**Утвърдил: ……..**
**/ ..... /**
Утвърден от Факултетен съвет
с протокол № ..... / .....

# СОФИЙСКИ УНИВЕРСИТЕТ “СВ. КЛИМЕНТ ОХРИДСКИ”

## Факултет по Математика и Информатика

Специалност: *Информатика, Информационни системи, Компютърни науки, Математика и информатика, Софтуерно инженерство*

☐☐☐☐☐☐☐

Курс: 2+, магистри РСМТ
Учебна година: 2017/2018
Семестър: IV

# УЧЕБНА ПРОГРАМА

**Дисциплина: Разработка на клиент-сървър (fullstack) приложения с Node.js + Express.js + React.js /**
**Fullstack Application Development with Node.js + Express.js + React.js**

☐☐☐☐ Избираема дисциплина

**Преподавател: ас. Траян Илиев**

Асистент: ас. Траян Илиев

| Учебна заетост | Форма | Хорариум |
|---|---|---|
| Аудиторна заетост | Лекции | 30 |
| | Семинарни упражнения | |
| | Практически упражнения (хоспетиране) | 30 |
| **Обща аудиторна заетост** | | **60** |
| Извънаудиторна заетост | Курсов учебен проект | 50 |
| | Доклад/Презентация | 10 |
| | Подготовка на домашни работи | 10 |
| | Подготовка за финален тестови изпит | 10 |
| | Самостоятелна работа с ресурси | 10 |
| **Обща извънаудиторна заетост** | | **90** |
| **ОБЩА ЗАЕТОСТ** | | **150** |
| **Кредити аудиторна заетост** | | **2** |
| **Кредити извънаудиторна заетост** | | **3** |
| **ОБЩО ЕКСТ** | | **5** |

| № | Формиране на оценката по дисциплината | % от оценката |
|---|---|---|
| 1. | Финален курсов проект | 40% |
| 2. | Финален тестови изпит | 40% |
| 3. | Домашни работи през семестъра (текущ контрол) | 10% |
| 4. | Участие в час (задачи по време на упражнения) | 10% |

**Анотация на учебната дисциплина:**

Целта на курса е да запознае участниците с актуалните тенденции и технологии за разработка на **клиент-сървър (fullstack) приложения с Node.js + Express.js + React.js + Redux + MongoDB + Webpack**, известни още като **MERN Stack**.

Курсът започва с разглеждане на технологиите за разработка откъм уеб клиента - *React.js* и *Redux*, които позволяват създаване на разширяеми и лесни за поддръжка компонентно-ориентирани *уеб приложения без презареждане на страницата (Single Page Applications – SPA)* с използване на езика *JavaScript/ ECMAScript 6*.

*Facebook React.js* се наложи като една от най-разпространените в практиката библиотеки за разработка на съвременни *SPA* уеб приложения, която съчетава простота и лекота на употребата с ефективност на реализацията. *JSX* представлява *XML*-подобно разширение на синтаксиса на *JavaScript/ECMAScript*, позволяващо удобна интеграция на темплейти на уеб компоненти и *JavaScript* код.

*Redux* надгражда *Flux* архитектурния шаблон за разработка, като позволява предсказуемо и мащабируемо управление на състоянието на клиентското уеб/мобилно приложение, чрез осигуряване на еднопосочен поток на обработка и *immutability* на данните + централизирано съхраняване на състоянието на приложението (*single store = single source of truth*). При *Redux* промяната на състоянието на приложението се осъществява с помощта на чисти (pure) функции, наречени – *reducers: (състояние, действие) => резултатно състояние*. *Redux* включва следните основни типове компоненти: *Actions, Action Creators, Reducers, Store, Selectors*.

Курсът разглежда задълбочено *React.js* и *Redux* от гледна точка на тяхната практическа употреба за реализиране на реални уеб и мобилни приложения (*SPA*). Включени са и по сложни елементи като асинхронни действия (*async actions) с използване на redux-thunk, redux-promise/ redux-promise-middleware/ redux-observable*, асинхронни потоци с *applyMiddleware()*, йерархично рутиране при вложени компоненти, интеграция на *React* рутера с *Redux* чрез използване на *Redux Router* и *React Router Redux* библиотеки, интерактивна визуализация на състоянието на приложението и отстраняване на грешки с *redux-devtools* и други. Специално внимание се обръща на използването на *immutable* структури от данни.

Втората част на курса разглежда разработката на уеб услуги в стил *REpresentational State Transfer (REST)* и *JSON APIs* (приложни интерфейси под

формата на уеб услуги). Разработката на тези услуги се извършва върху бързо развиващата се *Node.js* платформа, в комбинация с *Express.js* middleware надстройка и *LoopBack* фреймуърк. Включено е запознаване с *NoSQL* БД – *MongoDB*, както и *ODM (Object Document Mapper)* от типа на *Mongoose*.

Курсът завършва с разработка на цялостно клиент-сървър (*fullstack*) приложение с разгледаните технологии:

- *React + Redux SPA* клиент;
- *Node.js + Express + LoopBack + MongoDB JSON/REST API.*

Преобладаващата част от използваните материали са достъпни в електронен вид на английски език в Интернет. Кодът е достъпен в *GitHub* под свободен лиценз: https://github.com/iproduct/course-node-express-react

**Предварителни изисквания:**

Очаква се студентите да могат да боравят свободно с технически английски език. Необходимо е добро познаване на основните уеб дизайн технологии – HTML 5, CSS 3 и JavaScript/ ECMAScript 6, както и известен практически опит в разработката на уеб приложения с някоя от разпространените JS библиотеки от типа на jQuery.

**Важно:** Записването за курса включва **входящ тест** върху следния материал: *HTML 5, CSS 3, обектно-ориентиран и събитийно-ориентиран JavaScript, ECMAScript 6, responsive web design, jQuery.* Само успешно издържалите теста (с над 50%) студенти ще бъдат записани в курса. По преценка на преподавателя студенти могат да бъдат записвани и допълнително ако демонстрират компетентност в областта под формата на собствено JavaScript приложение с достатъчна сложност.

**Очаквани резултати:**

По време на курса студентите ще придобият и затвърдят знания и умения за:

- *обектно-ориентиран JavaScript (JS)* и новости в *ECMAScript 6, 7 и 8* - класове, конструктори, *let* и *var*, ламбда функции (=>), *destructuring*, *rest/spread operators*, *Promises*, *async/await*, *import()* и др.*;*
- JavaScript модули и модулни системи - *CommonJS* и *ES 6 (Harmony);*
- конфигуриране и използване на билд инструменти – *npm*, *webpack*;
- *Single Page Applications (SPA)* с *ReactJS, JSX* и *ECMAScript 6*;
- използване на *Redux* шаблона и *Immutable* структури от данни за създаване на лесни за поддръжка и разширяване уеб приложения с предсказуемо управление на състоянието;
- *HTTP* протокол и мултимедийни типове (*MIME* типове);
- софтуерен архитектурен стил *REST (REpresentational State Transfer)* и *HATEOAS (Hypermedia as the Engine of Application State);*
- разработка на уеб услуги (*APIs*) – *JavaScript Object Notation (JSON) APIs* на *Node.js + Express* сървърна платформа;
- съхраняване и извличане на данните в/от *NoSQL* база от данни *MongoDB;*
- използване на *LoopBack* фреймуърк за бързо изграждане на разширяеми и динамични *REST APIs* с минимално (или дори без) писане на програмен код;
- разработка на цялостни клиент-сървър (*fullstack*) приложения с *React + Redux SPA* клиент и *Node.js + Express + LoopBack + MongoDB JSON/REST API*.

Студентите ще изградят също умения за целенасочено търсене, анализиране и практическо използване на информация, както и умения за работа в екип по формулирано проектно задание.

***Учебно съдържание***

| № | Тема: | Хорариум |
|---|---|---|
| 1. | Представяне на курса, въведение в областта. Обектно-ориентиран *JavaScript*. Примитивни типове и обекти. Достъп до обектите по референция. Свойства функции и методи. Използване на *this*. Методи *call, apply* и *bind*. Прототипно наследяване (*prototypal inheritance*), полиморфизъм и предефиниране на методи (*method overriding*), класове и конструктори. Класово наследяване и използване на *instanceof*. Използване на *Visual Studio Code + linters* за разработка на *JS/ES6* приложения. | 4 |
| 2. | Новости в *ECMAScript 6* – класове, конструктори, *let* и *var*, ламбда функции (=>), *destructuring, rest/spread operators, Promises, async/await* и др. JavaScript модули и модулни системи – *CommonJS* и *ES 6*. Инструменти за изграждане на уеб приложения (*build toolchain) - npm, babel, babel-cli, webpack*. Конфигуриране на ES6/7 среда за разработка с *webpack, webpack-dev-server, Hot Module Replacement, development/production* конфигурации. | 4 |
| 3. | *Single Page Applications (SPA)* с *ReactJS, JSX* и *ECMAScript 6 – MV** шаблони. Изграждане на *React* компоненти и примерно приложение - *Comments Manager*. *React API*: *React, ReacrDOM, ReactDOMServer*. *React.createElement()* и *ReactDOM.render()* методи. Използване на *JSX* в *React*. Показване на динамични стойности. Композиране на компоненти с използване на свойства. Компоненти запазващи вътрешно състояние (*stateful components*) – установяване на начално състояние, промяна на състояние на компонент. Събития в *React*, обработка на *DOM* събития. Реализиране на TODO приложение. | 4 |
| 4. | *JSX* в дълбочина – разлики с HTML, трансформация към *JavaScript*, изрази, *if-else, immediately-invoked function expressions (IIFE)*. Композиция на компоненти – собственост (*ownership*), *this.props.children, React.Children utilities, child reconciliation, stateful children* и dynamic *children* с използване на ключове. Трансфер на | 4 |

| | | |
|---|---|---|
| | свойства. Използване на ES6 класове. Чисти (*stateless, pure*) функции като компоненти. Интеграция с други JavaScript библиотеки. Използване на *markdown* - *__html, dangerouslySetInnerHTML*. *Ajax* заявки с използване на *WHATWG Fetch API*. | |
| 5. | Жизнен цикъл при визуализация на *React* компонентите и *life cycle callbacks*. Двупосочна комуникация между компоненти собственик и дете (*owner-child*) с използване на *callbacks* подадени като свойства. Работа с форми – интерактивни свойства, контролирани и неконтролирани компоненти. Новости в *React.js – Dependency Injection (DI)* на собствени услуги с използване на *React Context*. Използване на *addons*. Двупосочно свързване с помощта на *two way binding helpers addon*. *Refs* към компоненти. *Higher-order* функции и *higher-order* компоненти (*HOCs*). Използване на библиотеката *Recompose*. | 4 |
| 6. | *React* Router *v4* – варианти: *<BrowserRouter>*, *<HashRouter>*, *<MemoryRouter>*. Дефиниране на маршрути с използване на *<Route>* компоненти – *component, render* и *children* свойства. Свойства подавани на рендерирания компонент – *match, location* и *history*. Индексни пътища и *exact matching*. Параметри на пътя – *props.match.params*. Шаблони за маршрути и опционални параметри на маршрути. *Path-to-RegExp* съпоставяне. Използване на *withRouter HOC*. *<Link>* връзки, активни пътища и *<NavLink>*. Едновременна навигация към няколко компонента. Избор на един от няколко маршрута със *<Switch>*. Пренасочване на маршрути с *<Redirect>*. Използване на маршрути без пътища. Рекурсивни пътища. Анимация при рутиране. Програмна навигация. Контролиране на навигацията и състоянието на приложението чрез дефиниране на *componentDidMount/ componentWillMount*, *componentDidUpdate/ componentWillUpdate* и *componentWillUnmount* "куки". *<Prompt>* компонент. *Code splitting* и *lazy loading* на модули при рутиране. | 4 |

| | | |
|---|---|---|
| 7. | Използване на *Redux* шаблон и *Immutable* структури от данни за създаване на лесни за поддръжка и разширение уеб приложения с предсказуемо управление на състоянието. Архитектурен шаблон: *Flux* – еднопосочен поток на данните, основни компоненти. Основни принципи на *Redux*: единствено хранилище за данни, състоянието е само за четене, промените се правят чрез "чисти" функции – *reducers*. Основни компоненти на *Redux – Actions, Action Creators, Reducers, Stores, Selectors*. Взаимодействие между *Redux* компоненти – проектиране на представяне на състоянието, обработка действия (*Actions)* с използване на отделни *Reducers*, комбиниране на *Reducers* в *Root Reducer* с използване на *combineReducers()*, регистриране на слушатели за промяна в състоянието чрез *Store.subscribe()*, иницииране на *Actions* чрез *Store.dispatch()*. Свързване на *Redux* с *React* чрез използване на *React Redux* bindings. Два вида компоненти – презентационни и контейнер компоненти. Реализация на *TODO* приложение с използване на *React* и *Redux*. | 4 |
| 8. | *Redux в дълбочина* – нормализация на вложени обектни структури (*JSON API* отговори*)* с използване на *normalizr*. Композируеми селектор функции и *memoization* с *Reselect.* Реализация на *async actions* с използване на *redux-thunk/ redux-promise/ redux-promise-middleware/ redux-observable*. Асинхронни потоци от данни с *applyMiddleware()*, примери. Използване на *redux-devtools*. Изграждане на *SPA* с *Redux* и *React Router*. Разширена *Redux* маршрутизация с *Redux Router* и *React Router Redux* библиотеки. Най-чести грешки при използване на *immutable* структури – използване на *Immutable.js*. | 4 |
| 9. | Разработка на сървърни приложения с *Node.js* – неблокиращ вход-изход (*IO*). Цикъл на събитията (*event loop*). Използване на *callbacks* и *promises*. *Continuation-passing style (CPS)* и *Node.js*. *npm* мениджър на пакети. Изпълнение на скриптове с *Node.js shell (REPL).* *CommonJS* модули и тяхното използване. Глобални | 8 |

| | | |
|---|---|---|
| | обекти в *Node.js*. Разработка на *HTTP* сървъри и клиенти с използване на *HTTP*. Маршрутизация. *HTTP* методи. Разработка на HTTP клиенти с *http.get()* и *http.request()*. Създаване на собствени модули. Сервиране на статични файлове (*File System module*) – asynchronous vs. synchronous, completion callbacks, using promises. Буфери. Основни операции с файлове и директории. Потоци, събития и пайпове – *EventEmitter*, собствени събития. *readable, writable, duplex* и *transform* потоци. Примери: *HTTP requests / responses, fs write streams, zlib streams, stdin, stdout*. Свързване на потоци с използване на *pipe()*. Реализация на собствени *readable, writable* и *transform* потоци. | |
| 10. | Разработка на уеб услуги *(APIs)* с *Express.js* (+ *Node.js*). Маршрутизация и обработка на заявки (*request handlers*). Генериране на Express приложение с използване на *express-generator (CLI)*. Структура на приложението. *Model-view-controller (MVC).Express API* - конфигуриране, настройки, среда. Работа с *Middleware* – дефиниране на middleware функции (*continuation passing style*), прилагане на middleware, монтиране към пътища. Параметри и маршрутизация. *Middleware* на ниво приложение (*app.use()* и *app.METHOD()* ), *middleware* на ниво маршрутизатор (*router.use()* и *router.METHOD() )*. Обработка на грешки. Вграден *middleware* (*express.static*). Използване на готови *middleware* модули. Съхраняване и извличане на данните в/от *NoSQL* база от данни *MongoDB*. Софтуерен архитектурен стил *REST (REpresentational State Transfer)*. Изграждане на *RESTful / JSON APIs* с *Express*. | 8 |
| 11. | Използване на *LoopBack* фреймуърк за бързо изграждане на разширяеми и динамични *REST APIs* с минимално (или дори без) писане на програмен код. Създаване на ново приложение с *LoopBack application generator*. Създаване на модел на данните с *LoopBack model generator*. Тестване на генерирано приложение с *LoopBack API Explorer*. | 12 |

| | | |
|---|---|---|
| | Свързване с база от данни (*data source - MongoDB*) с използване на *Data source generator* и добавяне на тестови данни. Добавяне на собствени *отдалечени методи (remote methods)*. Публикуване на статични уеб ресурси с *Express middleware*. Добавяне на маршрути. Синхронни и асинхронни стартиращи скриптове. Използване на *миксини*. *Middleware* фази. Реализация и регистриране на собствен *middleware*. Обработка на грешки. *LoopBack* събития. Използване на *promises*. Използване на готови *REST APIs* – управление на потребителите, аутентикация, оторизация и права. Дефиниране на модели - връзки, валидация. Разработка на цялостно клиент-сървър приложение с *React + Redux SPA* клиент и *Node.js + Express + LoopBack + MongoDB JSON/REST API*. | |

***Конспект за изпит***

| № | Въпрос |
|---|---|
| 1. | Представяне на курса, въведение в областта. Обектно-ориентиран *JavaScript*. Примитивни типове и обекти. Достъп до обектите по референция. Свойства функции и методи. Използване на *this*. Методи *call, apply* и *bind*. Прототипно наследяване (*prototypal inheritance*), полиморфизъм и предефиниране на методи (*method overriding*), класове и конструктори. Класово наследяване и използване на *instanceof*. Използване на *Visual Studio Code + linters* за разработка на *JS/ES6* приложения. |
| 2. | Новости в *ECMAScript 6* – класове, конструктори, *let* и *var*, ламбда функции (=>), *destructuring, rest/spread operators, Promises, async/await* и др. JavaScript модули и модулни системи – *CommonJS* и *ES 6*. Инструменти за изграждане на уеб приложения (*build toolchain) - npm, babel, babel-cli, webpack*. Конфигуриране на ES6/7 среда за разработка с *webpack, webpack-dev-server, Hot Module Replacement, development/production* конфигурации. |
| 3. | *Single Page Applications (SPA)* с *ReactJS, JSX* и *ECMAScript 6* – *MV** шаблони. Изграждане на *React* компоненти и примерно приложение - *Comments Manager*. *React API*: *React, ReacrDOM, ReactDOMServer*. *React.createElement()* |

| | |
|---|---|
| | и *ReactDOM.render()* методи. Използване на *JSX* в *React*. Показване на динамични стойности. Композиране на компоненти с използване на свойства. Компоненти запазващи вътрешно състояние (*stateful components*) – установяване на начално състояние, промяна на състояние на компонент. Събития в *React*, обработка на *DOM* събития. Реализиране на TODO приложение. |
| 4. | *JSX* в дълбочина – разлики с HTML, трансформация към *JavaScript*, изрази, *if-else, immediately-invoked function expressions (IIFE)*. Композиция на компоненти – собственост (*ownership*), *this.props.children, React.Children utilities, child reconciliation, stateful children* и dynamic *children* с използване на ключове. Трансфер на свойства. Използване на ES6 класове. Чисти (*stateless, pure*) функции като компоненти. Интеграция с други JavaScript библиотеки. Използване на *markdown - __html, dangerouslySetInnerHTML*. *Ajax* заявки с използване на *WHATWG Fetch API*. |
| 5. | Жизнен цикъл при визуализация на *React* компонентите и *life cycle callbacks*. Двупосочна комуникация между компоненти собственик и дете (*owner-child*) с използване на *callbacks* подадени като свойства. Работа с форми – интерактивни свойства, контролирани и неконтролирани компоненти. Новости в *React.js – Dependency Injection (DI)* на собствени услуги с използване на *React Context*. Използване на *addons*. Двупосочно свързване с помощта на *two way binding helpers addon*. *Refs* към компоненти. *Higher-order* функции и *higher-order* компоненти (*HOCs*). Използване на библиотеката *Recompose*. |
| 6. | *React* Router *v4* – варианти: *<BrowserRouter>, <HashRouter>, <MemoryRouter>*. Дефиниране на маршрути с използване на *<Route>* компоненти – *component, render* и *children* свойства. Свойства подавани на рендерирания компонент – *match, location* и *history*. Индексни пътища и *exact matching*. Параметри на пътя – *props.match.params*. Шаблони за маршрути и опционални параметри на маршрути. *Path-to-RegExp* съпоставяне. Използване на *withRouter HOC*. *<Link>* връзки, активни пътища и *<NavLink>*. Едновременна навигация към няколко компонента. Избор на един от няколко маршрута със *<Switch>*. Пренасочване на маршрути с *<Redirect>*. Използване на маршрути без пътища. Рекурсивни пътища. Анимация при рутиране. Програмна навигация. Контролиране на навигацията и състоянието на приложението чрез дефиниране на *componentDidMount/ componentWillMount, componentDidUpdate/ componentWillUpdate* и *componentWillUnmount* "куки". *<Prompt>* компонент. *Code splitting* и *lazy loading* на модули при рутиране. |

| | |
|---|---|
| 7. | Използване на *Redux* шаблон и *Immutable* структури от данни за създаване на лесни за поддръжка и разширение уеб приложения с предсказуемо управление на състоянието. Архитектурен шаблон: *Flux* – еднопосочен поток на данните, основни компоненти. Основни принципи на *Redux*: единствено хранилище за данни, състоянието е само за четене, промените се правят чрез "чисти" функции – *reducers*. Основни компоненти на *Redux – Actions, Action Creators, Reducers, Stores, Selectors*. Взаимодействие между *Redux* компоненти – проектиране на представяне на състоянието, обработка действия (*Actions)* с използване на отделни *Reducers*, комбиниране на *Reducers* в *Root Reducer* с използване на *combineReducers()*, регистриране на слушатели за промяна в състоянието чрез *Store.subscribe()*, иницииране на *Actions* чрез *Store.dispatch()*. Свързване на *Redux* с *React* чрез използване на *React Redux* bindings. Два вида компоненти – презентационни и контейнер компоненти. Реализация на *TODO* приложение с използване на *React* и *Redux*. |
| 8. | *Redux в дълбочина* – нормализация на вложени обектни структури (*JSON API* отговори*)* с използване на *normalizr*. Композируеми селектор функции и *memoization* с *Reselect.* Реализация на *async actions* с използване на *redux-thunk/ redux-promise/ redux-promise-middleware/ redux-observable*. Асинхронни потоци от данни с *applyMiddleware()*, примери. Използване на *redux-devtools*. Изграждане на *SPA* с *Redux* и *React Router*. Разширена *Redux* маршрутизация с *Redux Router* и *React Router Redux* библиотеки. Най-чести грешки при използване на *immutable* структури – използване на *Immutable.js*. |
| 9. | Разработка на сървърни приложения с *Node.js* – неблокиращ вход-изход (*IO*). Цикъл на събитията (*event loop*). Използване на *callbacks* и *promises*. *Continuation-passing style (CPS)* и *Node.js*. *npm* мениджър на пакети. Изпълнение на скриптове с *Node.js shell (REPL). CommonJS* модули и тяхното използване. Глобални обекти в *Node.js*. Разработка на *HTTP* сървъри и клиенти с използване на *HTTP*. Маршрутизация. *HTTP* методи. Разработка на HTTP клиенти с *http.get()* и *http.request()*. Създаване на собствени модули. Сервиране на статични файлове (*File System module*) – asynchronous vs. synchronous, completion callbacks, using promises. Буфери. Основни операции с файлове и директории. Потоци, събития и пайпове – *EventEmitter*, собствени събития. *readable, writable, duplex* и *transform* потоци. Примери: *HTTP requests / responses, fs write streams, zlib streams, stdin, stdout*. Свързване на потоци с използване на *pipe()*. Реализация на собствени *readable, writable* и |

| | |
|---|---|
| | *transform* потоци. |
| 10. | Разработка на уеб услуги *(APIs)* с *Express.js* (+ *Node.js*). Маршрутизация и обработка на заявки (*request handlers*). Генериране на Express приложение с използване на *express-generator (CLI)*. Структура на приложението. *Model-view-controller (MVC).Express API* - конфигуриране, настройки, среда. Работа с *Middleware* – дефиниране на middleware функции (*continuation passing style*), прилагане на middleware, монтиране към пътища. Параметри и маршрутизация. *Middleware* на ниво приложение (*app.use()* и *app.METHOD()* ), *middleware* на ниво маршрутизатор (*router.use()* и *router.METHOD() ).* Обработка на грешки. Вграден *middleware* (*express.static*). Използване на готови *middleware* модули. Съхраняване и извличане на данните в/от *NoSQL* база от данни *MongoDB*. Софтуерен архитектурен стил *REST (REpresentational State Transfer).* Изграждане на *RESTful / JSON APIs* с *Express*. |
| 11. | Използване на *LoopBack* фреймуърк за бързо изграждане на разширяеми и динамични *REST APIs* с минимално (или дори без) писане на програмен код. Създаване на ново приложение с *LoopBack application generator*. Създаване на модел на данните с *LoopBack model generator*. Тестване на генерирано приложение с *LoopBack API Explorer*. Свързване с база от данни (*data source - MongoDB*) с използване на *Data source generator* и добавяне на тестови данни. Добавяне на собствени *отдалечени методи (remote methods).* Публикуване на статични уеб ресурси с *Express middleware*. Добавяне на маршрути. Синхронни и асинхронни стартиращи скриптове. Използване на *миксини*. *Middleware* фази. Реализация и регистриране на собствен *middleware*. Обработка на грешки. *LoopBack* събития. Използване на *promises*. Използване на готови *REST APIs* – управление на потребителите, аутентикация, оторизация и права. Дефиниране на модели - връзки, валидация. Разработка на цялостно клиент-сървър приложение с *React + Redux SPA* клиент и *Node.js + Express + LoopBack + MongoDB JSON/REST API*. |

## *Библиография*

***Основна:***


1. Уеб сайт на *W3C* – http://w3.org
2. Уеб сайт на платформата *Node.js* – https://nodejs.org/en/
3. Уеб сайт на *Express.js* – http://expressjs.com/
4. Уеб сайт на *LoopBack* рамка за разработка на RESTful APIs на IBM – https://loopback.io/
5. Уеб сайт на библиотеката *React.js* – https://facebook.github.io/react/
6. Уеб сайт на проекта *Redux* – http://redux.js.org/
7. Уеб сайт на NoSQL база от данни *MongoDB* – https://www.mongodb.com/
8. Уеб сайт на *Webpack* – https://webpack.github.io/
9. Сайт на *Mozilla Developer Network* за съвременни уеб технологии – https://developer.mozilla.org/
10. Osmani, A., *Learning JavaScript Design Patterns*, O'Reilly - Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 – http://addyosmani.com/resources/essentialjsdesignpatterns/book/
11. Mardan, A., *Pro Express.js: Master Express.js: The Node.js Framework For Your Web Development*. APress, 2014, ISBN-10: 1484200381
12. Augarten, B., Kuo, M., Lin, E., Shaikh, A., Soriani, F., Tisserand, G., Zhang, C., Zhang, K, *Express.js Blueprints*. Packt Publishing, 2015, ISBN 978-1-78398-302-5


**Дата:**
5 декември 2017 г

**Съставил:**
ас. Траян Илиев