / TypeScript
  - SPA and Universal apps
  - Past: MVC VS Future Unidirectional Data Flow
    - MVC with Backbone & Ember
    - Uni Directional Data Flow with ReactJS & Redux
  - Past: Promises VS Observables
    - Reactive X programming with Angular2

WebPack

Setting our environment

What is Webpack

- Our web application is made out of a lot of files
- Webpack takes all our application files and pack them to a distributable application
- Webpack can replace our Grunt or Gulp
- Webpack has a lot of plugins that adds a lot of extra features
- Things we can do with Webpack:
  - Transpile our ES6 or TypeScript code to Vanilla JS
  - Convert styling files: SCSS, SASS, Less to CSS
  - Minify our code
  - Upload our code to CDN
  - Deal with caching

Install webpack

- You can Install Webpack with **npm**

  > npm init

  > npm install webpack --save-dev

Webpack - Basic config

- Webpack expects a configuration file called **webpack.config.js**
- In the config file, the entry tells webpack which files to process
- In the config file, the output tells webpack where to output the files

שבוע אורקל 2016

Webpack - loaders

- The loaders can be used to tell webpack how to deal with certain files
- We will use **babel-loader** to deal with our **js** files and will transpile our **ES6** code to **Vanilla JS**
- We will use **ts-loader** to transpile our **TypeScript** files to **Vanilla JS**

ORACLE

JOHN BRYCE
תלמד הי-טק. זה עובד!
a matrix company

Webpack H.W

- Use webpack **UglifyJsPlugin** to minimize your code
- Use **extract-text-webpack-plugin** and **sass-loader, css-loader, style-loader** to create a single **CSS** file from all your **SCSS** files
- Use **S3-plugin-webpack** to upload you files to **CDN**
- **use file-loader** to handle the loading of all the font files and image files

Webpack H.W

- Use **image-webpack-loader** to minimize the size of your images
- Use **html-webpack-plugin** and create an **index.html** template and use webpack to create a hash for your files and create the index file with those hashes

Webpack summary

- Webpack is a must have tool in modern web applications
  - And we will use it in the rest of the lecture

שבוע אורקל 2016

The Future of JavaScript

ORACLE

JOHN BRYCE
תלמד ה-טק. זה עובד!
a matrix company

What is ES6

- ES6 is the significant update to JS since ES5 in 2009
- ES6 Support:
  - Desktop browser support

Chrome (54)	Firefox (52)	IE 14 Edge	Safari 10
97%	94%	93%	100%

  - mobile browser support

Android 5.1	iOS 10
25%	100%

What is TypeScript

- JavaScript extension created by Microsoft
- Not supported by browsers, has to be compiled to JavaScript
- Add statickness to a dynamic language
- Angular2 created with TypeScript and they are pushing their users to use it

Past JavaScript

Variable Declaration

What will the following code print?

```
for(var count=0; count<10; count++){
    for(var count=0; count<10; count++){
        console.log(count);
    }
}
```

and this one?

```
function printHello(isPrint) {
    if(isPrint){
        var message = 'hello world';
    }
    console.log(message);
}
printHello(true);
printHello(false);
```

Future ES6 & TypeScript

Variable Declaration - Const

const <var name> = <var value>

Example:
const helloObject = {'hello': 'world'}

Const has to be initialized once when it's declared, this will present an error:
helloObject = 'new value in const var = error'

Note that const vars are mutable, you can still do this:
helloObject['hello'] = 'pikachu';

Future ES6 & TypeScript

Variable Declaration - let

let <var name> = <var value>

Example:
let helloObject = {'hello': 'world'}

let can be initialized more than once and it's scope is inside the code block it's defined in

```
if(isPrint){
    let message = 'hello world';
}
console.log(message) //error
```

Future ES6 & TypeScript

Class

```
class <name of class> {
    constructor(){
        ...
    }
}
```

ES6

```
class Pokemon {
    constructor(name = 'pikachu') {
        this.name = name;
    }
}
```

TypeScript

```
class Pokemon {
    constructor(public name : string = 'pikachu') {}
}
```

Past JavaScript

Class

How did we define a class with Vanilla JS?

```
function Pokemon(name){
  this.name = name || 'pikachu';
}

var pikachu = new Pokemon();
console.log(pikachu.name);
```

Future ES6 & TypeScript

Inheritance

To inherit from a class we need to use the **extends** keyword

```
class Pikachu extends Pokemon{
  constructor(){
    super();
  }

  sayHello(){
    console.log(this.name + ' said hello');
  }
}

const pikachu = new Pikachu();
pikachu.sayHello();
```

Past JavaScript

Inheritance

How do we Inherit in good old JavaScript ?

```
function Pokemon(name){
  this.name = name || 'pikachu';
}

function Pikachu(){
  Pokemon.call(this);

  this.sayHello = function sayHello(){
     console.log(this.name + ' said hello');
  }
}

var pikachu = new Pikachu();
pikachu.sayHello();
```

Past JavaScript

Function

What will be printed in the following code?

```
function Pokemon2(name, age) {
  this.name = name || 'pikachu';
  this.age = age || 10;

  this.birthday = function(){
    setTimeout(function(){
      this.age++;
    }, 1000)
  }
}

var pikachu = new Pokemon2();
console.log(pikachu.age);
pikachu.birthday();
setTimeout(function(){
  console.log(pikachu.age);
}, 2000);
```

this tends to get lost, how do you solve this problem?

Future TypeScript/ES6

lambda functions

Lambda functions **this** is set when the function is defined.

Syntax:

```
(args) => {
        ...
}
```

if we have one line return:

```
args => 3
```

```
function Pokemon2(name : string = 'pikachu', age : number = 10) {
  this.name = name;
  this.age = age;

  this.birthday = function(){
     setTimeout(() => {
        this.age++;
     }, 1000)
  }
}

var pikachu = new Pokemon2();
console.log(pikachu.age);
pikachu.birthday();
setTimeout(function(){
  console.log(pikachu.age);
}, 2000);
```

Future TypeScript/ES6

Modules

- you can now easily split your project file
- the variables defined in the file are only available on that file
- use **export** and **import**
- Example

Export:

```
export class Pokemon {
  constructor(public name : string = 'pikachu') {}
}
```

Import:

```
import {Pokemon} from './pokemon.ts';

const pikachu = new Pokemon();
console.log(pikachu.name);
```

Past JavaScript

Modules

How do we do that in JavaScript?

TypeScript

TypeAnnotation

```
const stringVar : string = 'hello world';

const unknownType : any = {'hello': 'world'}

const stringArray : Array<string> = []

function retBoolean() : boolean {
}

function retClassPokemon() : Pokemon{
    return new Pokemon();
}

interface Pokemon2 {
    name : string;
    age? : number
}

const pikachu : Pokemon2 = {name: 'pikachu'}
```

JS/TypeScript/ES6

Summary

- Embedding future languages in your project is made easy with webpack
- When OOP features are easy to use in a language you will notice that everyone in the dev team uses them
- Using advanced languages make your code more understandable and scalable

שבוע אורקל 2016

Single Page Applications (SPA)

Future or Past?

ORACLE

JOHN BRYCE
תלמד ה'-טק. זה עובד!
a matrix company

What is SPA?

- One HTML file is loaded to the user along with all our application resources (JS, CSS, Images, Font Files ...)
- Using Ajax based technology all application content is dynamically injected to our single HTML file
- no page reloading
- The SPA application usually communicates with a server to get data for the application.
- The lack of page reloading make our app UX much better

The challenges of SPA

- Creating SPA application can be challenging:
  - We need a templating engine to render our server data to HTML
  - We need a router to deal with the URL
  - We need to take care of the application architecture (MVC, Components based unidirectional)
  - We need to deal with Server Communication
  - Form Validation

- To help us with these challenges we can use one of the many SPA frameworks

SPA Past or Future?

- SPA sounds AWESOME!
- No page reloading means better UX
- So SPA gotta be the future right!?
- SPA comes with some problems
  - SPA is less friendly to web crawlers
  - Initial loading speed is slow especially for mobile devices

The Future: Universal App

- What is Universal App?
  - Our application can run both on the browser client side, and the backend server side.
- New Frameworks like React and Angular2 helps us create Universal app
- When user requests to load our application this is what happening:
  - The server runs our application and create from the application files the fully rendered HTML page of the application (not a page with just script tags)
  - The user get a fully rendered page and from that point client side rendering takes control

Why Universal app

- the first rendering is done in our server and we can know for sure that the users will get our app loaded quickly, even on mobile devices
- after that the user enjoys the UX advantages of SPA

The down side of Universal app?

Can you guess what it is ?

Summary

- SPA are the future of web development but it has some problems
- If we want a faster loading web application we need to think about Universal app/ Server side rendering / Isomorphic app

שבוע אורקל 2016

NodeJS & Express

Building our REST Server

ORACLE

JOHN BRYCE
תלמד ה'-טק. זה עובד!
a matrix company

REST Server

- To create our todo application we will have to serve application data from a database
- Our SPA can’t connect directly to a database
- We will have to create a server that will connect to the database
- The SPA will talk to the server and the server will talk to the database
- The recommended way of client server communication is via REST

What is REST?

- REST - Representational State Transfer
- Communication convention between client and server
- Server is connected to a data source and from the client request understands how to manipulate the data source

Client request - method

By request method the server knows what we want to do:

- **GET -** read data
- **POST -** create new data
- **PUT -** modify existing data
- **DELETE -** delete data

Client request - url

- Using the url the server knows which data you want to perform the action on
- In our server app accessing the URL **/api/task** will access the task collection.
- The URL **/api/task/<Id>** will perform the action on a specific row in the collection with that Id

Client request - body

We use the request body for **POST** and **PUT** request to send the servers the fields of the item we want to insert or update

Our REST server

- Our rest server will contain a **GET** and **POST** methods in the URL **/api/task** to create a new task and read all the tasks in the collection.
- Our server will be connected to a local Mongo Database

Mongo - Installation

- We will start with installing Mongo:

Windows:

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

Mac:

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/

Mongo - Daemon

- Start the Mongo daemon by typing

  > mongod

- Activate mongo by typing

  > mongo

Mongo - db & collection

- create a new db by typing

  > use todo

- collection is created when you insert a new document

  > db.tasks.insert({title: …, description: …
  date: new Date()})

mongo - find

- we can print all the tasks by typing:

  > db.tasks.find()

- we can look for specific task by typing:

  > db.tasks.find({title: ‘test1’})

Node Express
Installing packages

- We will start by creating a server that print hello world
- First let's start **npm** and install the needed packages

  > npm init

  > npm install express --save

Express - Hello world

Create a new file that will start the server called **app.js**

```
// create express app
var express = require('express');
var app = express();

// create hello world in all routes
app.use('*', function(req, res){
  res.send('hello world');
});

// start the server
app.listen(3000, function(){
  console.log('starting our server...');
})
```

Express - Connect to DB

- let's connect to our DB and print all the tasks
- We will use **Mongoose** to help us connect to Mongo
- Everything in **Mongoose** start with a schema that describes a collection in the DB
- From the Schema we can create a model that will allow us to perform action with the collection

Mongoose schema & model

```
//import
var mongoose = require('mongoose');

// create schema
var TaskSchema = new mongoose.Schema({
  title: String,
  description: String,
  date: Date
});

// create model
mongoose.model('Task', TaskSchema);
var Task = mongoose.model('Task');

// export model
module.exports = Task;
```

mongoose - app.js

- let's connect mongoose to the database and use the model to grab the tasks

```
// connect to the db
var mongoose = require('mongoose');
var db = mongoose.connect(process.env.DATABASE_URL);
var Task = require('./models/Task.model.js');

// fetch all records in all routes
app.use('*', function(req, res){
   Task.find(function(err, data){
      res.send(data);
   })
});
```

Express Routes - GET

Let's move the get all tasks to a new express route

```
// create a new router
var express = require('express');
var router = express.Router();
var Task = require('../models/Task.model.js');

/**
* send all the tasks in the collection
*/
router.get('/task/', function(req, res){
   Task.find(function(err, data){
       res.send(data);
   });
})
```

Connect router to app

Now to connect the router to our main express app:

```
// fetch all records in all routes
var router = require('./routes/task.route.js');
app.use('/api', router);
```

Express - bodyParser

- To create the **POST** api we will have to attach the task properties to the body of the request
- **bodyParser** is a middleware the place the post body in a dictionary in the request

```
// add body parser so the post put params will be availble in the body
var bodyParser = require('body-parser');
app.use(bodyParser.json());
```

Router - POST

Let's create the API for creating a new task

```
router.post('/task/', function(req, res){
  var task = new Task({
      title: req.body.title,
      description: req.body.description,
      group: req.body.group,
      date: new Date(req.body.date)
  })
  task.save(function(err, data){
      res.send(data);
  })
});
```

Integration Tests

- Integration test means running the api we just created in a test
- How can we verify that our test data will be consistent
  - Use a test database
  - wipe the database clean before each test
  - load fixtures as the data of the test

Integration tests clean DB

```
var DatabaseCleaner = require('database-cleaner');
connect = require('mongodb').connect;

module.exports = function cleanDb(callback){
  var databaseCleaner = new DatabaseCleaner('mongodb');
  connect(process.env.DATABASE_URL, function(err, db) {
    databaseCleaner.clean(db, callback);
  });
};
```

Integration Tests
Loading Fixtures

```
var fixtures = require('pow-mongodb-fixtures').connect(process.env.DATABASE_URL);

/**
* will load all the fixtures in the array
* @param {string[]} arrFixtures absolute path to fixture files
* @param {Function} cb callback after all fixtures are resolved
*/
module.exports = function loadFixtures(arrFixtures, cb) {
  var promises = [];
  for(var i=0; i<arrFixtures.length; i++){
     var p = new Promise(function(resolve, reject){
        fixtures.load(arrFixtures[i], function(){
           resolve();
        });
     });
     promises.push(p);
  }
  Promise.all(promises).then(cb);
}
```

Fixture

```
var id = require('pow-mongodb-fixtures').createObjectId;
exports.tasks = {
  task1: {
      _id: id('000000000000000000000000'),
      title: 'task1',
      description: 'task1 description',
      group: 'yariv-katz',
      date: new Date()
  },
  task2: {
      _id: id('000000000000000000000001'),
      title: 'task2',
      description: 'task2 description',
      group: 'yariv-katz',
      date: new Date()
  }
}
```

Test file

```
beforeEach(function(done){
  cleanDb(function(){
      loadFixtures([path.join(__dirname,
'../fixtures/task.fixture.js')], function(){
          done();
      });
  });
});
```

Test file

```
it('should get all the tasks on get /task/', function(done){
   chai.request('http://localhost:3000')
      .get('/api/task/')
      .end(function(err, res){
         expect(res.body.length).to.equal(2);
         done();
      });
});
```

Test file

```
it('Should create a new task when doing a post', function(done){
   chai.request('http://localhost:3000')
      .post('/api/v1.0/task/')
      .send({
         'title': 'test1',
         'description': 'test1',
         'group': 'test1',
         'date': new Date()
      })
      .end(function(err, res){
         chai.request('http://localhost:3000')
            .get('/api/v1.0/task/')
            .end(function(err, res){
               expect(res.body.length).to.equal(3);
               done();
            });
      });
});
```

Node Express H.W

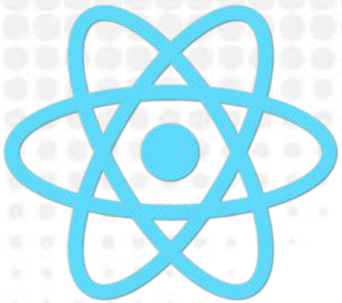
- Create api for **PUT** request in the URL: **/api/task/<id>/** that will update a document in the task collection
- Create api for **DELETE** request in the URL: **/api/task/<id>/** that will delete a document in the task collection
- Create Integration test for all your api methods

Node Express Summary

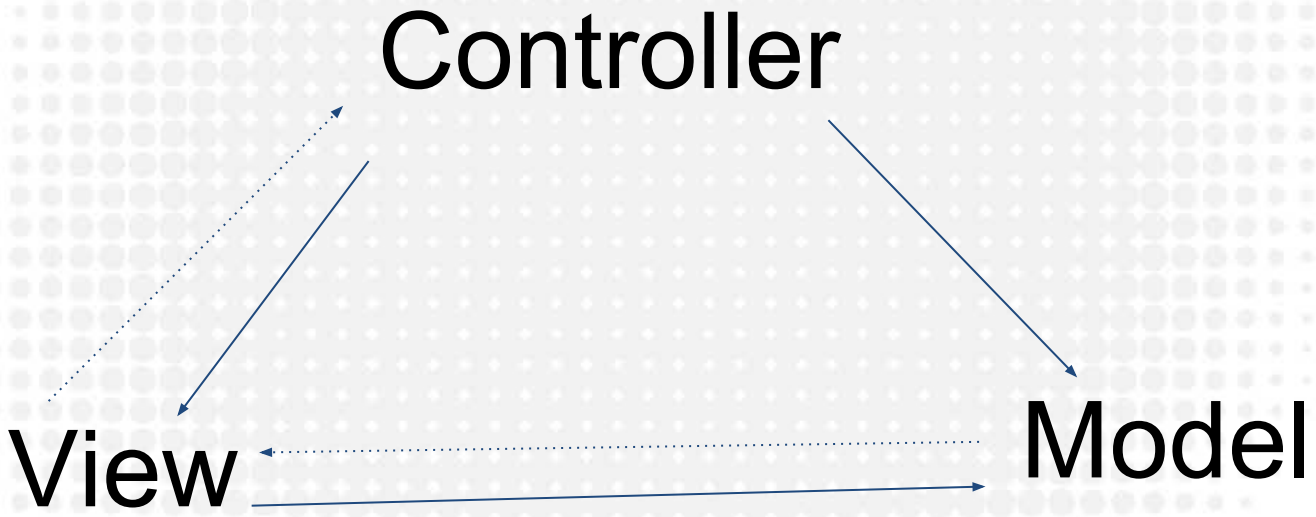
- Creating a REST server with Node & Express is straightforward and easy
- After you get used to writing tests it's faster than checking with other tools
- Testing your api is crucial after the rest server becomes more complicated
- Adding tests improve your app delivery with Jenkins and CI tools.

Past: MVC with Backbone

Future: Unidirectional data flow with React



Past MVC
What is MVC



MVC in Backbone

Lets see how MVC looks like in a backbone application

What is Backbone

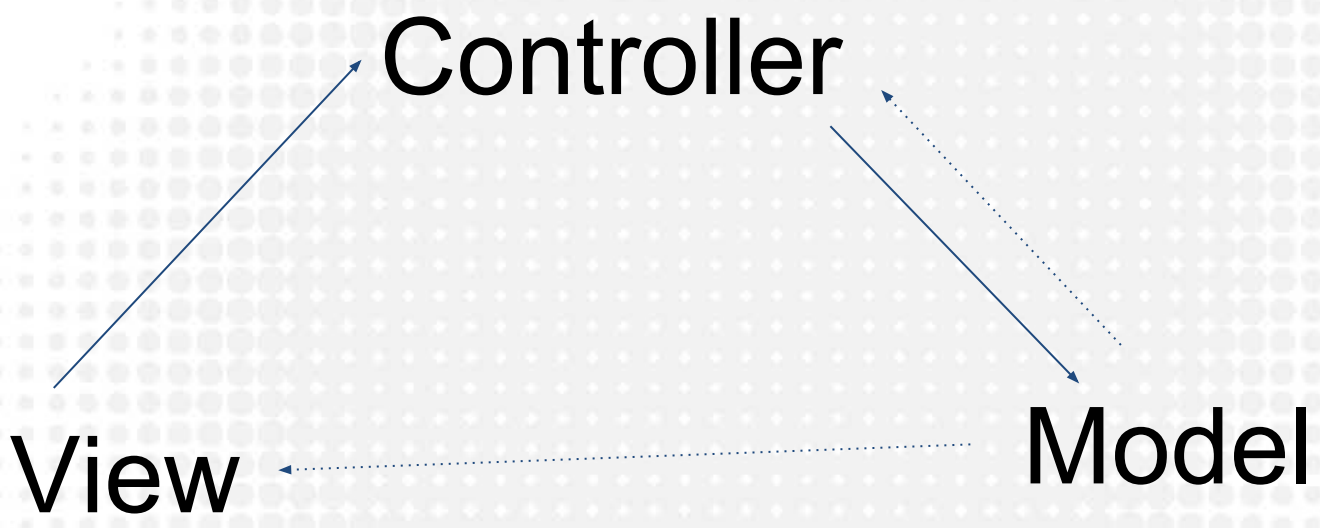
- Minimal frontend library
- Minimal footprint
- Does not enforce us to create our app a certain way

Backbone Architecture

Backbone has two main players:

- Model - Represent the data layer from the server, keeping data from server in sync, get changes of data from view and emits the change to listeners
- View - In charge of UI and changing the model, more resembles a controller in MVC

MVC in Backbone

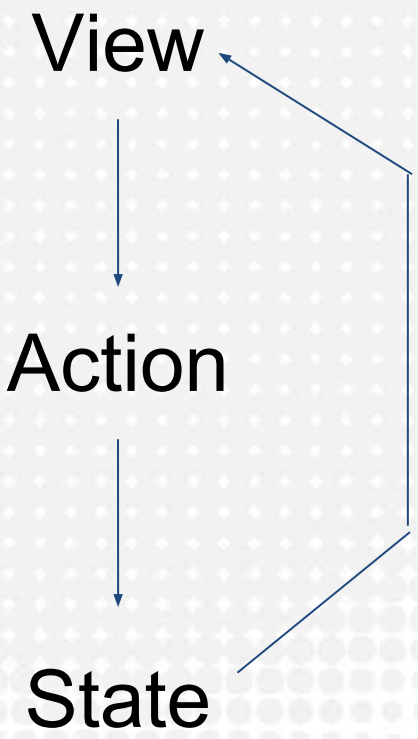


Backbone Demo

We will implement Backbone architecture by doing a small demo application

- We will create a Task Model that will connect to the REST server we created earlier
- We will create a view to display the task list

Future - unidirectional data flow



Introducing React

- Open source project maintained by facebook
- React is a framework to create the view component the user see and interact with
- Each view component gets state and properties
- When state is changing component is rerendered
- Change to the DOM is via the Virtual DOM

React hello world

Let's practice React by creating some assignments

- Create an hello world component
- make the hello world component get a property of the greeting
- make a form in the hello world with a text input to write the greeting

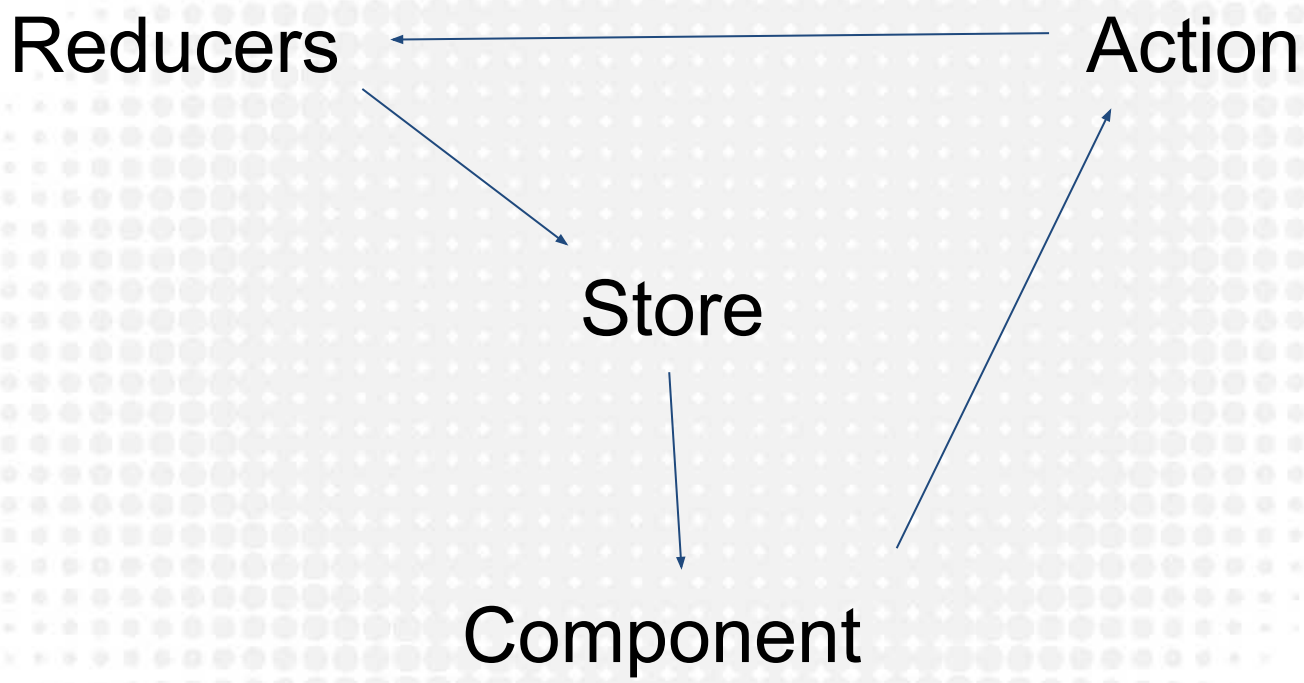
React Redux

- Components in our app usually affect each other
- State change in one component can change the look of another component
- We need a framework to manage the state of our entire app
- Enter Redux

What is Redux

- Redux define a single state object for the entire app stored in the Redux store
- The state is read only and can only be change by calling **store.dispatch(action)**
- An action is a simple object that looks like this {type : string, payload : any}
- The store then use the Reducer to determine the next state

Redux architecture



Our first Redux app

We will now build our first Redux application

Our application will have a form to create a new todo item and a list to display all the todo tasks.

React H.W

- Add **redux-thunk** middleware and try to create a call to the server that change the state of our application
- Try to include multiple reducers in your app and combine them together

Summary
Unidirectional Data Flow

- Unidirectional data flow is an application architecture widely adopted by react applications
- The architecture scales much better than MVC
- The architecture is better testable than MVC

Past promises

Promises are objects that get resolved or rejected in the future

Promise is one of the main building blocks in async actions

Sample promise

```
)var promise = new Promise
function(resolve, reject){
                )setTimeout
              ,{()function(){resolve
                        3000
                            (
                              {
                                (
```

Sample promise in backbone

Lets try to practice backbone and promises to grab the todo items from the servers we created.

Future Observables

- New pattern for dealing with async events
- Open source project called RXJS, and will be part of ES7
- The pattern defines an Observable that can emit different events
    - On the other side the pattern defines an observer which listens to a certain event

Sample Observable

```
import { Observable } from 'rxjs/Observable';
const observable = Observable.create((observer) => {
   setTimeout(() => {
       console.log('!!!!');
       observer.next();
   }, 5000);
});
observable.subscribe(function(){
   console.log('Im resolved');
});
```

Angular2 Observables

- let’s try to use Angular2 with Observables to search our todo server

Thank You!

שבוע אורקל 2016

ORACLE

JOHN BRYCE
תלמד הי-טק. זה עובד!
a matrix company