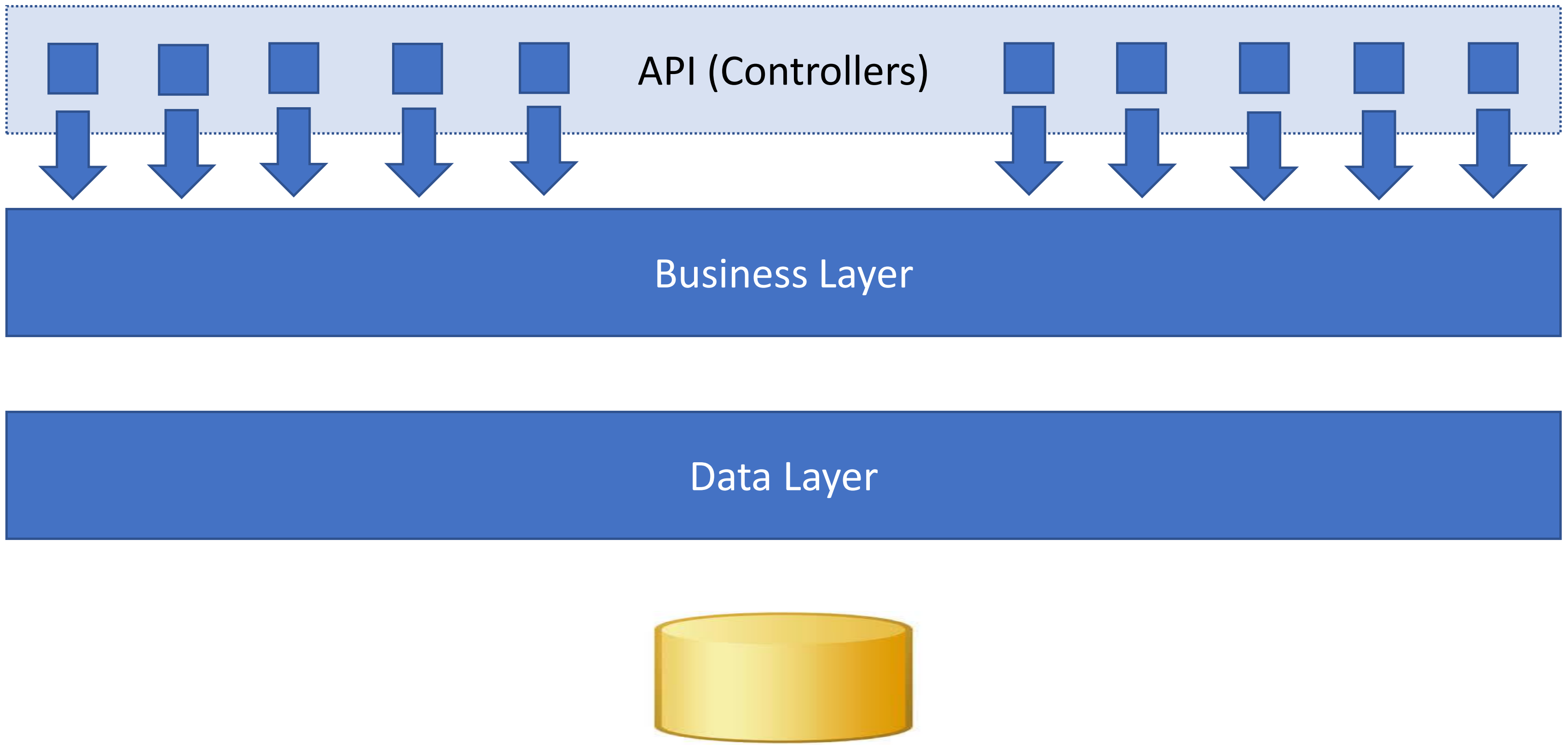
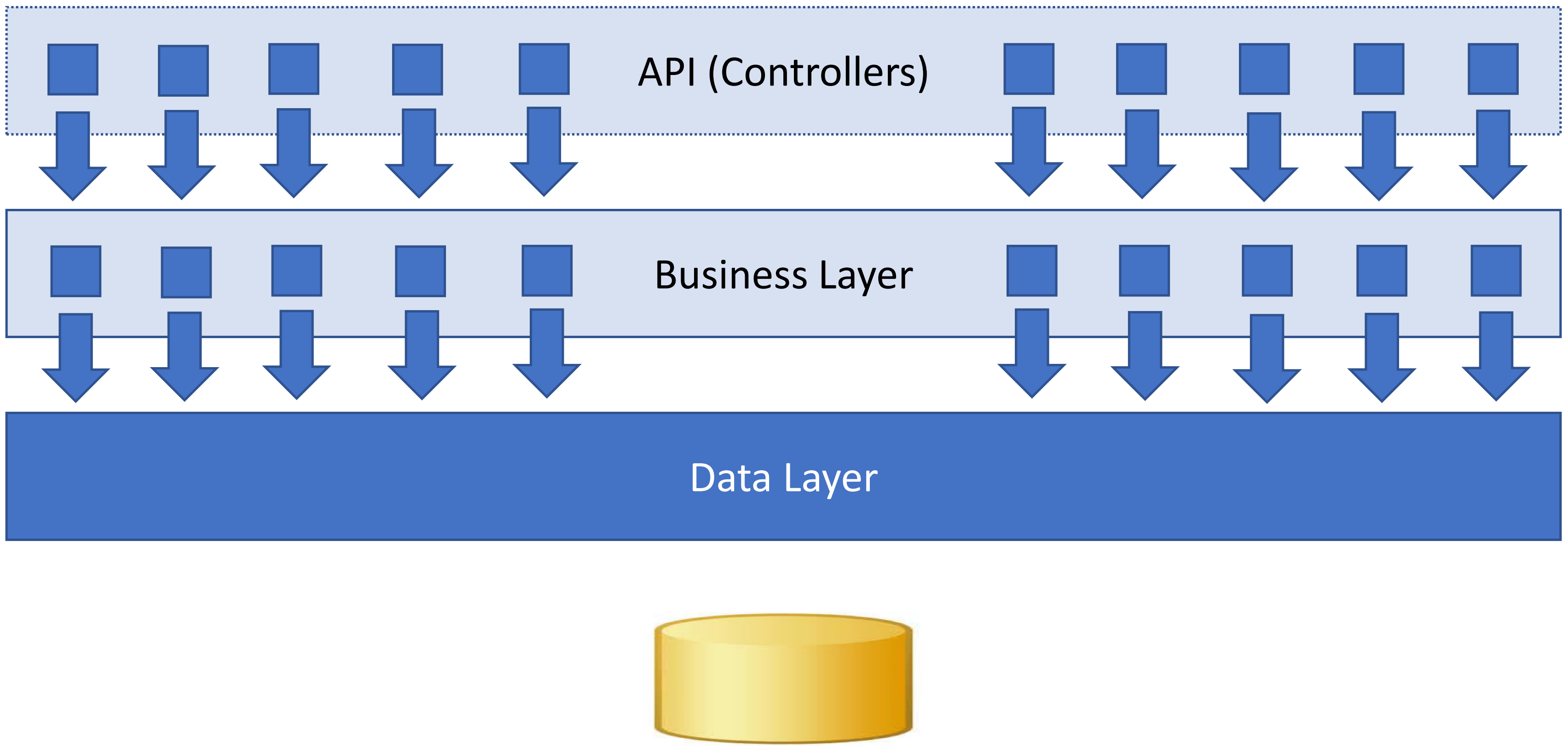
A Lap Around React and Angular 2

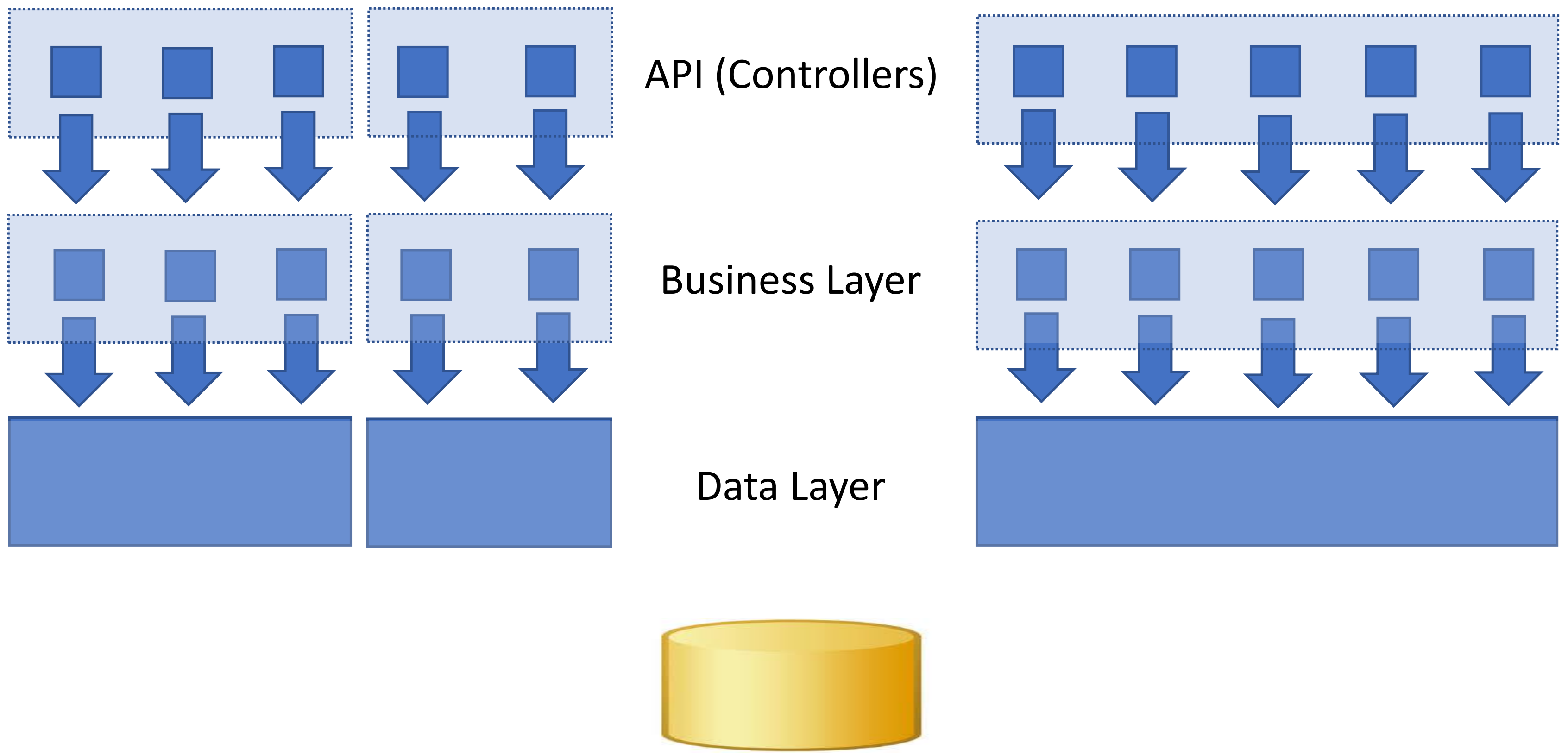
Server Side Architecture



Server Side Architecture – Action Based



Server Side Architecture – Modular

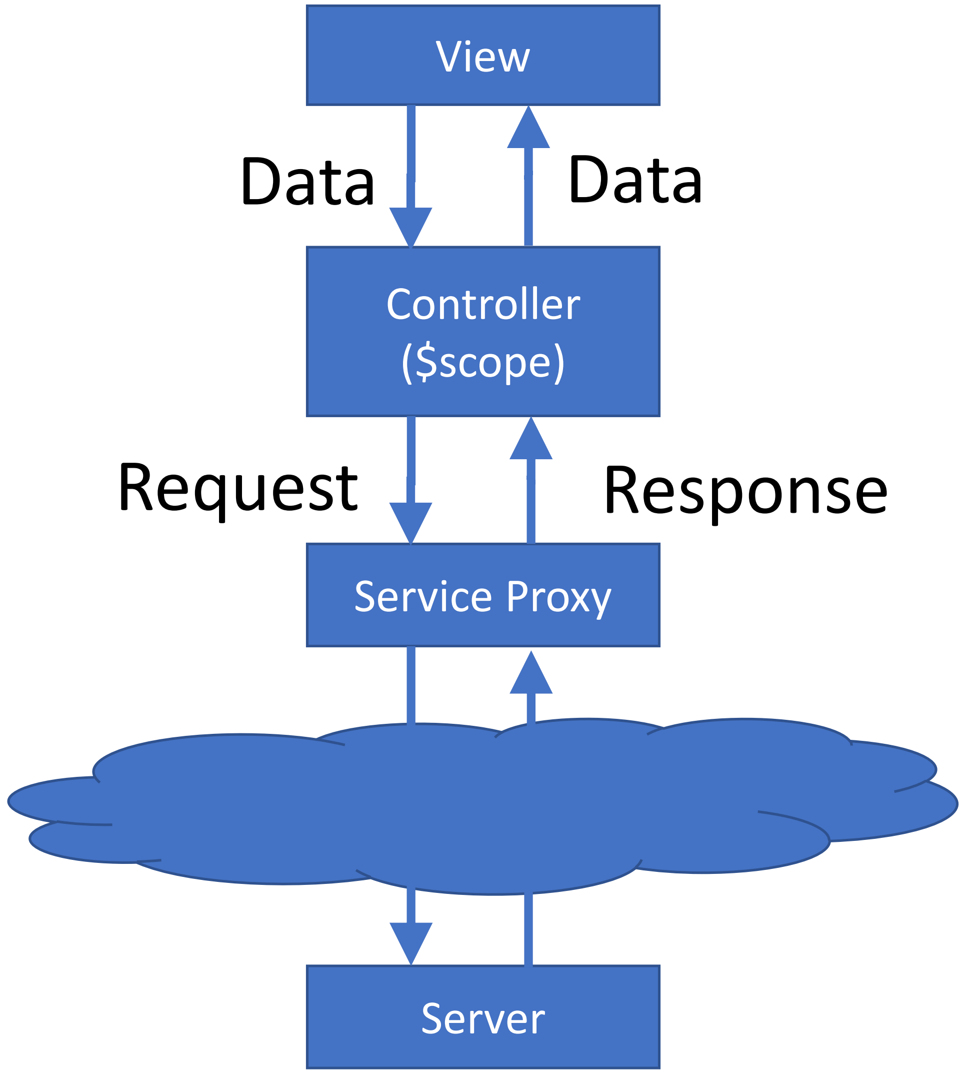


JavaScript Fatigue is a thing

- Over the lifetime of your system, you may want to change JavaScript frameworks/methodologies… multiple times
- You will put yourself behind in the recruiting game if you are using old technology
- You need to be able to change frameworks without having to rewrite the entire application
- You need to keep learning

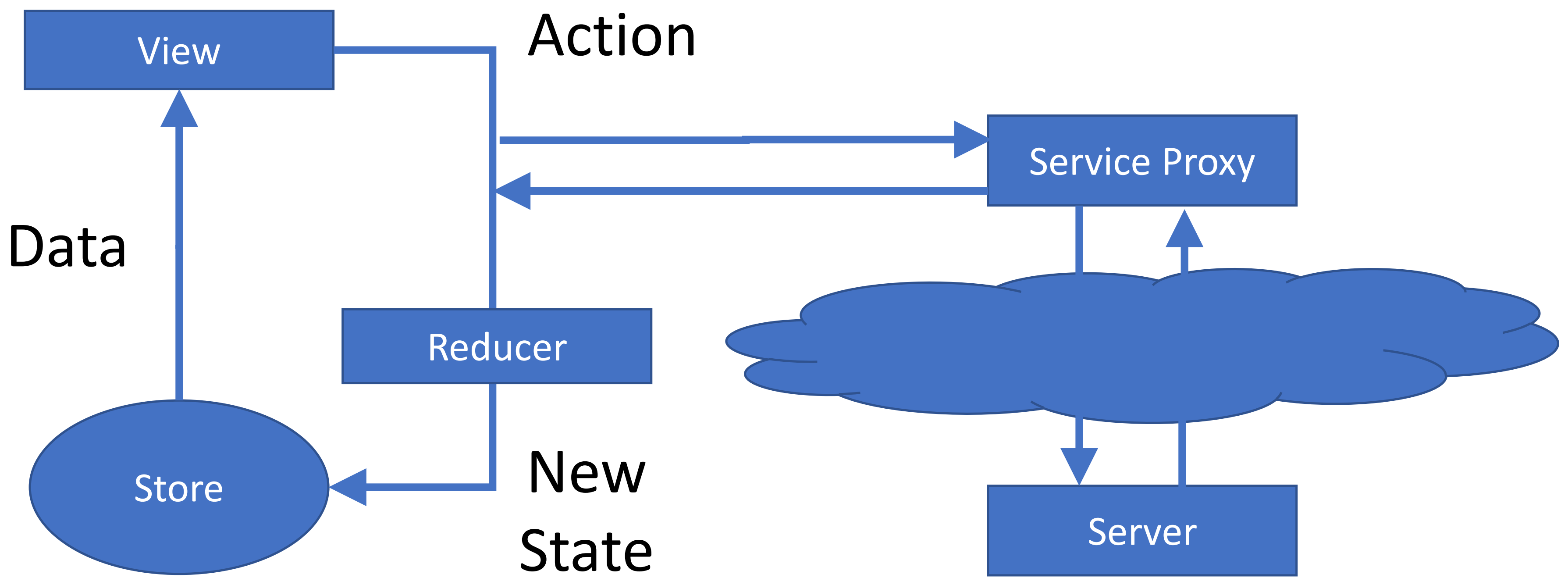
SPAs and ASPs

Angular 1 Data Binding



React/Redux

Redux - Unidirectional Data Binding



JSX – HTML in your JavaScript

```
class MyPage extends React.Component
{
    render() {
        return <div>Hello World!</div>;
    }
}
```

Working with props

```
class MyPage extends React.Component
{
    render() {
        return
            <div>
                {this.props.greeting}, {this.props.user}!
            </div>;
    }
}

const mapStateToProps = (state) => {
    return {
        user: state.currentUser,
        greeting: state.greeting
    };
}
```

Action creators

```
const SAY_THANKS = "SAY_THANKS";

export function sayThanks() {
  return (dispatch) => {
    dispatch({ type: SAY_THANKS });
  };
}
```

Working with actions

```
class MyPage extends React.Component
{
    sayThanks = (e) => {
        this.props.actions.sayThanks();
    }

    render() {
        return
            <div>
                {this.props.greeting},
                {this.props.user}!<br/>
                <input
                     type="button"
                     onclick={this.sayThanks}
                     value="Say Thanks!" />
            </div>;
    }
}
```

```
import myActions from '.\actions';
import {bindActionCreators} from 'redux';

const mapStateToProps = (state) => {
    return {
        user: state.currentUser,
        greeting: state.greeting
    };
}

const mapDispatchToProps = (dispatch) => {
  return {
    actions: bindActionCreators(
        myActions, dispatch)
  };
}
```

Connecting to the store

```
class MyPage extends React.Component
{
    sayThanks = (e) => {
        this.props.actions.sayThanks();
    }

    render() {
        return
            <div>
                {this.props.greeting},
                {this.props.user}!<br/>
                <input
                     type="button"
                     onclick={this.sayThanks}
                     value="Say Thanks!" />
            </div>;
    }
}
```

```
const mapStateToProps = (state) => {
    return {
        user: state.currentUser,
        greeting: state.greeting
    };
}

const mapDispatchToProps = (dispatch) => {
  return {
    actions: bindActionCreators(
        actions, dispatch)
  };
}

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(MyPage);
```

Reducer

```
const INITIAL_STATE = {
  currentUser: "whoever you are",
  greeting: "Hello"
}

export default function MyReducer(state = INITIAL_STATE, action) {
  switch (action.type) {
    case "SAY_THANKS":
    {
      return Object.assign({}, state, {greeting: "You're welcome"});
    }
  }
  return state;
}
```

2 Types of Components

Connected Components

- Receive updates from the store
- Can dispatch actions
- Need mapStateToProps() and mapDispatchToProps()
- Accesses data and actions through props

Presentation Components

- Do not receive updates from the store
- Cannot dispatch actions
- Can only use what is given to them (data and actions)

Presentation Components Can Be Simple!

```
let Greeting = ({greeting, user}) =>
    <span>{greeting}, {user}!</span>;

<div>
    <Greeting
        greeting={this.props.greeting}
        user={this.props.user} />
    <br/>
    <input
         type="button"
         onclick={this.sayThanks}
         value="Say Thanks!" />
</div>;
```

Lifecycle Methods

Mounting - Component is being created and inserted into the DOM:

- componentWillMount()
- render()
- componentDidMount()

Updating - Props or state changed

- componentWillReceiveProps()
- shouldComponentUpdate()
- componentWillUpdate()
- render()
- componentDidUpdate()

Unmounting - Component is being removed from the DOM:

- componentWillUnmount()

Virtual DOM

The Virtual DOM is an in-memory copy of the actual DOM.

When state changes:

1) render() is called
2) The entire Virtual DOM is re-rendered
3) The Virtual DOM is compared to the actual DOM and changes are pushed to the actual DOM

Why this matters:

- Changes to the actual DOM can be slow (redrawing, CSS, layout, etc.)
- Components are only re-rendered when the state changes

Testing

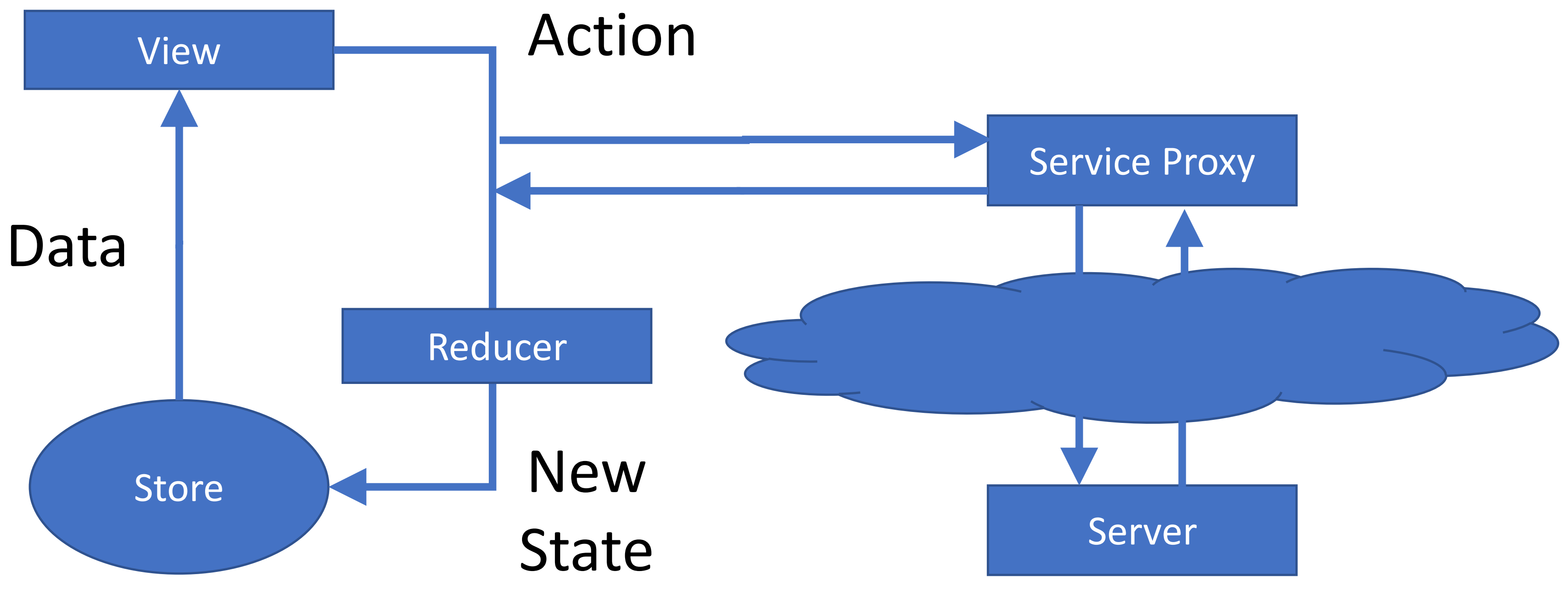
- Mocha, Jasmine, Jest – test runners
- Chai – assertions
- Enzyme – for testing React components

Things you can test:

- Reducers – state changes
- Component rendering – given some state, does it render correctly?

Demo!

Redux - Unidirectional Data Binding



Angular 2

Components

```
@Component({
  selector: 'greeting',
  templateUrl: './greeting.component.html',
  styleUrls: ['./greeting.component.css', '../../app.component.css']
})
export class GreetingComponent {
    greeting: string;
    currentUser: string
}
```

Components - OnInit

```
@Component({
  selector: 'greeting',
  templateUrl: './greeting.component.html',
  styleUrls: ['./greeting.component.css', '../../app.component.css']
})
export class GreetingComponent implements OnInit {
    greeting: string;
    currentUser: string

    ngOnInit() {
        this.greeting = "Hello";
        this.currentUser = "whoever you are";
    }
}
```

Services and Observables

```
@Injectable()
export class GreetingService {

  constructor(private http: Http) {}

  getGreetingData(): Observable<any> {
    return this.http.get('http://localhost:3000/api/greeting')
                    .map((res: Response) => res.json())
                    .catch(this.handleError);
  }

  handleError(error: any) {
    // do something
  }
}
```

Components – Working with Observables

```
export class GreetingComponent implements OnInit, OnDestroy {
    greeting: string;
    currentUser: string;
    private sub: Subscription;

    constructor(private _greetingService: GreetingService) { }

    ngOnInit() {
        this.sub = this._greetingService.getGreetingData()
            .subscribe(
            data => { this.greeting = data.greeting; this.currentUser = data.currentUser; },
            error => { /* handle errors */ }
        );
    }

    ngOnDestroy() {
        this.sub.unsubscribe();
    }
}
```

HTML Templates – Data Binding

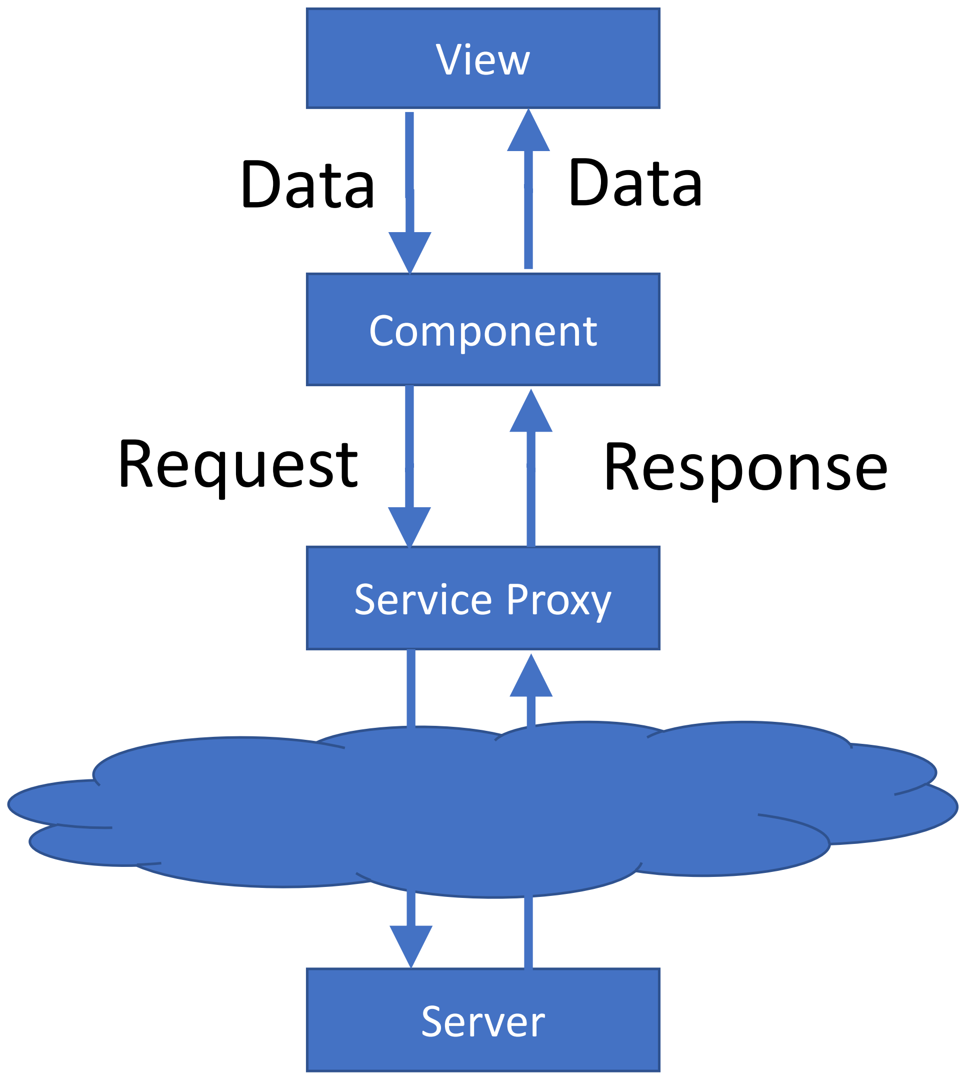
```
<div *ngIf="currentUser">
    {{greeting}}, {{currentUser}}!
</div>
<input type="text" [(ngModel)]="currentUser" />
<input
    type="button"
    [disabled]="isValid()"
    (click)="sayThanks()"
    value="Say Thanks!" />
```

Modules

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    AppRoutingModule,
    SessionModule
  ],
  exports: [
    LookupDataService
  ]
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- Declarations – components and directives that we will use within HTML markup
- Imports – other Angular modules that we are going to reference (you don’t need to reference npm packages)
- Exports – things you want to expose to other Angular modules
- Providers – classes that you are going to reference in this module that you want to load using dependency injection
- Bootstrap – the component that will load to start the app

Angular 2 Data Binding



Demo!

Comparison

	React/Redux	Angular 2
Transpiling	Babel (but you can use TypeScript)	TypeScript
Bundling	Webpack	Webpack
Linting	ESLint	TSLint
Hot reloading	Yes	Yes
Project Setup	create-react-app, or another starter project	angular-cli
Component testing	Enzyme	Angular TestBed
Unit testing	Your choice (I used mocha/chai)	Jasmine by default, but your choice
Native	React Native	NativeScript, Ionic, etc.
Size (of my app's dist folder)	4MB	0.8MB
Learning curve	Unidirectional data binding, reducers, actions, JSX	Angular concepts, view syntax
Fun to use	Yes!	Yes!

More Learning – React/Redux

- Getting Started With Redux (video)
  - https://egghead.io/courses/getting-started-with-redux
- Building React Applications With Idiomatic Redux (video)
  - https://egghead.io/courses/building-react-applications-with-idiomatic-redux
- Lots of React/Redux links
  - https://github.com/markerikson/react-redux-links
- Compare React starter projects
  - http://andrewhfarmer.com/starter-project/

More Learning – Angular 2

- Pluralsight – Angular 2: Getting Started
  - https://app.pluralsight.com/library/courses/angular-2-getting-started-update/table-of-contents
- Angular 2 official website
  - http://angular.io
- Angular 2 simple component tutorial
  - http://learnangular2.com/
- LOTS of Angular 2 content on Pluralsight and Egghead.io

Even More Learning

- ES2015 (a.k.a. ES6) tutorial
  - https://babeljs.io/learn-es2015/
- TypeScript official site
  - https://www.typescriptlang.org/
- Just try stuff
  - http://jsfiddle.net

Source Code

- Rails app (including back-end API)
  - http://github.com/jonkruger/openspaces
- Rails app in production (please clean up your test data ☺)
  - http://openspaces.stirtrek.org
- React app
  - http://github.com/jonkruger/openspaces-react
- Angular 2 app
  - http://github.com/jonkruger/openspaces-angular

Thanks! Happy coding!