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/

Full-stack Development with Node.js and React.js

Course code:	**IJ - 22**
Course domain:	**Software Engineering**
Number of modules:	**1**
Duration of the course:	**54 study[1] hours / 40 astr. hours**

Sofia, 2016

1 Duration of a study hour is 45 minutes.

Full-stack Development with Node.js and React.js

STUDY PLAN

Module name	1. Full-stack Development with Node.js and React.js
Lectures, astr. hours	20
Laboratory exercises, astr. hours	20
Total, astr. hours	40

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

Target audience: Medium level JavaScript developers / students with practical experience in building web applications using HTML 5, CSS 3, JavaScript, jQuery, Bootstrap, and basic understanding of SQL or NoSQL databases, HTTP, REST, XML and JSON.

Course duration: Duration of the course is 54 study (40 astr.) hours total. Training will be conducted in multiple sessions 2 - 4 astr. hours each day.

Expected results:

- In depth understanding and practical experience building multi-tier web applications – front-end web UI frameworks, back-end service APIs, REpresentational State Transfer (REST), data repositories using SQL/ NoSQL databases.
- Implementing reusable presentation components using React.js framework.
- Practical experience with *Functional Asynchronous Programming* – using promises and observables, immutability.
- Details of developing *JSON/ REST* service APIs using *Node.js* and *hapi.js*.
- Ability to develop own *full stack JS project* using above technologies, and present the results.

Course Description:

The course provides in-depth study of state-of-the-art JavaScript multi-tier application architectures and development frameworks. *React.js* is chosen as front-end development framework for rapid development of modern, responsive single-page applications that are easy to extend and maintain in long run. The back-end technologies include *Node.js* and *Hapi* framework, The main topics that will be covered during the course include:

1. **Object-oriented JavaScript** – primitive types and objects, accessing objects by reference, properties, functions and methods, using *this* keyword, *call, apply*, and *bind* function methods, prototypal inheritance, polymorphism and method overriding, classes and constructors, classical inheritance and using *instanceof,* using *this, EcmaScript 6 (Harmony, ES 2015) class* and *constructor* syntax, *let* and *var*, function lambdas (=>), Promises. Brief overview of common JS design patterns: *Prototype, Constructor, Module, Singleton, Observer, Factory, Mixin, Decorator.* JavaScript modules – requirements for code modularity, module systems, loaders, and JS design patterns: *CommonJS* and *ES 6 (Harmony).* (4 astr. h.)

2. **Event-driven and asynchronous programming with JavaScript and Node.js.** Non-blocking IO. Event loop. Using callbacks and promises. *Node.js* and *npm* – installation, packages, command line arguments. Running scripts. Using *Node.js shell* (*REPL*) – main features, modes and commands, defining custom commands, redirection. Using *Sublime* editor. Modules and module usage patterns, require, core modules. Global objects in *Node.js*. Developing hello-world web server using *HTTP module*. Routing requests. HTTP methods. Developing HTTP clients using *http.get()* and *http.request()*. Creating custom modules. Serving files (*File System module*) – asynchronous vs. synchronous, completion callbacks, using promises. Buffers. Main file and directory operations, getting read/write streams. Streams, events and pipes. *EventEmitter,* custom events. (6 astr. h.)

3. **Node.js server-side frameworks: introduction to *hapi.js***. N-tier architectores. *JSON APIs*. *RESTful services* and *Service Oriented Architecture (SOA)*. Node.js frameworks. Modular, plugin-based, and configuration-centric architecture of hapi. Main components: servers, connections, routes, handlers and plugins. Installing *hapi*. Creating hello-hapi HTTP server and sample plugin. Paths and rooting – HTTP methods, path parameters, handler methods, configuration. Using plugins – *inert* to serve static content, *good* and *good-console* for logging, *vision* plugin for template

rendering, etc. Writing and registering custom *hapi* plugins. Validation using *joi*. *SQL* and *NoSQL* database integrations with *SQLite* and *MongoDB*. (6 astr. h.)

4. **React.js framework for building composable user interfaces using JavaScript and JSX.** Single Page Applications (SPA) development – setting up a build system, defining front and back-end architecture (router, models, views), application design and development (views, APIs and models, router config). React main features and advantages – simple (just the View) and super-fast, component oriented development using pure JavaScript (ES 6), virtual DOM, one-way reactive data flow, MVC framework agnostic. React development work-flow and project setup and configuration – *package.json, Babel (ES 6), Webpack*. Building simple React components and TODO 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. (4 astr. h.)

5. **React.js in depth.** Events in *React*, managing DOM events. *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. Components composition in depth – ownership, *this.props.children*, *React.Children* utilities, child reconciliation, stateful children and dynamic children using *keys*. Transferring props. Using *mixin*s 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*. React rendering lifecycle and *life cycle callbacks*. Bidirectional owner-child component communication using callbacks passed as properties. Working with forms – interactive props, controlled and uncontrolled components. Using addons. *Two way binding helpers* addon. *Refs* to components. Novelties in *React.js*: *Dependency Injection (DI)* of custom *application services* using *React Context*. (6 astr. h.)

6. **Building Single Page Applications (SPA) using *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. (4 astr. h.)

7. **React.js advanced**. Using immutability, *shouldComponentUpdate()* component method, *PureRenderMixin* and *shallowCompare()* to boost performance. *Immutability Helpers* addon. Cloning *ReactElements*. Animation addon – *ReactCSSTransitionGroup* and *ReactTransitionGroup*. Testing react components with *ReactTestUtils* addon – events simulation, mocking, shallow rendering. (4 astr. h.)
8. **Wrapping up** – developing *React.js SPA* end-to-end project (front-end + JSON /REST server API + database). Internationalization (i18n) and localization (l10n). (4 astr. h.)
9. **Final Test + Course Projects** demonstration and discussion. (2 astr. h.)

The course contains 50% lecture materials and 50% 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 course participants will get practical experience using *Node.js, hapi.js* and *React.js* frameworks for building *full-stack JavaScript applications* by solving problems and exercises. The learning is conducted in small groups (up to 10 participants) using problem-based methodology.

At the end of the course participants are expected to develop their own small to medium sized full-stack application projects using *Node.js, React.js,* and *SQL* or *NoSQL* database of choice (e.g. *SQLite* or *MongoDB*). Each participant will be provided opportunity to demonstrate his/her project and receive feedback and ideas from the instructor and other students.

During the course there will be opportunity for discussion of additional questions the participants are interested in.