IPT – Intellectual Products & Technologies Ltd.
Bl. 44- Office 2 Slatinska Str., Sofia, Bulgaria
☏ (+359-2) 973-38-38 🖷 (+359-2) 971-28-50 ✉ office@iproduct.org 🖰 http://www.iproduct.org/

Advanced React JS + Redux Development

Course code:	**IJ - 27**
Course domain:	**Software Engineering**
Number of modules:	**1**
Duration of the course:	**40 astr. hours / 54 study[1] hours**

Sofia, 2016

1 Duration of a study hour is 45 minutes.

IPT – Intellectual Products & Technologies Ltd.
Bl. 44- Office 2 Slatinska Str., Sofia, Bulgaria
☏ (+359-2) 973-38-38 🖷 (+359-2) 971-28-50 ✉ office@iproduct.org 🖰 http://www.iproduct.org/

Advanced React JS + Redux Development

STUDY PLAN

Module name	1. Advanced React JS + Redux Development
Lectures, astr. hours	16
Laboratory exercises, astr. hours	24
Total, astr. hours	40

Lecturer: **Trayan Iliev**
IPT – Intellectual Products & Technologies Ltd.
E-mail: tiliev@iproduct.org

Target audience: Experienced JavaScript developers with practical experience using HTML5, CSS3, Bootstrap, jQuery, object-oriented JS, and JS design patterns.

Course duration: Duration of the course is 40 astronomical (54 study) hours total. Training will be conducted in 10 sessions by 4 astr. hours.

Expected results:

- Experience using *TypeScript* and *ECMAScript 6 (ECMAScript 2015, Harmony)* new features, *Classes* and *Promises*.
- Implementing *reusable presentation components* using *React JS* framework.
- In depth understanding and practical experience building *single page web applications (SPA)* using *React JS , React Router, and HTML 5 History API.*
- Deep understanding and planning of *application state representation and management* opportunities and best practices for maintainability and performance optimization, using immutability helper libraries.
- Ability to create extensible and maintainable large-scale SPAs using *Redux* predictable state container.
- Asynchronous state transitions and data-flows with *redux-thunk, redux-promise, middleware, redux-observable*.
- Test-driven development of *React JS* and *Redux* components and apps using *Jest, redux-mock-store, React Test Utils, Enzyme*.

Course Description:

The course provides practical experience using *React JS* and *Redux* web component and application development frameworks. The main topics that will be covered during the course are:

1. **Novelties in ECMAScript 6 (ECMAScript 2015, Harmony)** – class and constructor syntax, static methods, inheritance, let and var, function lambdas (=>), template literals, destructuring assignment, default and rest parameters, spread operator, Promises. *ES6* development using *Babel* and *Webpack*. Writing reusable components as *ES6* classes. Defining and loading JavaScript modules – requirements for code modularity, module systems, loaders, and JS design patterns: *CommonJS* and *ES 6* modules. (2 h.)

2. **Introduction to TypeScript** – functions, interfaces, classes and constructors. Common types: *Boolean, Number, String, Array, Enum, Any, Void*. Duck typing. Declaring contracts using *Interfaces* – properties and optional properties, function types, class types, array types, hybrid types, extension of interfaces. *Classes* – constructors, public/private properties, *get* and *set* accessors, *static* and *instance* sides. *Functions* and function types – optional default and rest parameters, function lambdas and use of *this*, overloads. Using *Enums*. Modules in *TypeScript* – *namespaces* and *modules* (former internal and external modules), using *import, export* and *require*. Interoperability with external JS libraries – *ambient type declarations* and ambient modules. Module resolution. *Generic type parameters* – writing generic functions and classes, generic constructors, bounded generics. Type inference and type compatibility. Declaration merging. *Decorators*. Using *mixins* in *TypeScript*. Configuring, building and deploying *TypeScrip*t project – *tsconfig.json, typings* for *npm packages, compiler options*. Integration with build tools – *npm, webpack*. (4 h.)

3. **Introduction to Single Page Applications (SPA) development** – *Model-View-Controller (MVC), Model-View-Presenter (MVC), Model-View-ViewModel (MVVM) – MV** patterns. Setting up a build system, defining front and back-end architecture (roter, models, views), application design and development (views, APIs and models, router config). *React* main features and advantages – simple (just the *View*) and superfast, component oriented development using pure JavaScript (ES 6), virtual DOM, one-way reactive data flow, MVC framework agnostic. (1 h.)

4. **Building React components and applications.** Development workflow, project setup and configuration – *package.json, babel (ES 6), webpack*. Building simple *React* components and application. Top level API: *React, ReacrDOM, ReactDOMServer*. *React.createElement()* and *ReactDOM.render()* methods. Using JSX in React. Showing dynamic values. Composing components using properties. Building sample Comments application using official React.js tutorial. Stateful components – react states, setting initial state, updating state. Events in *React*, managing DOM events. (3 h.)

5. **JSX in depth** – differences with HTML, transformation to JavaScript, namespaced components, expressions, child expressions and comments, props mutation anti-pattern, spread attributes, using HTML entities, custom attributes, *if-else*, immediately-invoked function expressions (IIFE). (2 h.)

6. **Components composition in depth** – ownership, *this.props.children*, *React.Children* utilities, child *reconciliation*, stateful children and dynamic children using keys. Transferring props. Using *mixins* and *ES6 classes*. *Stateless (pure) functions* as components. Where should we put state – common patterns. Integration with other JavaScript libraries. Using markdown - *__html*, *dangerouslySetInnerHTML*. Ajax requests using *jQuery*. (2 h.)

7. **React rendering lifecycle and life cycle callbacks**. Bidirectional owner-child component communication using callbacks passed as properties. (1 h.)

8. **Working with forms** – interactive props, controlled and uncontrolled components. (1 h.)

9. **Novelties in React.js** – *Dependency Injection (DI)* of custom application services using *React Context*. Using *addons*. Two way binding helpers *addon*. *Refs* to components. (2 h.)

10. **React Router** – routes and components, named components, route parameters, index route, links, index links. Nested routs – nested UI components and routing URLs. Active links. *NavLink* wrapper component. Decoupling the UI from the URL by using routes without paths. Using route redirection to preserve changed routes. Controlling navigation and application state using *onEnter* and *onLeave* hooks. Route patterns and optional route params. Simultaneous navigation in multiple viewports using named routing components. Histories – *browserHistory* vs. *hashHistory*. Server-side support needed when using *HTML 5 History API*. Navigating routes programmatically.

Using custom *History* implementations – coding state as route query parameters. Configuring *Router* programmatically using *RouteConfig* objects and setting *RouteLeaveHooks* to validate navigation. (5 h.)

11. **Performance optimization using immutability** – *shouldComponentUpdate()* component method, *PureRenderMixin* and *shallowCompare()* to boost performance. *Immutability Helpers* addon. Cloning ReactElements. (2 h.)

12. **Testing react components** with *ReactTestUtils* addon – events simulation, mocking, shallow rendering. (3 h.)

13. **Animation addon** – *ReactCSSTransitionGroup* and *ReactTransitionGroup*. *Internationalization (i18n)* and *localization (l10n)*.(2 h.)

14. **Redux – predictable state container for JavaScript apps.** Predecessors: *Flux* design pattern – unidirectional data flow, main components (*Action creators, Actions, Dispatcher, Stores, Views* and *Controller-Views*). Building example TODO application using Flux. *Redux* – combining *Flux* reactive uni-directional data flows with event sourcing principles and state reduction. Three main principles of *Redux* – *single source of truth* (single store); *state is read-only* (only actions can change state); *changes are made using pure functions* – reducers: *(state, action) => state*. *Redux* main components – *Actions, Action Creators, Reducers, Stores*. Interaction between *Redux* components – designing state representation, handling *Actions* using separate *Reducers*, combing reduces into single *Root Reducer* using *combineReducers()* helper function, registering listeners via *Store.subscribe()*, dispatching *Actions* using *Store.dispatch()*, unidirectional data flow. Connecting *Redux* with *React* using *React Redux* bindings. Presentational and container components. Implementing TODO application using *React* and *Redux*. (5 h.)

15. **Advanced Redux** – nested *API* responses normalization using *normalizr.* Implementing *async actions* using *redux-thunk, redux-promise/ redux-promise-middleware,* or *redux-observable*. Asynchronous data-flows using *applyMiddleware()*, examples. Using *redux-devtools*. Building *SPA* with *Redux* and *React Router.* Extended *Redux routing* support using *Redux Router* and *React Router Redux* libraries. *Server rendering* with *Redux*. Computing derived data and *memoizzation* using *Reselect*. Implementing *Undo History*. Testing *Redux* components using *Jest, redux-mock-store, React Test Utils, Enzyme*. Common mistakes with immutable data structures – using immutable update utilities (like *Immutable.js*). (5 h.)

The workshop contains lecture materials and lab exercises. Lectures and exercises will be conducted in parallel and will not be divided in separate sessions in order to achieve immediate reinforcement of theoretical discussions with practical examples and exercises.

During the workshop participants will get practical experience using *ReactJS* and *Redux JS/TS frameworks* as well as developer tools like *create-react-app* and *redux-devtools* to develop and test (single page) web applications. The learning is conducted in small groups – up to 10 participants using problem-based methodology. During the workshop there will be opportunity for discussion of additional questions the participants are interested in.