Java REST Server

# Questionnaire

|  |
| --- |
| @Document(collection = **"questionnaires"**) **public class** Questionnaire {  @Id  **private** String **id**;   @NotNull  @Size(min = 2, max = 30)  **private** String **title**;   @NotNull  @Size(min = 10, max = 50)  **private** String **description**;   **public void** setId(String id) {  **this**.**id** = id;  }   **public** String getId() {  **return id**;  }   **public void** setTitle(String title) {  **this**.**title** = title;  }   **public** String getTitle() {  **return title**;  }   **public void** setDescription(String description) {  **this**.**description** = description;  }   **public** String getDescription() {  **return description**;  }   @Override  **public int** hashCode() {  **return** Objects.*hashCode*(**title**, **description**, **id**);  }   @Override  **public boolean** equals(**final** Object obj) {  **if** (obj **instanceof** Questionnaire) {  **final** Questionnaire other = (Questionnaire) obj;  **return** Objects.*equal*(**id**, other.**id**)  && Objects.*equal*(**title**, other.**title**)  && Objects.*equal*(**description**, other.**description**);  } **else** {  **return false**;  }  } } |

# QuestionnaireController

|  |
| --- |
| @RestController @CrossOrigin(origins = **"\*"**) @RequestMapping(**"/questionnaires"**) **public class** QuestionnaireController {   **private final** QuestionnaireRepository **questionnaireRepository**;   @Autowired  **public** QuestionnaireController(QuestionnaireRepository questionnaireRepository) {  **this**.**questionnaireRepository** = questionnaireRepository;  }   @GetMapping()  **public** ResponseEntity<List<Questionnaire>> findAll() {  Sort sort = **new** Sort(Sort.Direction.***ASC***, **"id"**);  List<Questionnaire> questionnaires = **questionnaireRepository**.findAll(sort);  **return new** ResponseEntity<>(questionnaires, HttpStatus.***OK***);  }   @GetMapping(**"/{id}"**)  **public** ResponseEntity<Questionnaire> findById(@PathVariable String id) {  Optional<Questionnaire> questionnaire = **questionnaireRepository**.findById(id);  **if** (questionnaire.isPresent()) {  **return new** ResponseEntity<>(questionnaire.get(), HttpStatus.***OK***);  } **else** {  **return new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);  }  }   @PostMapping  **public** ResponseEntity<Questionnaire> createQuestionnaire(@Valid @RequestBody Questionnaire questionnaire, BindingResult result) {  **if** (!result.hasErrors()) {  questionnaire = **questionnaireRepository**.save(questionnaire);  **return new** ResponseEntity<>(questionnaire, HttpStatus.***CREATED***);  } **else** {  **return new** ResponseEntity<>(HttpStatus.***PRECONDITION\_FAILED***);  }  }   @PutMapping(**"/{id}"**)  **public** ResponseEntity<Questionnaire> updateQuestionnaire(@PathVariable String id, @Valid @RequestBody Questionnaire questionnaire, BindingResult result) {  **if** (!result.hasErrors()) {  Optional<Questionnaire> tempQuestionnaireOptional = **questionnaireRepository**.findById(id);  **if** (tempQuestionnaireOptional.isPresent()) {  Questionnaire tempQuestionnaire = tempQuestionnaireOptional.get();  tempQuestionnaire.setTitle(questionnaire.getTitle());  tempQuestionnaire.setDescription(questionnaire.getDescription());  **questionnaireRepository**.save(tempQuestionnaire);  **return new** ResponseEntity<>(tempQuestionnaire, HttpStatus.***OK***);  } **else** {  **return new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);  }  } **else** {  **return new** ResponseEntity<>(HttpStatus.***PRECONDITION\_FAILED***);  }  }   @DeleteMapping(**"/{id}"**)  **public** ResponseEntity<String> deleteQuestionnaire(@PathVariable String id) {  Optional<Questionnaire> questionnaire = **questionnaireRepository**.findById(id);  **if** (questionnaire.isPresent()) {  **questionnaireRepository**.deleteById(id);  **return new** ResponseEntity<>(HttpStatus.***OK***);  } **else** {  **return new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);  }  } } |

# QuestionnaireControllerTest

|  |
| --- |
| @RunWith(SpringRunner.**class**) @WebMvcTest(QuestionnaireController.**class**) **public class** QuestionnaireControllerTest {   @Autowired  **private** MockMvc **mockMvc**;   @MockBean  **private** QuestionnaireRepository **questionnaireRepositoryMock**;   @Before  **public void** setUp() {  Mockito.*reset*(**questionnaireRepositoryMock**);  }   @Test  **public void** findById\_QuestionnaireNotExisting\_ShouldReturnNotFound() **throws** Exception {  **mockMvc**.perform(*get*(**"/questionnaires/{id}"**, 2L)  .header(**"Accept"**, **"application/json"**)  )  .andExpect(*status*().isNotFound());  Mockito.*verify*(**questionnaireRepositoryMock**, *times*(0)).findById(**"1"**);  }   @Test  **public void** findAll\_QuestionnairesFound\_ShouldReturnFoundQuestionnaires() **throws** Exception {  Questionnaire questionnaire1 = **new** QuestionnaireBuilder(Long.*toString*(1))  .description(**"MyDescription1"**)  .title(**"MyTitle1"**)  .build();   Questionnaire questionnaire2 = **new** QuestionnaireBuilder(Long.*toString*(2))  .description(**"MyDescription2"**)  .title(**"MyTitle2"**)  .build();   *when*(**questionnaireRepositoryMock**.findAll(Sort.*by*(**"id"**))).thenReturn(Arrays.*asList*(questionnaire1, questionnaire2));   **mockMvc**.perform(*get*(**"/questionnaires"**)  .header(**"Accept"**, **"application/json"**)  )  *//.andDo(print())* .andExpect(*status*().isOk())  .andExpect(*jsonPath*(**"$[0].id"**, *is*(**"1"**)))  .andExpect(*jsonPath*(**"$[0].title"**, *is*(**"MyTitle1"**)))  .andExpect(*jsonPath*(**"$[0].description"**, *is*(**"MyDescription1"**)))  .andExpect(*jsonPath*(**"$[1].id"**, *is*(**"2"**)))  .andExpect(*jsonPath*(**"$[1].title"**, *is*(**"MyTitle2"**)))  .andExpect(*jsonPath*(**"$[1].description"**, *is*(**"MyDescription2"**)));  Mockito.*verify*(**questionnaireRepositoryMock**, *times*(1)).findAll(Sort.*by*(**"id"**));  }   @Test  **public void** create\_NewQuestionnaire\_ShouldReturnOK() **throws** Exception {  Questionnaire questionnaire = **new** QuestionnaireBuilder(Long.*toString*(1))  .description(**"My Description"**)  .title(**"My Title"**)  .build();   *// Important: You must override Questionnaire.equals() to be able to execute these mock calls!  when*(**questionnaireRepositoryMock**.save(questionnaire)).thenReturn(questionnaire);   **mockMvc**.perform(*post*(**"/questionnaires"**)  .contentType(MediaType.***APPLICATION\_JSON***)  .content(TestUtil.*convertObjectToJsonBytes*(questionnaire)))  .andExpect(*status*().isCreated())  .andExpect(*jsonPath*(**"$.id"**, *is*(**"1"**)))  .andExpect(*jsonPath*(**"$.title"**, *is*(**"My Title"**)))  .andExpect(*jsonPath*(**"$.description"**, *is*(**"My Description"**)));  Mockito.*verify*(**questionnaireRepositoryMock**, *times*(1)).save(questionnaire);  }   @Test  **public void** create\_NewQuestionnaireWithEmptyTitle\_ShouldReturnNOK() **throws** Exception {  Questionnaire questionnaire = **new** QuestionnaireBuilder(Long.*toString*(1))  .description(**"MyDescription"**)  .build();   *// Important: You must override Questionnaire.equals() to be able to execute these mock calls!  when*(**questionnaireRepositoryMock**.save(questionnaire)).thenReturn(questionnaire);   **mockMvc**.perform(*post*(**"/questionnaires"**)  .contentType(MediaType.***APPLICATION\_JSON***)  .content(TestUtil.*convertObjectToJsonBytes*(questionnaire)))  .andExpect(*status*().isPreconditionFailed());  Mockito.*verify*(**questionnaireRepositoryMock**, *times*(0)).save(questionnaire);  }   @Test  **public void** update\_questionnaire\_ShouldReturnOK() **throws** Exception {  Questionnaire updatedQuestionnaire = **new** QuestionnaireBuilder(Long.*toString*(1))  .description(**"MyDescription"**)  .title(**"MyTitle"**)  .build();   Optional<Questionnaire> qOptional = Optional.*of*(updatedQuestionnaire);   *when*(**questionnaireRepositoryMock**.findById(**"1"**)).thenReturn(qOptional);  *when*(**questionnaireRepositoryMock**.save(updatedQuestionnaire)).thenReturn(updatedQuestionnaire);   **mockMvc**.perform(*put*(**"/questionnaires/1"**)  .contentType(MediaType.***APPLICATION\_JSON***)  .content(TestUtil.*convertObjectToJsonBytes*(updatedQuestionnaire)))  .andExpect(*status*().isOk())  .andExpect(*jsonPath*(**"$.id"**, *is*(**"1"**)))  .andExpect(*jsonPath*(**"$.title"**, *is*(**"MyTitle"**)))  .andExpect(*jsonPath*(**"$.description"**, *is*(**"MyDescription"**)));  Mockito.*verify*(**questionnaireRepositoryMock**, *times*(1)).save(updatedQuestionnaire);  }   @Test  **public void** delete\_questionnaire\_ShouldReturnOK() **throws** Exception {  Questionnaire questionnaire = **new** QuestionnaireBuilder(Long.*toString*(1))  .description(**"MyDescription"**)  .title(**"MyTitle"**)  .build();   Optional<Questionnaire> qOptional = Optional.*of*(questionnaire);   *when*(**questionnaireRepositoryMock**.findById(**"1"**)).thenReturn(qOptional);   **mockMvc**.perform(*delete*(**"/questionnaires/1"**)  .contentType(MediaType.***APPLICATION\_JSON***))  .andExpect(*status*().isOk());  Mockito.*verify*(**questionnaireRepositoryMock**, *times*(1)).deleteById(**"1"**);  } } |

Node.js REST Server

# app.js

|  |
| --- |
| **"use strict"**;  **const** log4js = require(**'log4js'**); **const** dotenv = require(**'dotenv-extended'**); **const** express = require(**'express'**); **const** bodyParser = require(**'body-parser'**); **const** cors = require(**'cors'**); **const** mongoose = require(**'mongoose'**); **const** dispatcher = require(**'./web/questionnaire-controller'**);  *// Read the properties from file '.env' and '.env.defaults'. // silent: true - no log message is shown when missing the .env or .env.defaults files.* dotenv.load({**silent**: **true**}); **const** PORT = process.env.PORT || 9090;  *// Create and configure the logger* **const** logger = log4js.getLogger(**'server'**); log4js.configure(**'log4js.json'**);  *// Establish the database connection* **const** url = **'mongodb://'** + process.env.MONGO\_HOST + **'/'** + process.env.MONGO\_DATABASE; mongoose.connect(url, {  **useNewUrlParser**: **true** });  *// Create the app* **const** app = express();  *// Install the middleware* app.use(bodyParser.json()); app.use(cors());  *// Install the dispatcher* app.use(**'/flashcard-express'**, dispatcher);  *// Start the application* app.listen(PORT);  *// Put a friendly message on the terminal* logger.info(**`Server running on** ${PORT}**`**);  module.**exports** = app; |

# .env

|  |
| --- |
| PORT=7000 MONGO\_HOST=172.17.0.2:27017 MONGO\_DATABASE=webfr MONGO\_USER= MONGO\_PASS= |

# log4j.json

|  |
| --- |
| {  **"appenders"**: {  **"out"**: {  **"type"**: **"console"** },  **"logfile"**: {  **"type"**: **"file"**,  **"filename"**: **"logfile.log"** }  },  **"categories"**: {  **"default"**: {  **"appenders"**: [  **"out"** ],  **"level"**: **"error"** },  **"app"**: {  **"appenders"**: [  **"logfile"**,  **"out"** ],  **"level"**: **"debug"** },  **"controller"**: {  **"appenders"**: [  **"out"** ],  **"level"**: **"debug"** },  **"server"**: {  **"appenders"**: [  **"out"** ],  **"level"**: **"debug"** }  } } |

# questionnaire.js

|  |
| --- |
| **"use strict"**;  **const** mongoose = require(**'mongoose'**); **const** Schema = mongoose.Schema;  */\*  \* Define the database schema for entity 'Questionnaire'.  \* ATTENTION: Must be compatible to the Spring Mongo document.  \*/* **const** questionnaireSchema = **new** Schema(  {  **title**: {  **type**: ***String***,  **min**: 2,  **max**: 30  },  **description**: {  **type**: ***String***,  **min**: 10,  **max**: 50  }  },  {  **collection**: **'questionnaires'** } );  */\*  \* We need a field called 'id' to be compatible with Spring Mongo document.  \* Make it virtual so this field is not persisted.  \*/* questionnaireSchema.virtual(**'id'**).get(**function** () {  **return this**.**\_id**.toHexString(); });  */\*  \* Activate the usage of virtual fields if toJSON method is called.  \*/* questionnaireSchema.set(**'toJSON'**, {  **virtuals**: **true** });  module.**exports** = mongoose.model(**'Questionnaire'**, questionnaireSchema); |

# questionnaire-controller.js

|  |
| --- |
| **"use strict"**;  **const** log4js = require(**'log4js'**); **const** express = require(**'express'**); **const Questionnaire** = require(**'./../domain/questionnaire'**);  *// Get the dispatcher* **const** dispatcher = express.Router();  *// Get the logger* **const** logger = log4js.getLogger(**'controller'**);  *// Show all questionnaires* dispatcher.route(**'/questionnaires'**).get((request, response) => {  **Questionnaire**.find((error, questionnaires) => {  **if** (error) {  **return** response.status(400).send(**'Database error'**);  } **else** {  response.status(200).json(questionnaires);  logger.debug(**`Found** ${questionnaires.**length**} **questionnaires`**);  }  }); });  *// Show a single questionnaire* dispatcher.route(**'/questionnaires/:id'**).get((request, response) => {  **Questionnaire**.findById(request.params.**id**, (error, questionnaire) => {  **if** (error) {  **return** response.status(400).send(**'Database error'**);  } **else** {  response.status(200).json(questionnaire);  logger.debug(**`Found a questionnaire with ID** ${questionnaire.**id**}**`**);  }  }); });  *// Create a new questionnaire* dispatcher.route(**'/questionnaires'**).post((request, response) => {  **const** questionnaire = **new Questionnaire**();  questionnaire.**title** = request.**body**.**title**;  questionnaire.**description** = request.**body**.description;  questionnaire.save((error, fullquestionnaire) => {  **if** (error) {  **return** response.status(400).send(**'Database error'**);  } **else** {  response.status(200).json(fullquestionnaire);  logger.debug(**`Created a questionnaire with ID** ${fullquestionnaire.**id**}**`**);  }  }); });  *// Update a questionnaire* dispatcher.route(**'/questionnaires/:id'**).put((request, response) => {  **Questionnaire**.findById(request.params.**id**, (error, questionnaire) => {  **if** (error) {  **return** response.status(400).send(**'Database error'**);  } **else** {  questionnaire.**title** = request.**body**.**title**;  questionnaire.**description** = request.**body**.description;  questionnaire.save((error, fullquestionnaire) => {  **if** (error) {  **return** response.status(400).send(**'Database error'**);  } **else** {  response.status(200).json(fullquestionnaire);  logger.debug(**`Updated a questionnaire with ID** ${fullquestionnaire.**id**}**`**);  }  });  }  }); });  *// Delete a questionnaire* dispatcher.route(**'/questionnaires/:id'**).delete((request, response) => {  **Questionnaire**.deleteOne({**\_id**: request.params.**id**}, (error) => {  **if** (error) {  **return** response.status(400).send(**'Database error'**);  } **else** {  response.send();  logger.debug(**`Removed a questionnaire with ID** ${request.params.**id**}**`**);  }  }); });  *// export dispatcher to be able to use it outside of this module* module.**exports** = dispatcher; |

Simple React App

# application.json

|  |
| --- |
| {  **"url"**: **"http://localhost:8080/flashcard-mvc/questionnaires"**,  **"subtitle"**: **"React SPA App"** } |

# index.html

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**>  <**head**>  <**meta charset="utf-8"**>  <**link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico"**>  <**meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"**>  <**meta name="theme-color" content="#000000"**>  *<!--  manifest.json provides metadata used when your web app is added to the  homescreen on Android. See https://developers.google.com/web/fundamentals/web-app-manifest/  -->* <**link rel="manifest" href="%PUBLIC\_URL%/manifest.json"**>  *<!--  Notice the use of %PUBLIC\_URL% in the tags above.  It will be replaced with the URL of the `public` folder during the build.  Only files inside the `public` folder can be referenced from the HTML.   Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will  work correctly both with client-side routing and a non-root public URL.  Learn how to configure a non-root public URL by running `npm run build`.  -->* <**title**>React App</**title**>  </**head**>  <**body**>  <**noscript**>  You need to enable JavaScript to run this app.  </**noscript**>  <**div id="root"**></**div**>  *<!--  This HTML file is a template.  If you open it directly in the browser, you will see an empty page.   You can add webfonts, meta tags, or analytics to this file.  The build step will place the bundled scripts into the <body> tag.   To begin the development, run `npm start` or `yarn start`.  To create a production bundle, use `npm run build` or `yarn build`.  -->* </**body**> </**html**> |

# index.js

|  |
| --- |
| **import** React **from 'react'**; **import** ReactDOM **from 'react-dom'**; **import** App **from './app/App'**;  **import 'bootstrap/dist/css/bootstrap.min.css'**;  ReactDOM.render(<**App** />, ***document***.getElementById(**'root'**)); |

# App.js

|  |
| --- |
| **class** App **extends** Component {   constructor(props) {  **super**(props);  **this**.**state** = {  **url**: **''**,  **subtitle**: **''**,  **error**: **''** }  }   componentDidMount() {  **let** app = **this**;  *fetch*(**'application.json'**).then(response => {  **if** (response.ok) {  **return** response.json();  } **else** {  **throw new *Error***(response.statusText);  }  }).then(config => {  app.setState({  **url**: config.url,  **subtitle**: config.**subtitle** })  }).catch(exception => {  app.setState({  **error**: **'Unable to load and parse the application configuration: '** + exception.getError()  })  })  }   render() {  **let** component = (**this**.**state**.**error** === **''**) ? <**Loader**/> : <**Message message=**{**this**.**state**.**error**}/>;  **return** (  <**Container fluid**>  <**Header  title="Flashcard Client with React"  subtitle=**{**this**.**state**.**subtitle**}  />  {(**this**.**state**.**url** !== **''**) ? (<**QuestionnaireContainer serverUrl=**{**this**.**state**.**url**}/>) : (component)}  </**Container**>  );  } }  **export default** App; |

# Header.js

|  |
| --- |
| **import** React **from 'react'**; **import** {Jumbotron} **from 'reactstrap'**;  **const** *Header* = ({title, subtitle}) => (  <**Jumbotron**>  <**h1**>{title}</**h1**>  <**h3**>{subtitle}</**h3**>  </**Jumbotron**> );  **export default** *Header*; |

# Loader.js

|  |
| --- |
| **class** Loader **extends** React.Component {  render() {  **return** (  <**div style=**{centerStyle}><**img src=**{loader} **alt="Loading..."**/></**div**>  )  } }  **const** centerStyle = {  **textAlign**: **'center'** };  **export default** Loader; |

# Message.js

|  |
| --- |
| **class** Message **extends** React.Component {  render() {  **return** (  <**Alert color="danger"**>{**this**.**props**.**message**}</**Alert**>  )  } }  **export default** Message; |

# QuestionnaireRestService.js

|  |
| --- |
| **class** QuestionnairesRestService {  **set** serverUrl(url) {  **this**.**\_serverUrl** = url;  }   checkIfPropertiesAreSet() {  **if** (!**this**.**\_serverUrl**) {  **throw *Error***(**'serverUrl not set'**);  }  }   */\*\*  \* Returns all questionnaires as array of questionnaire objects  \*/* **async** getAll() {  **this**.checkIfPropertiesAreSet();  **const** response = **await** *fetch*(**this**.**\_serverUrl**);  *// Convert to JSON questionnaires* **return** response.json()  }   */\*\*  \* Deletes given questionnaire.  \*  \** ***@param*** *id id of questionnaire to delete  \*/* **async delete**(id) {  **this**.checkIfPropertiesAreSet();  **const** request = **new *Request***(**this**.**\_serverUrl** + **'/'** + id, {  **method**: **'DELETE'** });  **const** response = **await** *fetch*(request);  **return** response.ok;  }   */\*\*  \* Saves given questionnaire.  \*  \** ***@param*** *{\*} questionnaire questionnaire to save  \*/* **async** save(questionnaire) {  **this**.checkIfPropertiesAreSet();  **const** request = **new *Request***(**this**.**\_serverUrl**, {  **method**: **'POST'**,  **headers**: **new *Headers***({  **'Content-Type'**: **'application/json'** }),  **body**: ***JSON***.stringify(questionnaire)  });  **const** response = **await** *fetch*(request);  **return** response.json()  }   */\*\*  \* Updates given questionnaire.  \*  \** ***@param*** *{questionnaire} questionnaire questionnaire to update  \*/* **async** update(questionnaire) {  **this**.checkIfPropertiesAreSet();  **const** request = **new *Request***(**this**.**\_serverUrl** + **'/'** + questionnaire.id, {  **method**: **'PUT'**,  **headers**: **new *Headers***({  **'Content-Type'**: **'application/json'** }),  **body**: ***JSON***.stringify(questionnaire)  });  **const** response = **await** *fetch*(request);  **return** response.ok  } }  *// Singleton pattern in ES6* **export default new** QuestionnairesRestService(); |

# QuestionnaireContainer.js

|  |
| --- |
| **class** QuestionnaireContainer **extends** React.Component {   constructor(props) {  **super**(props);  **this**.**state** = {  **questionnaires**: [],  **error**: **''** };  **this**.create = **this**.create.bind(**this**);  **this**.update = **this**.update.bind(**this**);  **this**.delete = **this**.delete.bind(**this**);  }   **async** componentDidMount() {  **try** {  QuestionnairesRestService.serverUrl = **this**.**props**.**serverUrl**;  **let** questionnaires = **await** QuestionnairesRestService.getAll();  **this**.setState({  **questionnaires**: questionnaires  })  } **catch** (exception) {  **this**.setState({  **error**: **'Unable to load the questionnaires: '** + exception.**message** })  }  }   **async** create(questionnaire) {  **try** {  **let** questionnaires = **this**.**state**.**questionnaires**;  **let** newquestionnaire = **await** QuestionnairesRestService.save(questionnaire);  questionnaires.push(newquestionnaire);  **this**.setState({  **questionnaires**: questionnaires  })   } **catch** (exception) {  **this**.setState({  **error**: **'Unable to create the questionnaire: '** + exception.**message** })  }  }   **async** update(questionnaire) {  **try** {  **let** questionnaires = **this**.**state**.**questionnaires**;  **let** tmpquestionnaire = questionnaires.find(item => item.**id** === questionnaire.**id**);  tmpquestionnaire.**title** = questionnaire.**title**;  tmpquestionnaire.**description** = questionnaire.**description**;  **await** QuestionnairesRestService.update(tmpquestionnaire);  **this**.setState({  **questionnaires**: questionnaires  })  } **catch** (exception) {  **this**.setState({  **error**: **'Unable to update the questionnaire: '** + exception.**message** })  }  }   **async delete**(id) {  **try** {  **let** questionnaires = **this**.**state**.**questionnaires**;  **await** QuestionnairesRestService.delete(id);  questionnaires = questionnaires.filter(questionnaire => questionnaire.**id** !== id);  **this**.setState({  **questionnaires**: questionnaires  })  } **catch** (exception) {  **this**.setState({  **error**: **'Unable to delete the questionnaire: '** + exception.**message** })  }  }   render() {  **let** numberOfItems = **this**.**state**.**questionnaires**.length + **' Questionnaires found'**;  **return** (  <**div**>  {**this**.**state**.**error** !== **''** && <**Message message=**{**this**.**state**.**error**}/>}  <**Row**>  <**Col**>  <**h2**>Questionnaires</**h2**>  </**Col**>  <**Col className="text-right"**>  <**QuestionnaireCreateDialog  create=**{**this**.create}  />  </**Col**>  </**Row**>  <**Row**>  <**QuestionnaireTable  questionnaires=**{**this**.**state**.**questionnaires**}  **update=**{**this**.update}  **delete=**{**this**.delete}  />  </**Row**>  <**QuestionnaireFooter  leftMessage="2018 FHNW Team"  rightMessage=**{numberOfItems}  />  </**div**>  )  } }  **export default** QuestionnaireContainer; |

# QuestionnaireTable.js

|  |
| --- |
| **class** QuestionnaireTable **extends** React.Component {  render() {  **return** <**table className="table"**>  <**thead**>  <**tr**>  <**th**>ID</**th**>  <**th**>Title</**th**>  <**th**>Description</**th**>  <**th**>Actions</**th**>  </**tr**>  </**thead**>  <**tbody**>  {**this**.**props**.**questionnaires**.map(row => {  **return** <**QuestionnaireTableElement  key=**{row.**id**}  **questionnaire=**{row}  **update=**{**this**.**props**.**update**}  **delete=**{**this**.**props**.**delete**}  />  }  )}  </**tbody**>  </**table**>  } }  **export default** QuestionnaireTable; |

# QuestionnaireTableElement.js

|  |
| --- |
| **class** QuestionnaireTableElement **extends** React.Component {  constructor(props) {  **super**(props)  **this**.delete = **this**.delete.bind(**this**);  }   **delete**() {  **this**.**props**.delete(**this**.**props**.questionnaire.**id**);  }   render() {  **return** <**tr**>  <**td**>{**this**.**props**.questionnaire.**id**}</**td**>  <**td**>{**this**.**props**.questionnaire.**title**}</**td**>  <**td**>{**this**.**props**.questionnaire.**description**}</**td**>  <**td**>  <**Row**>  <**QuestionnaireShowDialog  questionnaire=**{**this**.**props**.questionnaire}  />  <**QuestionnaireUpdateDialog  questionnaire=**{**this**.**props**.questionnaire}  **update=**{**this**.**props**.**update**}  />  <**Button color="danger" onClick=**{**this**.delete}>Delete</**Button**>  </**Row**>  </**td**>  </**tr**>  } } |

# QuestionnaireShowDialog.sh

|  |
| --- |
| **class** QuestionnaireShowDialog **extends** React.Component {  constructor(props) {  **super**(props);  **this**.**state** = {  **showModal**: **false** };  **this**.open = **this**.open.bind(**this**);  **this**.close = **this**.close.bind(**this**);  }   open() {  **this**.setState({  **showModal**: **true** })  }   close() {  **this**.setState({  **showModal**: **false** })  }   render() {  **return** (  <**div**>  <**Button color="secondary" onClick=**{**this**.open}>Show</**Button**>  <**Modal isOpen=**{**this**.**state**.**showModal**} **toggle=**{**this**.close} **size="lg"  autoFocus=**{**false**}>  <**ModalHeader toggle=**{**this**.close}>  Show Questionnaire  </**ModalHeader**>  <**ModalBody**>  <**Form**>  <**FormGroup row**>  <**Label md=**{2} **for="formTitle"**>Title</**Label**>  <**Col md=**{10}>  <**Input type="text" id="formTitle" plaintext**>{**this**.**props**.questionnaire.**title**}</**Input**>  </**Col**>  </**FormGroup**>  <**FormGroup row**>  <**Label md=**{2} **for="formDescription"**> Description </**Label**>  <**Col md=**{10}>  <**Input type="text" id="formDescription"  plaintext**>{**this**.**props**.questionnaire.**description**} </**Input**>  </**Col**>  </**FormGroup**>  <**FormGroup**>  <**Col className="clearfix" style=**{{**padding**: **'.2rem'**}}>  <**Button className="float-right" color="secondary"  onClick=**{**this**.close}>Close</**Button**>  </**Col**>  </**FormGroup**>  </**Form**>  </**ModalBody**>  </**Modal**>  </**div**>  )  } }  **export default** QuestionnaireShowDialog; |

# QuestionnaireCreateDialog.js

|  |
| --- |
| **class** QuestionnaireCreateDialog **extends** React.Component {  constructor(props) {  **super**(props);  **this**.**state** = {  **showModal**: **false**,  **title**: **''**,  **description**: **''** };  **this**.open = **this**.open.bind(**this**);  **this**.close = **this**.close.bind(**this**);  **this**.handle = **this**.handle.bind(**this**);  **this**.create = **this**.create.bind(**this**);  }   open() {  **this**.setState({  **showModal**: **true** })  }   close() {  **this**.setState({  **title**: **''**,  **description**: **''**,  **showModal**: **false** })  }   handle(event) {  **let** value = event.**target**.**value**;  **if** (event.**target**.**id** === **'formTitle'**) {  **this**.setState({**title**: value});  } **else if** (event.**target**.**id** === **'formDescription'**) {  **this**.setState({**description**: value});  }  }   create() {  **this**.**props**.create({**title**: **this**.**state**.**title**, **description**: **this**.**state**.**description**});  **this**.close();  }   render() {  **return** (  <**div**>  <**Button color="success" onClick=**{**this**.open}>Add Questionnaire</**Button**>  <**Modal isOpen=**{**this**.**state**.**showModal**} **toggle=**{**this**.close} **size="lg"  autoFocus=**{**false**}>  <**ModalHeader toggle=**{**this**.close}>  Create Questionnaire  </**ModalHeader**>  <**ModalBody**>  <**Form**>  <**FormGroup row**>  <**Label md=**{2} **for="formTitle"**>Title</**Label**>  <**Col md=**{10}>  <**Input type="text" id="formTitle" value=**{**this**.**state**.**title**} **onChange=**{**this**.handle}/>  </**Col**>  </**FormGroup**>  <**FormGroup row**>  <**Label md=**{2} **for="formDescription"**> Description </**Label**>  <**Col md=**{10}>  <**Input type="text" id="formDescription" value=**{**this**.**state**.**description**}  **onChange=**{**this**.handle}/>  </**Col**>  </**FormGroup**>  <**FormGroup**>  <**Col className="clearfix" style=**{{**padding**: **'.2rem'**}}>  <**Button className="float-right" color="secondary"  onClick=**{**this**.create}>Save</**Button**>  </**Col**>  </**FormGroup**>  </**Form**>  </**ModalBody**>  </**Modal**>  </**div**>  )  } }  **export default** QuestionnaireCreateDialog; |

# QuestionnaireUpdateDialog.js

|  |
| --- |
| **class** QuestionnaireUpdateDialog **extends** React.Component {  constructor(props) {  **super**(props);  **this**.**state** = {  **showModal**: **false**,  **title**: **this**.**props**.questionnaire.**title**,  **description**: **this**.**props**.questionnaire.**description** };  **this**.open = **this**.open.bind(**this**);  **this**.close = **this**.close.bind(**this**);  **this**.handle = **this**.handle.bind(**this**);  **this**.update = **this**.update.bind(**this**);  }   open() {  **this**.setState({  **showModal**: **true** })  }   close() {  **this**.setState({  **showModal**: **false** })  }   handle(event) {  **let** value = event.**target**.**value**;  **if** (event.**target**.**id** === **'formTitle'**) {  **this**.setState({**title**: value});  } **else if** (event.**target**.**id** === **'formDescription'**) {  **this**.setState({**description**: value});  }  }   update() {  **let** questionnaire = **this**.**props**.questionnaire;  questionnaire.**title** = **this**.**state**.**title**;  questionnaire.**description** = **this**.**state**.**description**;  **this**.**props**.update(questionnaire);  **this**.close();  }   render() {  **return** (  <**div**>  <**Button color="primary" onClick=**{**this**.open}>Update</**Button**>  <**Modal isOpen=**{**this**.**state**.**showModal**} **toggle=**{**this**.close} **size="lg"  autoFocus=**{**false**}>  <**ModalHeader toggle=**{**this**.close}>  Update Questionnaire  </**ModalHeader**>  <**ModalBody**>  <**Form**>  <**FormGroup row**>  <**Label md=**{2} **for="formTitle"**>Title</**Label**>  <**Col md=**{10}>  <**Input type="text" id="formTitle" value=**{**this**.**state**.**title**} **onChange=**{**this**.handle}/>  </**Col**>  </**FormGroup**>  <**FormGroup row**>  <**Label md=**{2} **for="formDescription"**> Description </**Label**>  <**Col md=**{10}>  <**Input type="text" id="formDescription" value=**{**this**.**state**.**description**}  **onChange=**{**this**.handle}/>  </**Col**>  </**FormGroup**>  <**FormGroup**>  <**Col className="clearfix" style=**{{**padding**: **'.2rem'**}}>  <**Button className="float-right" color="secondary"  onClick=**{**this**.update}>Update</**Button**>  </**Col**>  </**FormGroup**>  </**Form**>  </**ModalBody**>  </**Modal**>  </**div**>  )  } }  **export default** QuestionnaireUpdateDialog; |

# QuestionnaireFooter.js

|  |
| --- |
| **class** QuestionnaireFooter **extends** React.Component {  render() {  **return** <**Row**>  <**Col**>{**this**.**props**.leftMessage}</**Col**>  <**Col className="text-right"**>{**this**.**props**.rightMessage}</**Col**>  </**Row**>  } } |

Extended React Redux App

# Index.js

|  |
| --- |
| **import** React **from 'react'**; **import** ReactDOM **from 'react-dom'**; **import** App **from './app/App'**; **import** *registerServiceWorker* **from './registerServiceWorker'**; **import** { createStore, combineReducers, applyMiddleware, compose } **from 'redux'**; **import** { Provider } **from 'react-redux'**; **import** thunk **from 'redux-thunk'**; **import** { *questionnaires*, *isLoading* } **from './questionnaire/Reducer'**; **import 'bootstrap/dist/css/bootstrap.min.css'**;  **const** initialState = {}  **const** flashcardApp = combineReducers({  *questionnaires*,  *isLoading* })  **const** composeEnhancers = ***window***.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_ || compose; **const** store = createStore(flashcardApp, initialState, composeEnhancers(  applyMiddleware(thunk) ));  ReactDOM.render(  <**Provider store=**{store}>  <**App**/>  </**Provider**>,   ***document***.getElementById(**'root'**)); *registerServiceWorker*(); |

# App.js

|  |
| --- |
| **import** React **from 'react'**; **import** Header **from './Header'**; **import** Footer **from './Footer'**; **import** { Container } **from 'reactstrap'**; *//import QuestionnaireContainer from '../questionnaire/QuestionnaireContainer';* **import** { QuestionnaireList } **from '../questionnaire/QuestionnaireContainer'**; **import** Loader **from '../misc/Loader'**; **import** Message **from '../misc/Message'**; **import** QuestionnairesRestService **from '../integration/QuestionnairesRestService'**;  **export default class** App **extends** React.Component {  constructor() {  **super**();  *// 'loading' is used locally only to this component to signal loading of config file* **this**.**state** = {  **loading**: **true**,  **error**: **''** }  }    checkUrl(url) {  **if** (url.slice(-1) === **'/'**) {  url = url.slice(0, -1);  }  **return** url;  }   **async** componentDidMount() {  **try** {  **const** response = **await** *fetch*(**'application.json'**);  **if** (!response.ok) {  **throw *Error***(response.statusText);  }  **const** file = **await** response.json();  ***console***.log(file.url);  **const** url = **this**.checkUrl(file.url);  QuestionnairesRestService.serverUrl = url + **"/questionnaires"**;  **this**.setState(  {  **loading**: **false**,  **subTitle**: file.subtitle  }  )  } **catch** (ex) {  ***console***.log(ex);  **this**.setState(  {  **loading**: **false**,  **error**: **'Error while trying to load config file'** }  )  }  }   render() {  **var** comp;  **if** (**this**.**state**.**error** === **''**) {  comp = <**Loader**/>;  } **else** {  comp = <**Message message=**{**this**.**state**.**error**}/>;  }  **return** (  <**Container fluid**>  <**Header  title="RIA-Client with React-Redux"  subtitle=**{**this**.**state**.**subTitle**} />    {(! **this**.**state**.**loading**) ? (  <**QuestionnaireList** />  ) : (  comp  )}    <**Footer message="The FHNW Team"**/>  </**Container**>  )  } } |

# Action.js

|  |
| --- |
| **import** QuestionnairesRestService **from '../integration/QuestionnairesRestService'**;  **export function** *createQuestionnaire*(questionnaire) {  **return** {  **type**: **'CREATE\_QUESTIONNAIRE'**,  **questionnaire**: questionnaire  }; }  **export function** *updateQuestionnaire*(questionnaire) {  **return** {  **type**: **'UPDATE\_QUESTIONNAIRE'**,  **questionnaire**: questionnaire  }; }  **export function** *deleteQuestionnaire*(id) {  **return** {  **type**: **'DELETE\_QUESTIONNAIRE'**,  **id**: id  }; }  **export function** *isLoading*(bool) {  **return** {  **type**: **'IS\_LOADING'**,  **isLoading**: bool  }; }  **export function** *questionnairesFetchSuccess*(questionnaires) {  **return** {  **type**: **'QUESTIONNAIRES\_FETCH\_SUCCESS'**,  questionnaires  }; }  **export function** *fetchQuestionnairesFromServer*() {  **return** (dispatch) => {  *// Signal start loading* dispatch(*isLoading*(**true**));  QuestionnairesRestService.getAll().then((questionnaires) => {  *// Update redux store with loaded questionnaires* dispatch(*questionnairesFetchSuccess*(questionnaires))  })  *// Signal loading finished* dispatch(*isLoading*(**false**));  } }  **export function** *createQuestionnaireOnServer*(questionnaire) {  **return** (dispatch) => {  *// Signal start loading* dispatch(*isLoading*(**true**));   QuestionnairesRestService.save(questionnaire).then(q => {  *// Update redux store with loaded questionnaires* ***console***.log(q);  dispatch(*createQuestionnaire*(q))  ***console***.log(**"Successfully created questionnaire with id="** + q.**id**);   *// Signal loading finished* dispatch(*isLoading*(**false**));  })  } }  **export function** *deleteQuestionnaireOnServer*(id) {  **return** (dispatch) => {  *// Signal start loading* dispatch(*isLoading*(**true**));   QuestionnairesRestService.delete(id).then(ok => {  **if** (ok) {  *// Update redux store* dispatch(*deleteQuestionnaire*(id))  ***console***.log(**"Successfully deleted questionnaire with id="** + id);  } **else** {  ***console***.log(**"Could not delete questionnaire with id="** + id);  }  *// Signal loading finished* dispatch(*isLoading*(**false**));  })  } }  **export function** *updateQuestionnaireOnServer*(questionnaire) {  **return** (dispatch) => {  *// Signal start loading* dispatch(*isLoading*(**true**));   QuestionnairesRestService.update(questionnaire).then(ok => {  **if** (ok) {  *// Signal loading finished* dispatch(*updateQuestionnaire*(questionnaire))  ***console***.log(**"Successfully updated questionnaire with id="** + questionnaire.**id**);  } **else** {  ***console***.log(**"Could not update questionnaire with id="** + questionnaire.**id**);  }  *// Signal loading finished* dispatch(*isLoading*(**false**));  })  } } |

# Reducers.js

|  |
| --- |
| **const** initialState = {  **questionnaires**: [  {**id**: **'0'**, **title**: **'initial title 1'**, **description**: **'initial description 1'**},  {**id**: **'1'**, **title**: **'initial title 2'**, **description**: **'initial description 2'**},  {**id**: **'2'**, **title**: **'initial title 3'**, **description**: **'initial description 3'**}  ] }  **export function** *isLoading*(state = **false**, action) {  **switch** (action.**type**) {  **case 'IS\_LOADING'**:  **return** action.**isLoading**;  **default**:  **return** state;  } }  **export function** *questionnaires*(state=initialState.**questionnaires**, action) {  **switch** (action.**type**) {  **case 'CREATE\_QUESTIONNAIRE'**:  **let** questionnaire = action.**questionnaire**;  *//questionnaire.id = generateId(state).toString();  // add new questionnaire at the tail of old array* **return** [  ...state,  questionnaire  ];  **case 'UPDATE\_QUESTIONNAIRE'**: {  **let** tmp = state.find(q => q.**id** === action.**questionnaire**.**id**);  *// update questionnaire with new values; tmp is a reference into the array element* tmp.**title** = action.**questionnaire**.**title**;  tmp.**description** = action.**questionnaire**.**description**;  **return** state;  }  **case 'DELETE\_QUESTIONNAIRE'**: {  **let** qs = state.slice();  qs = qs.filter(q => q.**id** !== action.**id**);  **return** qs;  */\*  return [  ...state.slice(0, action.id),  ...state.slice(action.id + 1)  ]  \*/* }  **case 'QUESTIONNAIRES\_FETCH\_SUCCESS'**: {  **return** action.**questionnaires**;  }  **default**:  **return** state;  } }  */\* const generateId = questionnaires => {  if (questionnaires.length > 0) {  const nr = parseInt(questionnaires[questionnaires.length - 1].id, 10);  return nr + 1;  }  return 0; } \*/* |

# QuestionnaireContainer.js

|  |
| --- |
| **import** React **from 'react'**; **import** QuestionnaireCreateDialog **from './QuestionnaireCreateDialog'**; **import** QuestionnaireTable **from './QuestionnaireTable'**; **import** Loader **from '../misc/Loader'**; **import** { *createQuestionnaireOnServer*, *updateQuestionnaireOnServer*, *deleteQuestionnaireOnServer*, *fetchQuestionnairesFromServer* } **from './Actions'**; **import** { connect } **from 'react-redux'**;  **class** QuestionnaireContainer **extends** React.Component {  componentDidMount() {  *// Load questionnaires from backend to initialize this component* **this**.props.fetchData();  }   render() {  **return** (  <**div**>  {**this**.props.**loading** ? (  <**Loader** />  ) : (  <**div**>  <**QuestionnaireCreateDialog className="float-right"  createHandler=**{**this**.props.createHandler}  />  <**h3**>Questionnaires</**h3**>  <**QuestionnaireTable  questionnaires=**{**this**.props.**questionnaires**}  **deleteHandler=**{**this**.props.deleteHandler}  **updateHandler=**{**this**.props.updateHandler}  />  </**div**>  )}  </**div**>  )  } }  **const** *mapStateToProps* = (state) => {  **return** {  **questionnaires**: state.**questionnaires**,  l**oading**: state.**isLoading** } }  **const** *mapDispatchToProps* = (dispatch) => {  **return** {  createHandler: (questionnaire) => dispatch(*createQuestionnaireOnServer*(questionnaire)),  updateHandler: (questionnaire) => dispatch(*updateQuestionnaireOnServer*(questionnaire)),  deleteHandler: (id) => dispatch(*deleteQuestionnaireOnServer*(id)),  fetchData: () => dispatch(*fetchQuestionnairesFromServer*())  } }  **export const** QuestionnaireList = connect(  *mapStateToProps*,  *mapDispatchToProps* )(QuestionnaireContainer) |