Full Stack Reactive with React and Spring WebFlux

Matt Raible | **@mraible**

March 12, 2020

{okta}

Photo by **Lukas Schlagenhauf** flickr.com/photos/lschlagenhauf/33223388493

GITHUB.COM/JOSHLONG/bootiful-reactive-microservices

Reactive Revolution

龍之春 Джош Ջոշ Լոնգ

Josh Long

जोश ג'וש לונג ジョシュ・ロング

Spring Developer Advocate

TWITTER @starbuxman

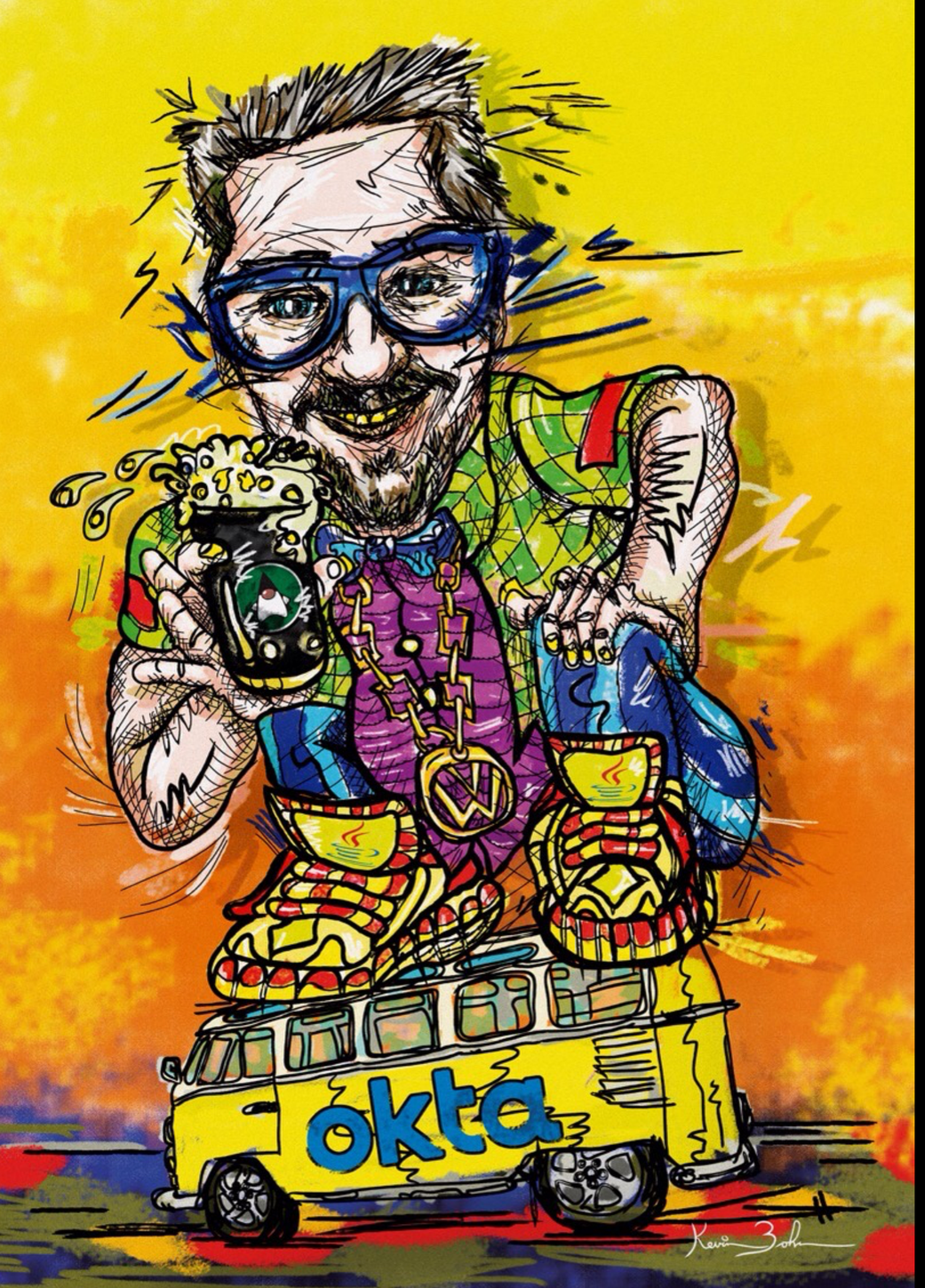
E-MAIL josh@joshlong.com

spring Pivotal

book (2):

MY NEW BOOK (COMING SOON!)

ReactiveSpringBook.io



Who is Matt Raible?

Father, Skier, Mountain Biker,
Whitewater Rafter

Open Source Connoisseur

Web Developer and Java Champion

Okta Developer Advocate

Bus Lover

Blogger on raibledesigns.com and
developer.okta.com/blog


PHILLIPS 66
The UPS Store
perry nails
DUGOUT
DENTIST

developer.okta.com


What About You?

Full Stack Reactive

Full Stack Reactive with Spring WebFlux, WebSockets, and React

Matt Raible

Spring WebFlux can be used to create a REST API with streaming data. Spring WebFlux can also be integrated with WebSockets to provide notifications that clients can listen to. Combining the two is a powerful way to provide real-time data streaming to JavaScript or mobile clients. Add React to the mix and you have an excellent foundation for a full-stack reactive architecture.

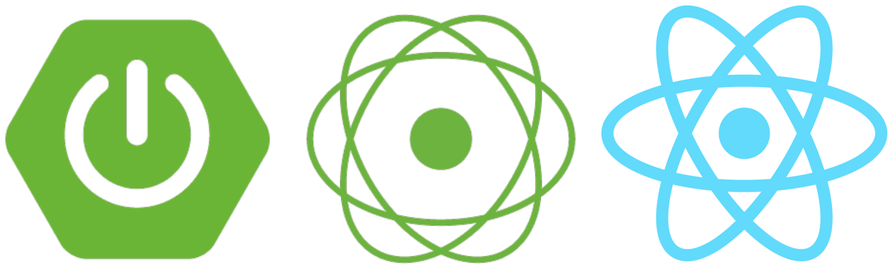
React is a UI toolkit (similar to GWT) that lets you build components with JavaScript (or TypeScript) and JSX. JSX is how you define elements in React and it looks very similar to XML. React's API and JSX are the core of the framework; everything else is an add-on. I won't go into nitty-gritty details about React, we'll assume you've heard of it and are eager to learn how to make it even *more* reactive!

I teamed up with Josh Long to write this post. Josh is a fellow Java Champion, Spring Developer Advocate, and all around excellent guy at Pivotal. Josh and I've been good friends for a while now, sharing the same passion for Java, developers, and building bleeding-edge applications. We hope you like this series!

Today, we'll show you how to build a *full-stack* application using Spring WebFlux for the API, WebSockets for notifications, and React for the UI. This article is the third in a three-part series about reactive programming and Spring WebFlux. The first two are listed below.

1. Get Started with Reactive Programming in Spring
2. Build Reactive APIs with Spring WebFlux

http://bit.ly/webflux-and-react



Today’s Agenda

What is reactive programming?

Introduction to Spring WebFlux

Developing an API with WebFlux

Handling Streaming Data with React

Securing WebFlux and React

What is reactive programming?

Asynchronous I/O

```java
package com.example.io;

import lombok.extern.log4j.Log4j2;
import org.springframework.util.FileCopyUtils;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.function.Consumer;

@Log4j2
class Synchronous implements Reader {

    @Override
    public void read(File file, Consumer<BytesPayload> consumer) throws IOException {
        try (FileInputStream in = new FileInputStream(file)) {
            byte[] data = new byte[FileCopyUtils.BUFFER_SIZE];
            int res;
            while ((res = in.read(data, 0, data.length)) != -1) {
                    consumer.accept(BytesPayload.from(data, res));
            }
        }
    }
}
```

```
class Asynchronous implements Reader, CompletionHandler<Integer, ByteBuffer> {

    private int bytesRead;
    private long position;
    private AsynchronousFileChannel fileChannel;
    private Consumer<BytesPayload> consumer;
    private final ExecutorService executorService = Executors.newFixedThreadPool(10);

    public void read(File file, Consumer<BytesPayload> c) throws IOException {
        this.consumer = c;
        Path path = file.toPath();
        this.fileChannel = AsynchronousFileChannel.open(path,
            Collections.singleton(StandardOpenOption.READ), this.executorService);
        ByteBuffer buffer = ByteBuffer.allocate(FileCopyUtils.BUFFER_SIZE);
        this.fileChannel.read(buffer, position, buffer, this);
        while (this.bytesRead > 0) {
                this.position = this.position + this.bytesRead;
                this.fileChannel.read(buffer, this.position, buffer, this);
        }
    }

    @Override
    public void completed(Integer result, ByteBuffer buffer) {
        ...
    }
}
```

```java
@Override
public void completed(Integer result, ByteBuffer buffer) {

    this.bytesRead = result;

    if (this.bytesRead < 0)
        return;

    buffer.flip();

    byte[] data = new byte[buffer.limit()];
    buffer.get(data);


    consumer.accept(BytesPayload.from(data, data.length));

    buffer.clear();

    this.position = this.position + this.bytesRead;
    this.fileChannel.read(buffer, this.position, buffer, this);
}

@Override
public void failed(Throwable exc, ByteBuffer attachment) {
    log.error(exc);
}
```

REACTIVE CORE

Reactor is a **fully non-blocking** foundation with efficient demand management. It directly interacts with Java 8 *functional API*, *Completable Future*, *Stream* and *Duration*.

TYPED [0|1|N] SEQUENCES

Reactor offers **2 reactive composable API** Flux [N] and Mono [0|1] extensively implementing Reactive Extensions.

NON BLOCKING IO

Suited for **Microservices** Architecture, Reactor offers **backpressure-ready network engines** for HTTP (including Websockets), TCP and UDP. Reactive Encoding/Decoding is fully supported.

Efficient Message Passing

Reactor Processors, Operators and Timers can **sustain high throughput rates** on the order of 10's of millions of messages per second. Plus its **low memory footprint** should go under most of the radars. Reactor is part of an ongoing research effort to challenge further the possible flow optimizations.

Reactive Streams

reactive-streams.org

```
package org.reactivestreams;

public interface Publisher<T> {

    void subscribe(Subscriber<? super T> s);
}
```

Reactive Streams

reactive-streams.org

```
package org.reactivestreams;

public interface Subscriber<T> {

    public void onSubscribe(Subscription s);

    public void onNext(T t);

    public void onError(Throwable t);

    public void onComplete();
}
```

Reactive Streams

reactive-streams.org

JAVA

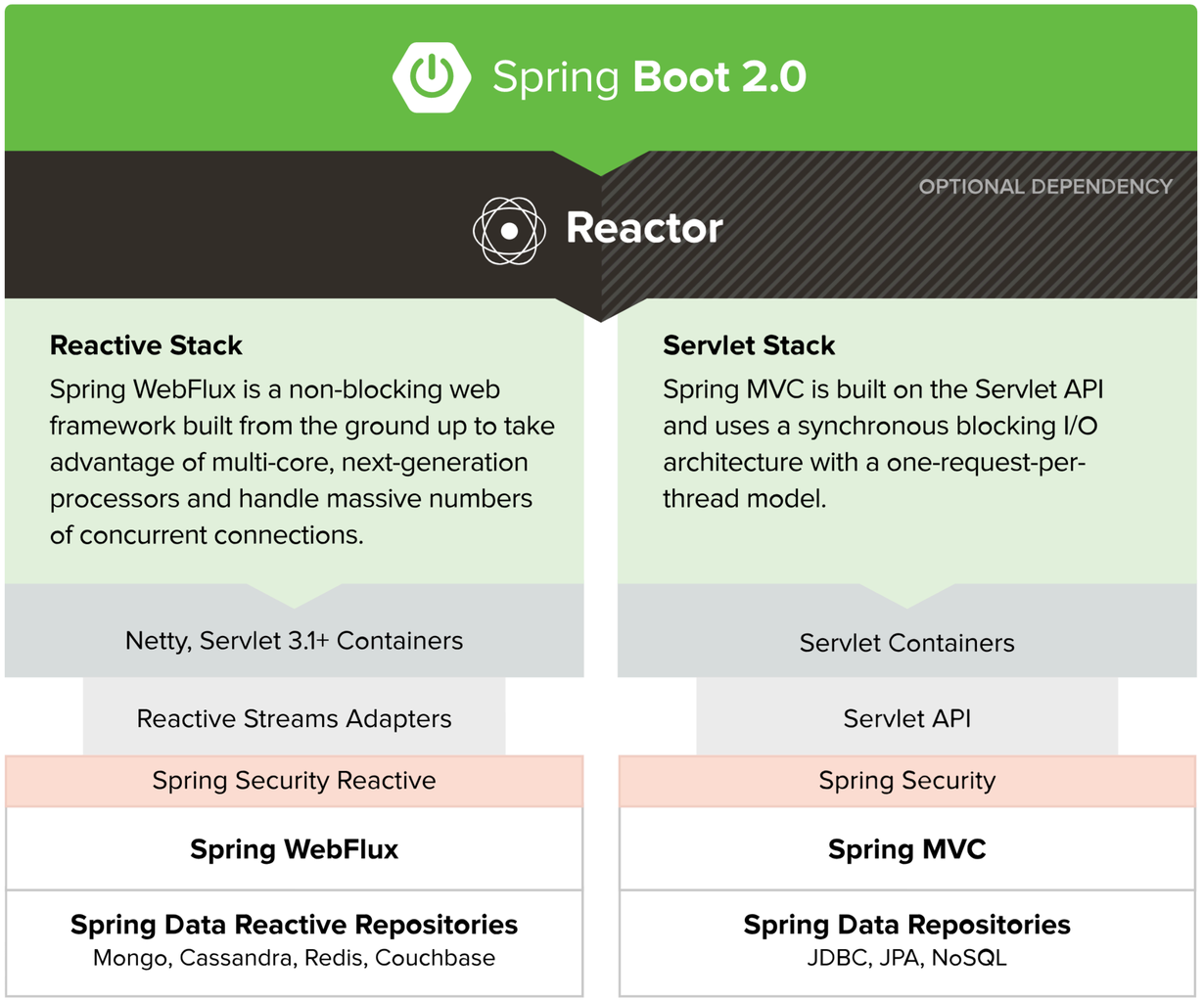
```java
package org.reactivestreams;

public interface Subscription {

    public void request(long n);

    public void cancel();
}
```

Reactive Streams

reactive-streams.org

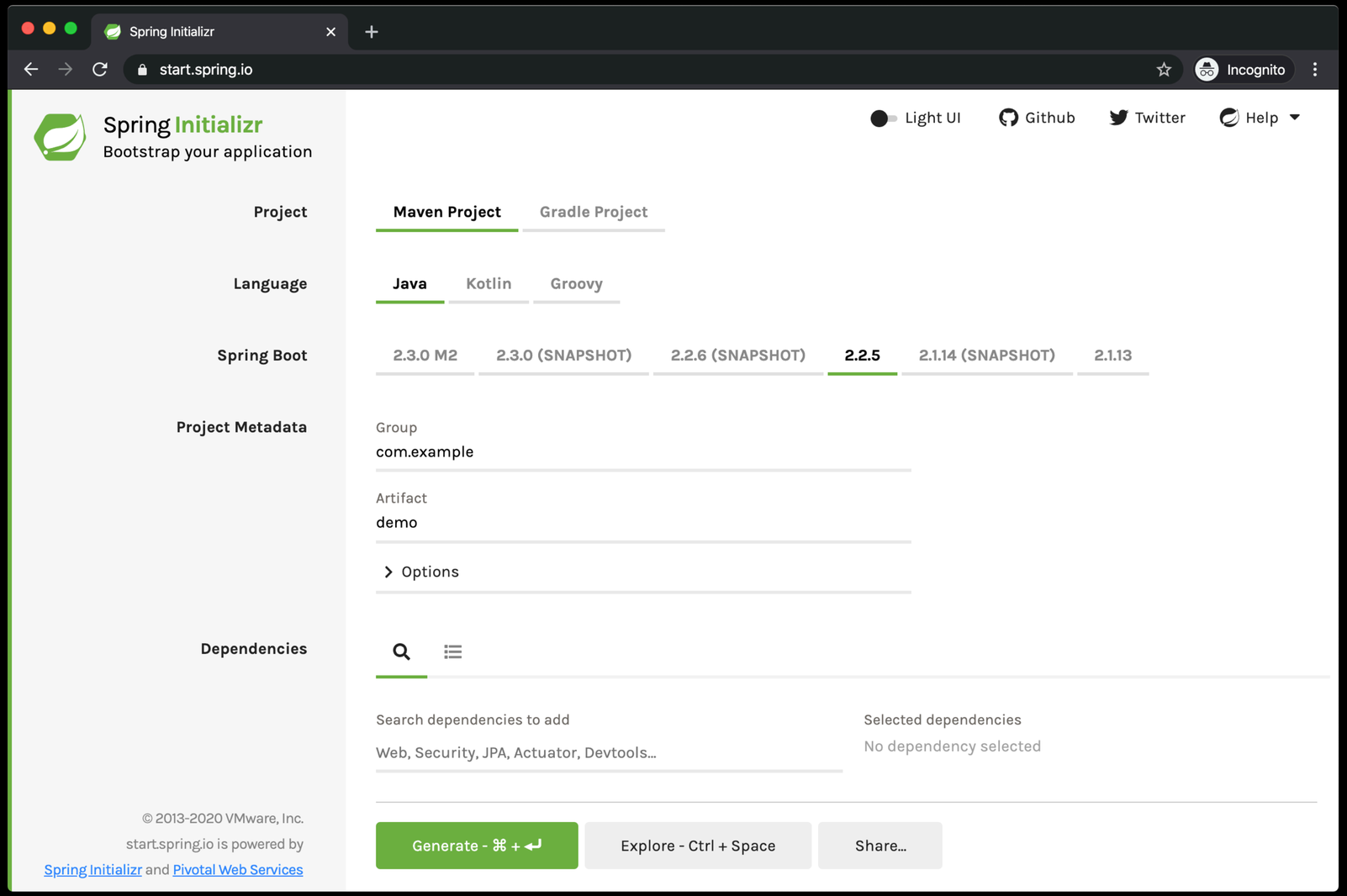
```
package org.reactivestreams;

public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {
}
```

Spring WebFlux

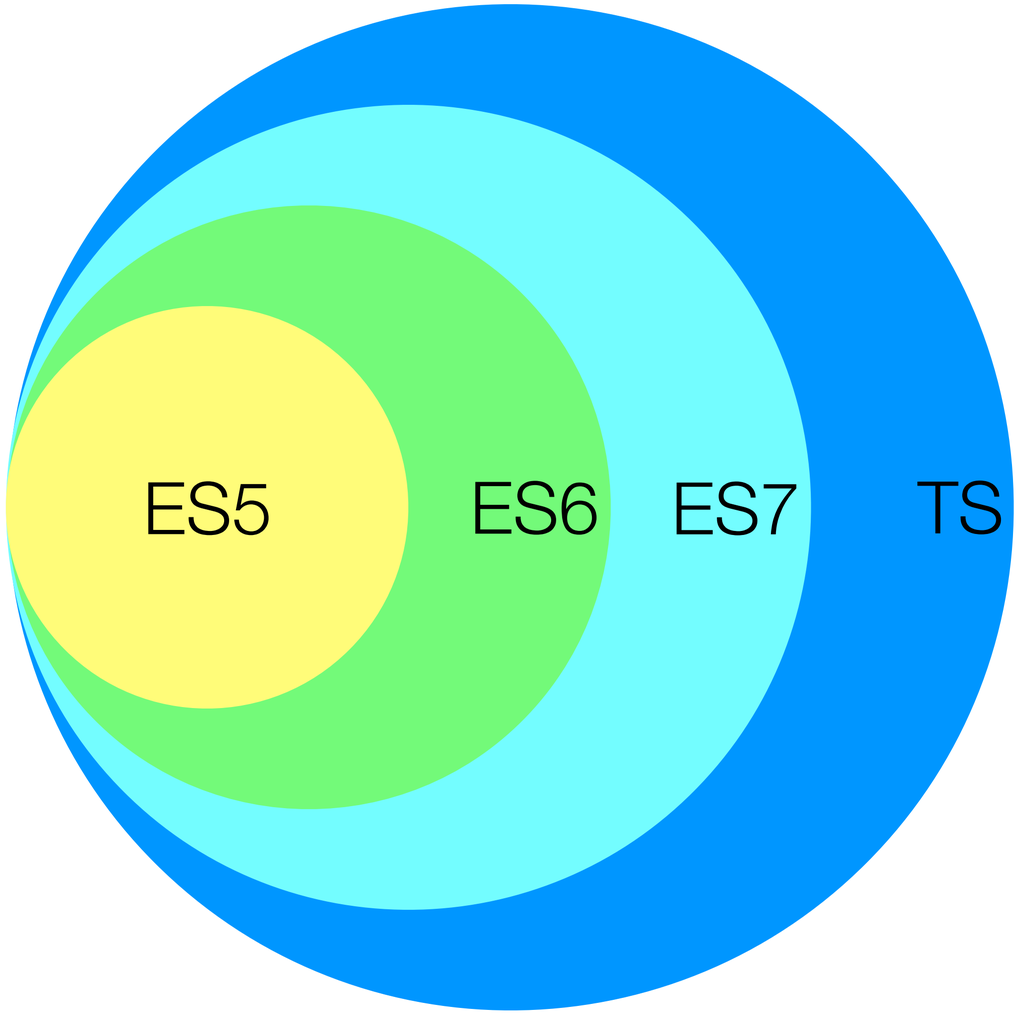


```java
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}

@Entity
class Blog {

    @Id
    @GeneratedValue
    private Long id;
    private String name;

    // getters, setters, toString(), etc
}

@RepositoryRestResource
interface BlogRepository extends JpaRepository<Blog, Long> {
}
```

Demo: Build a Spring WebFlux API



ES6, ES7 and TypeScript



ES5: es5.github.io

ES6: git.io/es6features

ES7: bit.ly/es7features

TS: www.typescriptlang.org

TypeScript

```ts
export class Bus {
  private _name: string;

  constructor(name?: string) {
    this._name = name;
  }

  get name(): string {
    return this._name;
  }

  set name(value: string) {
    this._name = value;
  }

  toString() : string {
    return `This bus's name is ${this._name}`;
  }
}
```

“Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.”

https://nodejs.org

https://github.com/creationix/nvm

Hello World with React

```
<div id="root"></div>

<script>
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
</script>
```

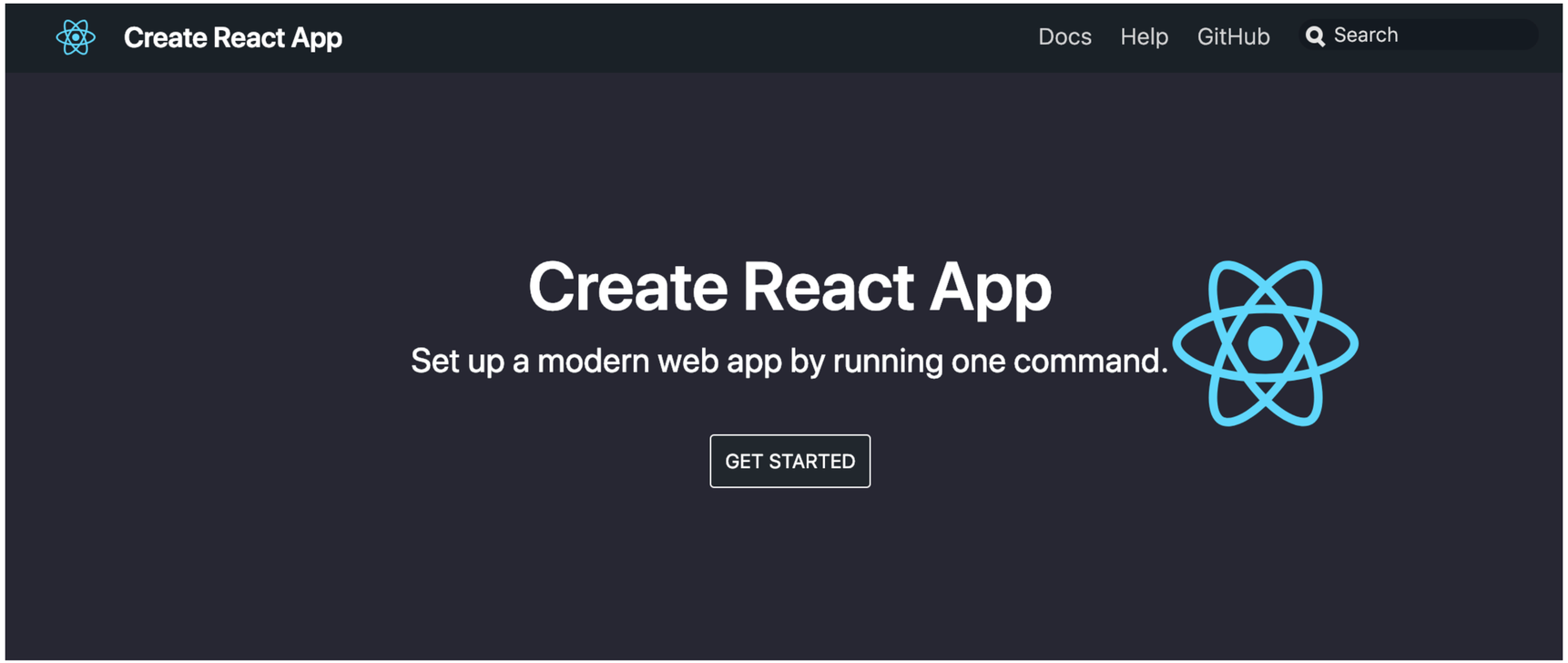
http://codepen.io/gaearon/pen/ZpvBNJ?editors=0100

Imperative Code

```
if (count > 99) {
  if (!hasFire()) {
    addFire();
  }
} else {
  if (hasFire()) {
    removeFire();
  }
}
if (count === 0) {
  if (hasBadge()) {
    removeBadge();
  }
  return;
}
if (!hasBadge()) {
  addBadge();
}
var countText = count > 99 ? "99+" : count.toString();
getBadge().setText(countText);
```

Declarative Code

```
if (count === 0) {
  return <div className="bell"/>;
} else if (count <= 99) {
  return (
    <div className="bell">
      <span className="badge">{count}</span>
    </div>
  );
} else {
  return (
    <div className="bell onFire">
      <span className="badge">99+</span>
    </div>
  );
}
```



Less to Learn

You don't need to learn and configure many build tools. Instant reloads help you focus on development. When it's time to deploy, your bundles are optimized automatically.

Only One Dependency

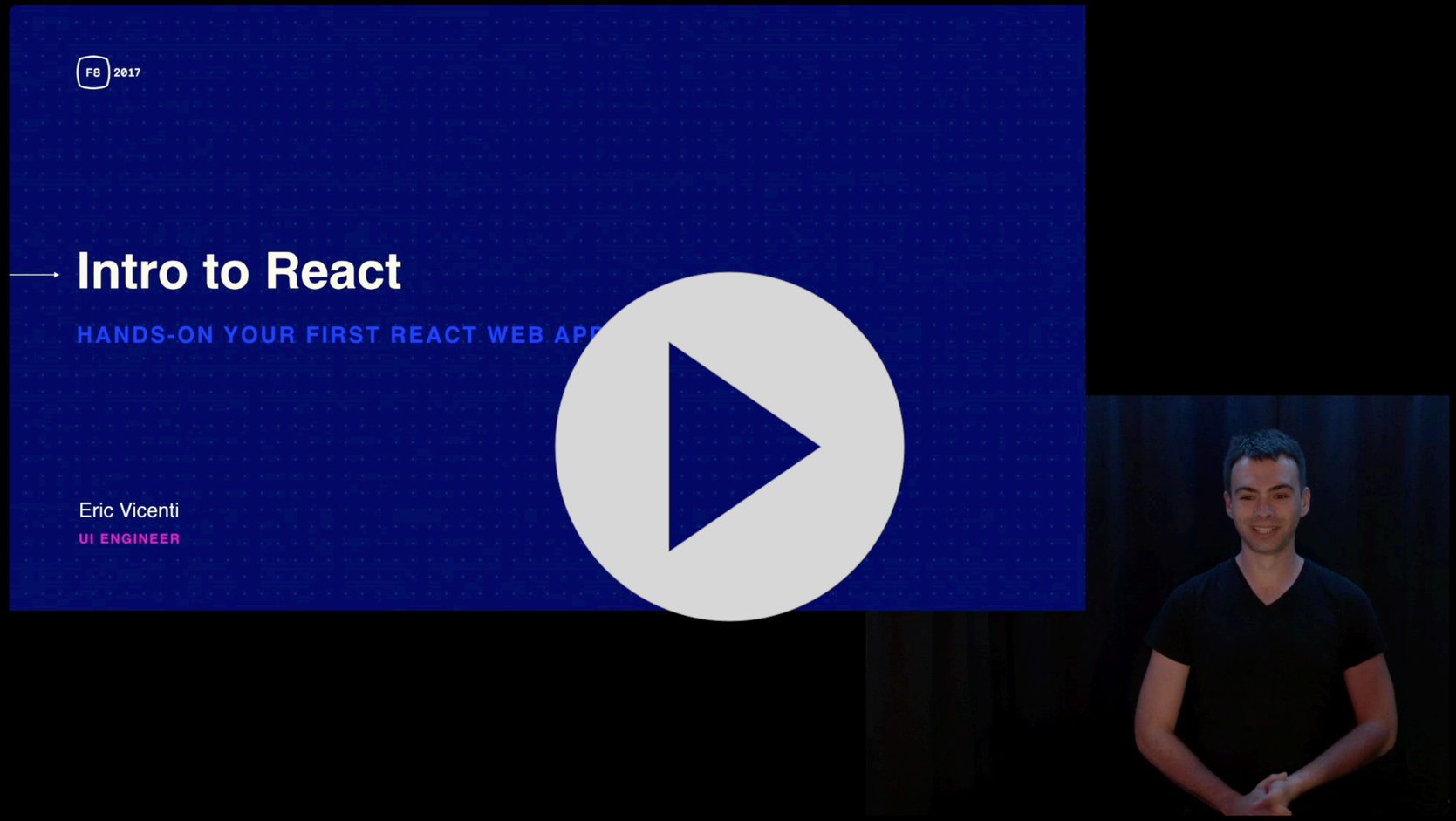
Your app only needs one build dependency. We test Create React App to make sure that all of its underlying pieces work together seamlessly – no complicated version mismatches.

No Lock-In

Under the hood, we use Webpack, Babel, ESLint, and other amazing projects to power your app. If you ever want an advanced configuration, you can ”eject” from Create React App and edit their config files directly.

https://facebook.github.io/create-react-app/

Learning React



https://vimeo.com/213710634

Handling Streaming Data in React

Polling with Interval

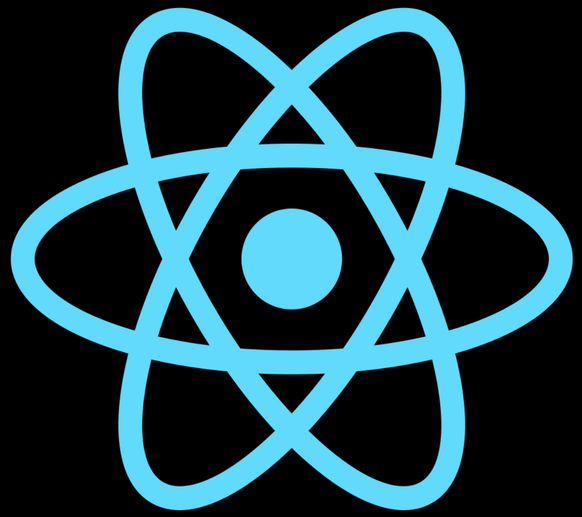
Polling with RxJS

WebSocket

Server-Sent Events and EventSource

RSocket

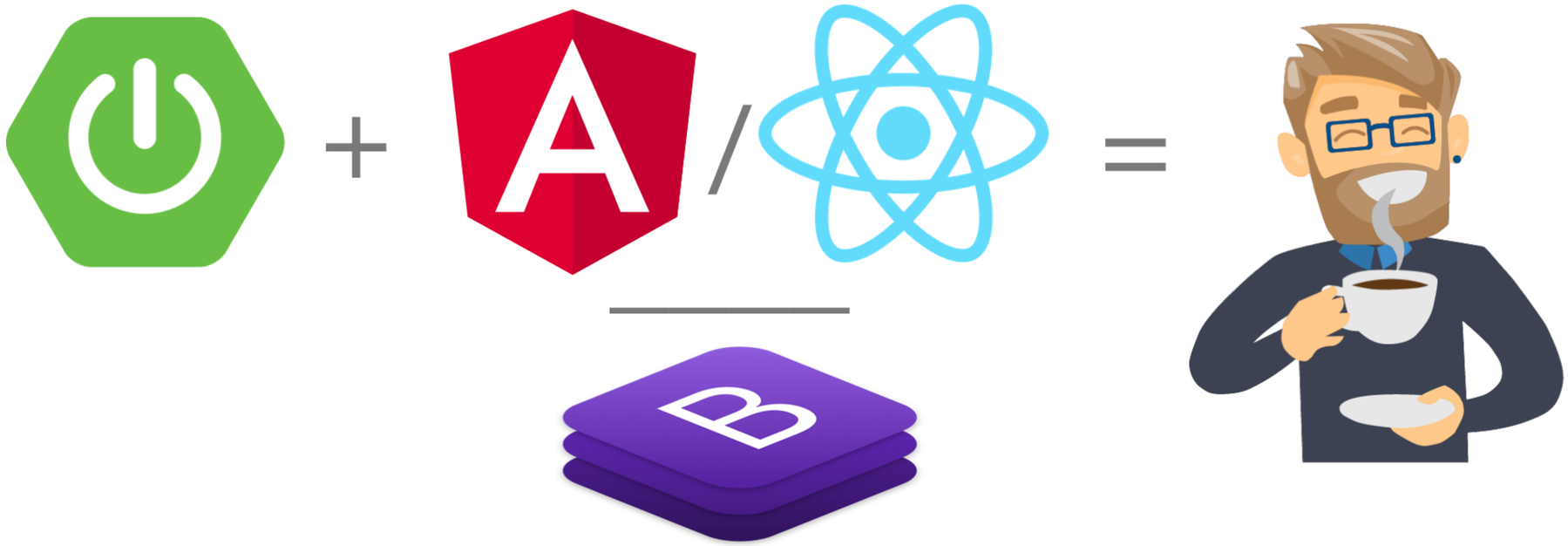
Demo: Build a React Client



```
class ProfileList extends React.Component<ProfileListProps, ProfileListState> {

  constructor(props: ProfileListProps) {
    super(props);

    this.state = {
      profiles: [],
      isLoading: false
    };
  }

  async componentDidMount() {
    this.setState({isLoading: true});

    const response = await fetch('http://localhost:8080/profiles', {
      headers: {
        Authorization: 'Bearer ' + this.props.authState.accessToken
      }
    });
    const data = await response.json();
    this.setState({profiles: data, isLoading: false});
  }

  render() {
    const {profiles, isLoading} = this.state;
    ...
  }
}
```

JHipster

jhipster.tech



JHipster is a development platform to generate, develop and deploy
Spring Boot + Angular/React Web applications and Spring microservices.

The JHipster Mini-Book

v5.0 Available Now!

jhipster-book.com

21-points.com

@jhipster_book

Write your own InfoQ mini-book! github.com/mraible/infoq-mini-book

Action!

Try Spring WebFlux

Try React

Try OIDC

Explore PWAs

Enjoy the experience!

DIY: Full Stack Reactive

Full Stack Reactive with Spring WebFlux, WebSockets, and React

Matt Raible

Spring WebFlux can be used to create a REST API with streaming data. Spring WebFlux can also be integrated with WebSockets to provide notifications that clients can listen to. Combining the two is a powerful way to provide real-time data streaming to JavaScript or mobile clients. Add React to the mix and you have an excellent foundation for a full-stack reactive architecture.

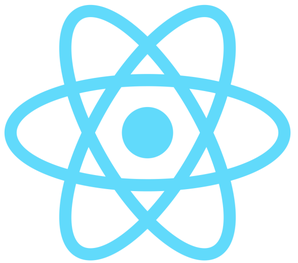
React is a UI toolkit (similar to GWT) that lets you build components with JavaScript (or TypeScript) and JSX. JSX is how you define elements in React and it looks very similar to XML. React's API and JSX are the core of the framework; everything else is an add-on. I won't go into nitty-gritty details about React, we'll assume you've heard of it and are eager to learn how to make it even *more* reactive!

I teamed up with Josh Long to write this post. Josh is a fellow Java Champion, Spring Developer Advocate, and all around excellent guy at Pivotal. Josh and I've been good friends for a while now, sharing the same passion for Java, developers, and building bleeding-edge applications. We hope you like this series!

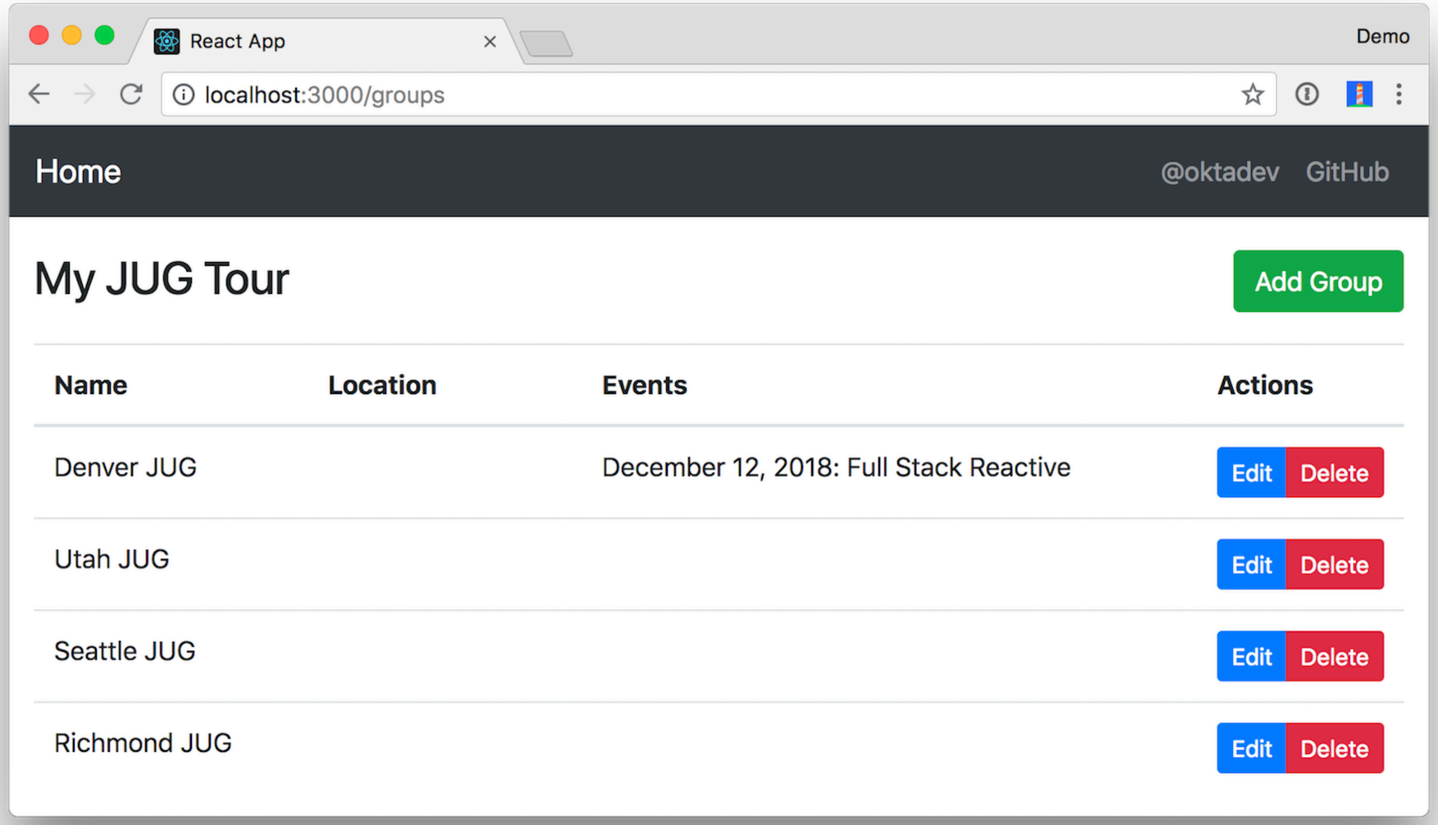
Today, we'll show you how to build a *full-stack* application using Spring WebFlux for the API, WebSockets for notifications, and React for the UI. This article is the third in a three-part series about reactive programming and Spring WebFlux. The first two are listed below.

1. Get Started with Reactive Programming in Spring
2. Build Reactive APIs with Spring WebFlux

http://bit.ly/webflux-and-react



CRUD with React and Spring Boot



http://bit.ly/react-boot-crud

#LearnAllTheThings

Matt Raible September 25, 2018

Full Stack Reactive with Spring WebFlux, WebSockets, and React

Spring WebFlux can be used to create a REST API with streaming data. Spring WebFlux can also be integrated with WebSockets to provide notifications that clients can listen to. Combining the two is a powerful way to provide real-time data streaming to JavaScript or mobile clients. Add React to the mix and you have an excellent foundation for a full-stack reactive architecture. React is a UI toolkit (similar to GWT) that lets you build components...

Read more

Matt Raible September 24, 2018

Build Reactive APIs with Spring WebFlux

Spring Boot 2.0 was a long-awaited release from the good folks at Pivotal. One of its new features is reactive web programming support with Spring WebFlux. Spring WebFlux is a web framework that's built on top of Project Reactor, to give you asynchronous I/O, and allow your application to perform better. If you're familiar with Spring MVC and building REST APIs, you'll enjoy Spring WebFlux. There's just a few basic concepts that are different. Once...

Read more

Matt Raible September 21, 2018

Get Started with Reactive Programming in Spring


developer.okta.com/blog
@oktadev

Use the Source, Luke!

https://github.com/oktadeveloper/okta-spring-webflux-react-example

Questions?

Keep in touch!

@starbuxman

@mraible

Presentation

speakerdeck.com/mraible

Code

github.com/oktadeveloper

@oktadev