Công cụ:

JDK 1.8+ (<https://www.oracle.com/java/technologies/javase-jdk8-downloads.html>)

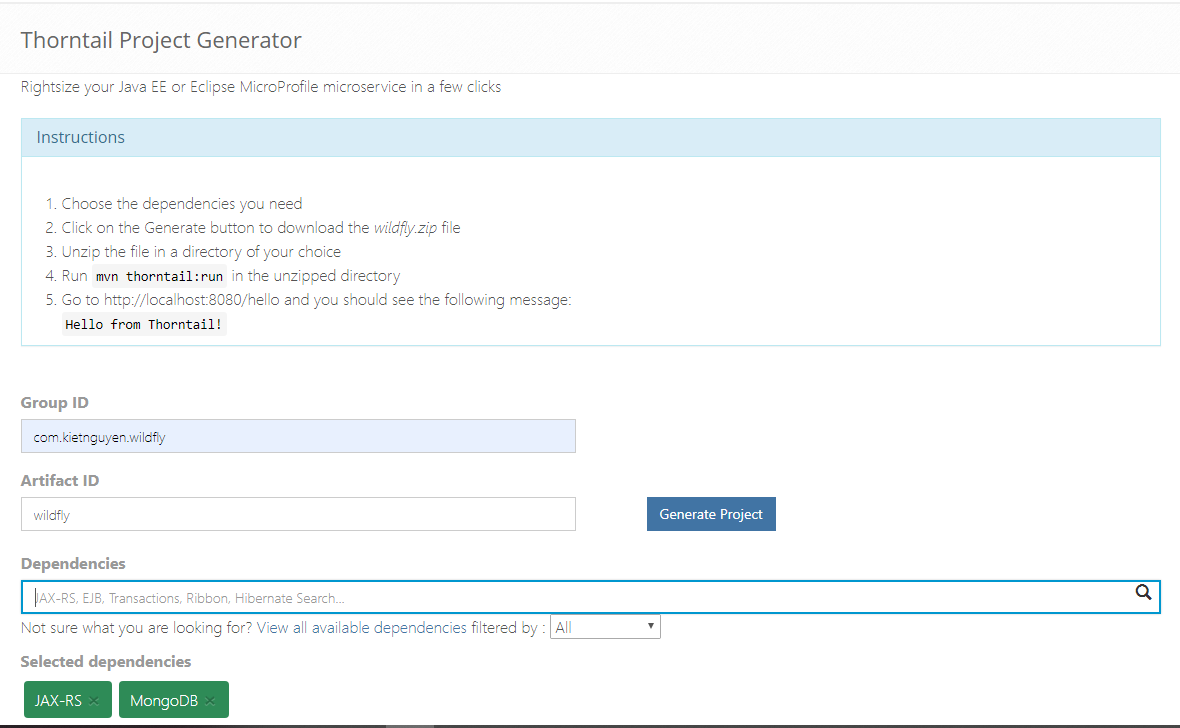
IDE: Eclipse hoặc Netbeans hoặc Intellij (<https://www.eclipse.org/downloads/>)

Visual Studio Code (<https://code.visualstudio.com/download>)

MongoDB 4.x (https://docs.mongodb.com/manual/installation/)

Yarn (<https://classic.yarnpkg.com/en/docs/install>)

Truy cập vào trang web <https://thorntail.io/generator/> để tiến hành tạo dự án wildfly (thorntail là tên sau này của wildfly)



Tiếp theo click chọn “Generate Project” để tiến hành tạo và download project (dưới dạng file .zip) về máy. Sau đó, import project vào IDE đã download và cài đặt ở bước trên.

Cấu hình project

Sau khi import ta mở file pom.xml sẽ thấy những thư viện được tự động thêm vào như sau:

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.kietnguyen</groupId>  <artifactId>wildfly</artifactId>  <version>1.0.0-SNAPSHOT</version>  <packaging>war</packaging>  <name>Thorntail Example</name>  <properties>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <failOnMissingWebXml>false</failOnMissingWebXml>  <version.thorntail>2.6.0.Final</version.thorntail>  </properties>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>io.thorntail</groupId>  <artifactId>bom-all</artifactId>  <version>${version.thorntail}</version>  <scope>import</scope>  <type>pom</type>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <finalName>${project.artifactId}</finalName>  <plugins>  <plugin>  <groupId>io.thorntail</groupId>  <artifactId>thorntail-maven-plugin</artifactId>  <version>${version.thorntail}</version>  <executions>  <execution>  <goals>  <goal>package</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build>  <dependencies>  <dependency>  <groupId>org.mongodb</groupId>  <artifactId>mongo-java-driver</artifactId>  <version>3.4.1</version>  </dependency>  <dependency>  <groupId>io.thorntail</groupId>  <artifactId>jaxrs</artifactId>  </dependency>  </dependencies>  </project> |

**Xây dựng RESTful web services với wildfly:**

Trước tiên, trong folder src/main/java/com/kietnguyen/wildfly/config ta cần tạo một lớp DatabaseManager để kết nối tới MongoDB

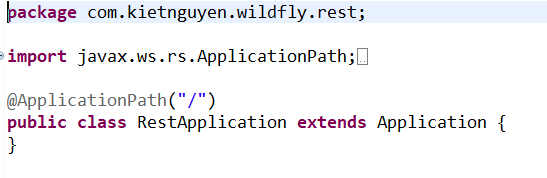
|  |
| --- |
| **package** com.kietnguyen.wildfly.config;  **import** com.mongodb.MongoClient;  **import** com.mongodb.client.MongoDatabase;  **public** **class** DatabaseManager {  **private** **static** DatabaseManager *instance*;  **public** **static** DatabaseManager getInstance() {  **if** (*instance* == **null**) {  *instance* = **new** DatabaseManager();  }  **return** *instance*;  }    **public** MongoDatabase getDB() {  MongoDatabase db;    **try** {  MongoClient mongo = **new** MongoClient("localhost", 27017);    db = mongo.getDatabase("todo-app");  } **catch**(Exception e) {  **return** **null**;  }    **return** db;  }  } |

getInstance(): phương thức này trả về instance của lớp DatabaseManager dưới dạng singleton pattern.

getDB(): phương thức này có nhiệm vụ kết nối với MongoDB và trả về đối tượng đã được kết nối. Thông qua đối tượng này ta có thể truy xuất đến các collections và các thao tác (đọc, ghi) dữ liệu trên collections.

Cũng trong folder config này ta tạo class CORSFilter.java. Class này cho phép những domain khác domain hiện tại có thể truy cập được API.

|  |
| --- |
| **package** com.kietnguyen.wildfly.config;  **import** java.io.IOException;  **import** javax.ws.rs.container.ContainerRequestContext;  **import** javax.ws.rs.container.ContainerResponseContext;  **import** javax.ws.rs.container.ContainerResponseFilter;  **import** javax.ws.rs.ext.Provider;      @Provider  **public** **class** CORSFilter **implements** ContainerResponseFilter {    @Override  **public** **void** filter(ContainerRequestContext requestContext, ContainerResponseContext responseContext) **throws** IOException {  responseContext.getHeaders().add("Access-Control-Allow-Origin", "\*");  responseContext.getHeaders().add("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS");  responseContext.getHeaders().add("Access-Control-Max-Age", "-1");  responseContext.getHeaders().add("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");  }    } |

Lúc này, mặc định wildfly sẽ tạo cho ta folder src/main/java/com/kietnguyen/wildfly/rest chứa RestApplication.java

Annotation @ApplicationPath là entry point để định nghĩa api.

Tiếp theo, trong folder src/main/java/com/kietnguyen/wildfly/controller ta tạo file TodoController.java để nhận request từ phía client

|  |
| --- |
| **package** com.kietnguyen.wildfly.controller;  **import** javax.ws.rs.Path;  **import** javax.ws.rs.PathParam;  **import** org.bson.Document;  **import** com.kietnguyen.wildfly.dao.TodoDAO;  **import** com.kietnguyen.wildfly.model.Todo;  **import** javax.ws.rs.core.\*;  **import** javax.ws.rs.Consumes;  **import** javax.ws.rs.DELETE;  **import** javax.ws.rs.FormParam;  **import** javax.ws.rs.GET;  **import** javax.ws.rs.POST;  **import** javax.ws.rs.PUT;  **import** javax.ws.rs.Produces;  @Path("/todos")  **public** **class** TodoController {  TodoDAO todoDAO = **new** TodoDAO();  @GET  @Produces(MediaType.***APPLICATION\_JSON***)  **public** Response getAllTodos() {  **return** Response.*ok*(todoDAO.getAll()).build();  }  @GET  @Path("{id}")  @Produces(MediaType.***APPLICATION\_JSON***)  **public** Response getById(@PathParam("id") String id) {  Todo todo = todoDAO.findById(id);    **if** (todo == **null**) **return** Response.*status*(404).build();    **return** Response.*ok*(todo).build();  }  @POST  @Consumes("application/x-www-form-urlencoded")  @Produces(MediaType.***APPLICATION\_JSON***)  **public** **void** create(@FormParam("task") String task, @FormParam("isCompleted") **boolean** isCompleted) {  todoDAO.create(task, isCompleted);  }  @DELETE  @Path("{id}")  **public** Response delete(@PathParam("id") String id) {  Todo todo = todoDAO.findById(id);  **if** (todo == **null**)  **return** Response.*status*(404).build();  todoDAO.delete(todo.getId());  **return** Response.*ok*("Deleted success").build();  }    @PUT  @Path("{id}")  @Consumes("application/x-www-form-urlencoded")  @Produces(MediaType.***APPLICATION\_JSON***)  **public** Response update(@PathParam("id") String id, @FormParam("isCompleted") **boolean** isCompleted) {  Todo todo = todoDAO.findById(id);    **if** (todo == **null**) **return** Response.*status*(404).build();  todoDAO.update(id, isCompleted);    **return** Response.*ok*().build();  }  } |

@Path: annotation xác định URI ứng với resource tương ứng

@GET: annotation xử lí http get request

@POST: annotation xử lí http post request

@PUT: annotation xử lí http put request

@DELETE: annotation xử lí http delete request

@Consumes: annotation xác định chỉ nhận những request có dữ liệu dưới dạng cụ thể (XML, JSON, x-www-form-urlencoded,..)

@Produces: annotation chỉ định response trả về dữ liệu dưới dạng cụ thể (XML, JSON,...)

@PathParam: annotation xác định dữ liệu được truyền thông qua URI

@FormParam: annotation xác định dữ liệu được truyền dưới dạng form data

getAllTodos(): lấy danh sách công việc trả về client

getById(): lấy ra công việc theo id

create(): tạo công việc

update(): sửa công việc theo id

delete(): xóa công việc theo id

**Business Logic**

Trong folder src/main/java/com/kietnguyen/wildfly/dao tạo file TodoDAO.java là class trực tiếp tương tác với database.

|  |
| --- |
| **package** com.kietnguyen.wildfly.dao;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.bson.Document;  **import** org.bson.types.ObjectId;  **import** com.kietnguyen.wildfly.config.DatabaseManager;  **import** com.kietnguyen.wildfly.model.Todo;  **import** com.mongodb.BasicDBObject;  **import** com.mongodb.client.FindIterable;  **import** com.mongodb.client.MongoCollection;  **import** com.mongodb.client.MongoCursor;  **import** com.mongodb.client.MongoDatabase;  **public** **class** TodoDAO {  MongoCollection<Document> collection;    **public** TodoDAO() {  MongoDatabase db = DatabaseManager.*getInstance*().getDB();  **this**.collection = db.getCollection("todos");  }    **public** **void** create(String task, **boolean** isCompleted) {  Document document = **new** Document("task", task).append("isCompleted", isCompleted);    collection.insertOne(document);  }    **public** **void** delete(String id) {  BasicDBObject query = **new** BasicDBObject();    query.put("\_id", **new** ObjectId(id));    collection.deleteOne(query);  }    **public** List<Todo> getAll() {  FindIterable<Document> document = collection.find();  MongoCursor<Document> iterator = document.iterator();    List<Todo> todos = **new** ArrayList<Todo>();    **while** (iterator.hasNext()) {  Document doc = iterator.next();    Todo todo = **new** Todo();    todo.setId(doc.getObjectId("\_id").toHexString());  todo.setTask(doc.getString("task"));  todo.setCompleted(doc.getBoolean("isCompleted"));    todos.add(todo);  }    **return** todos;  }    **public** **void** update(String id, **boolean** isCompleted) {  BasicDBObject filter = **new** BasicDBObject();    filter.put("\_id", **new** ObjectId(id));    BasicDBObject body = **new** BasicDBObject();    body.put("isCompleted", isCompleted);    BasicDBObject updateObject = **new** BasicDBObject();    updateObject.put("$set", body);    collection.updateOne(filter, updateObject);  }    **public** Todo findById(String id) {  BasicDBObject query = **new** BasicDBObject();    query.put("\_id", **new** ObjectId(id));    FindIterable<Document> document = collection.find(query);  MongoCursor<Document> iterator = document.iterator();    Todo todo = **new** Todo();    **while** (iterator.hasNext()) {  Document doc = iterator.next();    todo.setId(doc.getObjectId("\_id").toHexString());  todo.setTask(doc.getString("task"));  todo.setCompleted(doc.getBoolean("isCompleted"));  }    **return** todo;  }  } |

Create(): thêm một công việc vào database

Delete(): xóa công việc dựa vào id truyền vào khỏi database

Update(): cập nhật trạng thái công việc theo id truyền vào

FindById(): lấy ra công việc dựa vào id truyền vào

getAll(): lấy ra danh sách công việc trong database

Sau khi hoàn thành các bước trên ta sẽ có cấu trúc folder như sau

├── pom.xml

└── src

├── main

│   ├── java

│   │   └── com

│   │   └── kietnguyen

│   │   └── wildfly

│   │   ├── config

│   │ └── CORSFilter.java

│   │ └── DatabaseManager.java

│   │   ├── dao

│   │   │   └── TodoDAO.java

│   │   ├── model

│   │   │   └── Todo.java

│   │   └── rest

│   │   └── RestApplication.java

Tiếp đến ta cần xây dựng giao diện sử dụng bộ API này.

Sau khi cài đặt yarn ta gõ 2 lệnh để tạo React.js project

$ yarn global add create-react-app

$ create-react-app todo-frontend

Mở project vừa tạo bằng visual studio code. Khi đó, yarn sẽ tạo cho file App.js là nơi ta sẽ tạo giao diện cho ứng dụng. Thêm đoạn code sau vào file App.js

|  |
| --- |
| import React, { Component } from 'react';  import './App.css';  import { connect } from 'react-redux';  import  \* as actions from './actions/todo';  class App extends Component {    constructor(props) {      super(props);      this.state = {        task: ''      }    }      componentWillMount = () => {      this.props.fetchTodoList();    };      handleOnChange = e => {      const target = e.target;      const key = target.name;      const isCompleted = target.checked;      this.props.updateTodo({        id: key,        isCompleted      });    }    handleInputChange = e => {      const target = e.target;      const key = target.name;      const value = target.value;      this.setState({ [key] : value });    }    addTodo = (e) => {      e.preventDefault();      if (this.state.task) {        this.props.addTodo(this.state.task);        this.setState({ task: '' });      } else {        alert('Please add task you need to do !');      }    }    renderTodoItem = () => {      if (this.props.todos) {        return this.props.todos.map(item => {          return this.ToDoItem(item);        })      }    }    onDelete = (e, id) => {      e.preventDefault();      let r = window.confirm('Do you want to delete this to do ?');      if (r) {        this.props.deleteTodo(id);      }    }      ToDoItem = (props) => {      if (props.completed) {        return (          <li key={props.id} className="completed">            <div className="form-check"> <label className="form-check-label"> <input key={props.id} name={props.id} onChange={(e) => this.handleOnChange(e)} className="checkbox" type="checkbox" checked={true} value={props.task}/>{props.task}<i className="input-helper"></i></label> </div> <i onClick={(e) => this.onDelete(e, props.id)} className="remove mdi mdi-close-circle-outline" />          </li>        );      } else {        return (          <li key={props.id}>            <div className="form-check"> <label className="form-check-label"> <input key={props.id} name={props.id} onChange={(e) => this.handleOnChange(e)} className="checkbox" type="checkbox" checked={false} value={props.task}/> {props.task} <i className="input-helper"></i></label> </div> <i onClick={(e) => this.onDelete(e, props.id)} className="remove mdi mdi-close-circle-outline" />          </li>        );      }    }    handleKeyDown = (e) => {      if (e.key === 'Enter') {        this.addTodo(e);      }    }      render() {      return (        <div className="container">          <div className="page-content page-container" id="page-content">            <div className="padding">              <div className="row container d-flex justify-content-center">                <div className="col-lg-12">                  <div className="card px-3">                    <div className="card-body">                      <h1 className="card-title">TO DO LIST</h1>                      <div className="add-items d-flex"> <input name="task" value={this.state.task} onKeyDown={(e) => this.handleKeyDown(e)} onChange={(e) => this.handleInputChange(e)} type="text" className="form-control todo-list-input" placeholder="What do you need to do today?" /> <button onClick={(e) => this.addTodo(e)} className="add btn btn-primary font-weight-bold todo-list-add-btn">Add</button> </div>                      <div className="list-wrapper">                        <ul className="d-flex flex-column-reverse todo-list">                          {this.renderTodoItem()}                        </ul>                      </div>                    </div>                  </div>                </div>              </div>            </div>          </div>        </div>      );    }  }  const mapStateToProps = (state, ownProps) => {    return {      todos: state.todos    }  }  const mapDispatchToProps = (dispatch, ownProps) => {    return {      fetchTodoList: () => dispatch(actions.fetchTodoList()),      updateTodo: (todo) => dispatch(actions.updateTodo(todo)),      addTodo: (task) => dispatch(actions.addTodo(task)),      deleteTodo: (id) => dispatch(actions.deleteTodo(id))    }  }  export default connect(mapStateToProps, mapDispatchToProps)(App) |

Tiếp theo, trong folder src/services ta tạo 2 file BaseService.js và TodoService.js để gọi tới wildfly API.

|  |
| --- |
| import axios from 'axios';  class BaseService {      constructor(props) {          const service = axios.create({              headers: {}          });          service.interceptors.response.use(this.onSuccess, this.onError);          this.service = service;      }      onSuccess(response) {          return response;      }      onError = error => {          return Promise.reject(error.response);      }      get(endpoint) {          return this.service.get(endpoint);      }      post(endpoint, payload) {          return this.service.request({              method: 'POST',              url: endpoint,              responseType: 'json',              data: payload          });      }      put(endpoint, payload) {          return this.service.request({              method: 'PUT',              url: endpoint,              responseType: 'json',              data: payload          });      }      delete(endpoint) {          return this.service.request({              method: 'DELETE',              url: endpoint,              responseType: 'json',          });      }    }  export default BaseService; |

onSuccess(): hàm này được trigger khi api trả về kết quả thành công

onError(): hàm này được trigger khi quá trình gọi api trả về lỗi

get(): hàm này dùng để gọi api bằng giao thức http get

post():hàm này dùng để gọi api bằng giao thức http post

put():hàm này dùng để gọi api bằng giao thức http put

delete():hàm này dùng để gọi api bằng giao thức http delete

|  |
| --- |
| import BaseService from './BaseService';  import \* as AppConstant from '../constants/constants';  import qs from 'querystring';  import axios from 'axios';  class TodoService extends BaseService {      constructor(props) {          super(props);          this.requestURL = AppConstant.API\_URL;      }      getAll() {          return this.get(this.requestURL);      }        create(task) {          return new Promise((resolve, reject) => {              axios({                  method: 'POST',                  url: `${this.requestURL}`,                  data: qs.stringify({                      task,                      isCompleted: false                  }),                  headers: {'Content-Type': 'application/x-www-form-urlencoded' }                  })                  .then(function (res) {                     resolve(res);                  })                  .catch(function (err) {                      reject(err);                  });          });      }      update(id, isCompleted) {          return new Promise((resolve, reject) => {              axios({                  method: 'PUT',                  url: `${this.requestURL}/${id}`,                  data: qs.stringify({                      isCompleted                  }),                  headers: {'Content-Type': 'application/x-www-form-urlencoded' }                  })                  .then(function (res) {                     resolve(res);                  })                  .catch(function (err) {                      reject(err);                  });          });      }      removeTodo(id) {          let endpoint = '';          if (id) {              endpoint = `${this.requestURL}/${id}`;          }          return this.delete(endpoint);      }    }  export default new TodoService(); |

getAll(): hàm này gọi api lấy danh sách công việc

update(): hàm này gọi api để cập nhật trạng thái công việc

create(): hàm này gọi api để tạo công việc

removeTodo(): hàm này dùng để xóa công việc

Kế tiếp trong thư mục src ta tạo folder actions là nơi trực tiếp xử lí kết quả sau khi API trả về kết quả

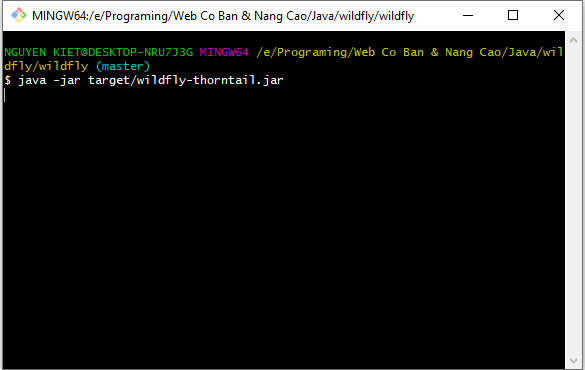
|  |
| --- |
| import { StatusCode,      FETCH\_TODO\_LIST  } from '../constants/constants';  import TodoService from '../services/TodoService';  export const fetchTodoListSuccess = (todos) => {      return {          type: FETCH\_TODO\_LIST,          todos      }  }  export const fetchTodoList = () => {      return dispatch => {          TodoService.getAll()              .then(res => {                  if (res.status === StatusCode.SUCCESS || res.status === StatusCode.NO\_CONTENT) {                      dispatch(fetchTodoListSuccess(res.data));                  }              })              .catch(err => {                  console.log(err)              })      }  }  export const updateTodo = (todo) => {      const { id, isCompleted } = todo;      return dispatch => {          TodoService.update(id, isCompleted)                      .then(res => {                          if (res.status === StatusCode.SUCCESS || res.status === StatusCode.NO\_CONTENT) {                              return TodoService.getAll();                          }                      })                      .then(res2 => {                          if (res2.status === StatusCode.SUCCESS || res2.status === StatusCode.NO\_CONTENT) {                              dispatch(fetchTodoListSuccess(res2.data));                          }                      })                      .catch(err => {                          console.log(err)                      })      }  }  export const addTodo = task => {      return dispatch => {          TodoService.create(task)                      .then(res => {                          if (res.status === StatusCode.SUCCESS || res.status === StatusCode.NO\_CONTENT) {                              dispatch(fetchTodoList());                          }                      })                      .catch(err => {                          console.log(err);                      })      }  }  export const deleteTodo = id => {      return dispatch => {          TodoService.removeTodo(id)                      .then(res => {                          if (res.status === StatusCode.SUCCESS || res.status === StatusCode.NO\_CONTENT) {                              dispatch(fetchTodoList());                          }                      })                      .catch(err => {                          console.log(err);                      })      }  } |

Chạy thử 2 project

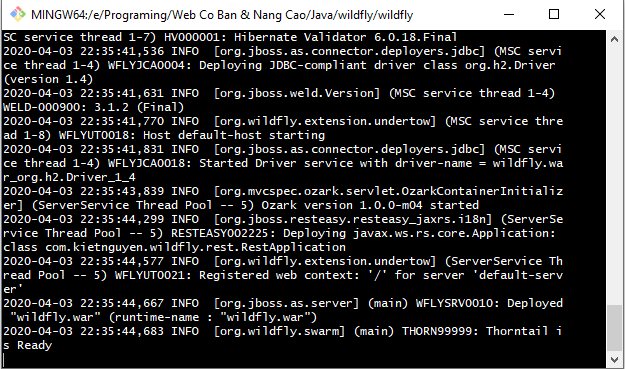
- Bước tiếp theo ta cần thực hiện là tiến hành chạy thử project.

- Mở wildfly project bằng eclipse. Click chuột phải vào project -> Run as -> Maven Install. Lúc này eclipse sẽ gom hết source code ta đã tạo và nén thành 1 file wildfly-thorn.jar trong thư mục target.

- Lúc này ta tiến hành chạy file wildfly-thorn.jar bằng câu lệnh:

$ java –jar target/wildfly-thorn.jar

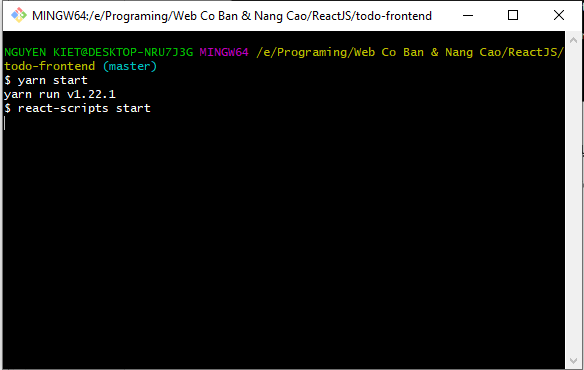
Sau khi chạy thành công, ta được kết quả như sau:

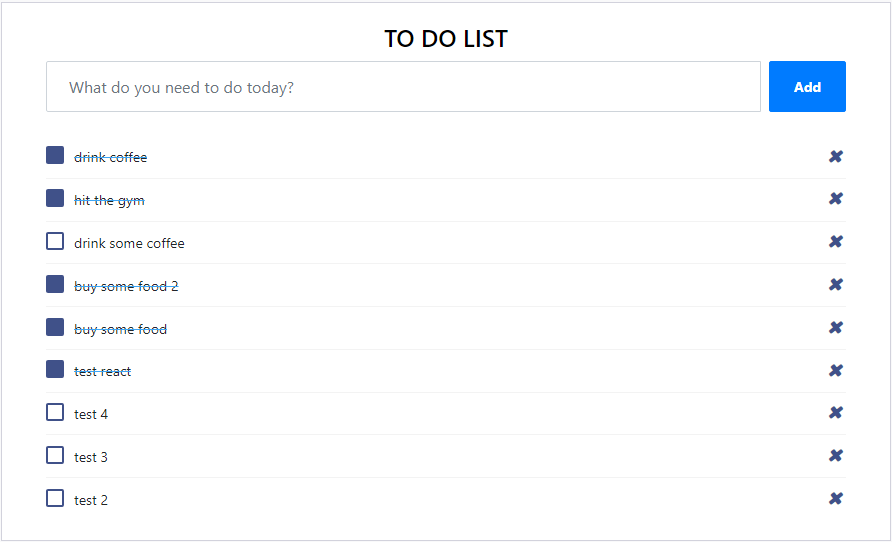


Tiếp theo, ta cần chạy phần frontend đã xây dựng bằng React.js

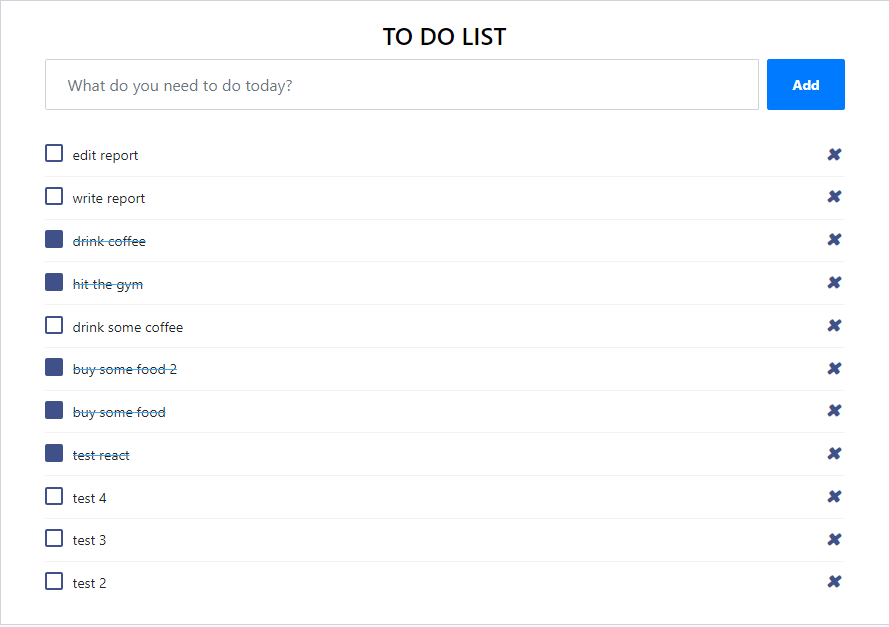
- Mở cửa sổ cmd tại thư mục chứa file package.json. Gõ lệnh sau:

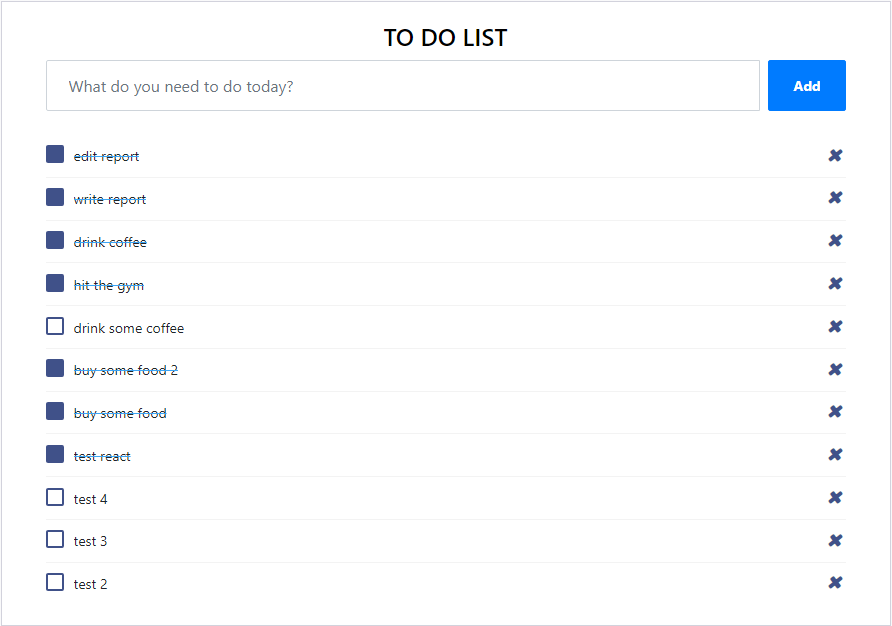
$ yarn start



Truy cập vào đường dẫn <http://localhost:3000> sẽ thấy giao diện như sau

Tiếp theo, ta tiến hành thêm thử 2 công việc mới có tên là “write report” và “edit report” sau đó bấm “Add” ta sẽ thấy kết quả như sau:



Giả sử công việc đã được hoàn thành, ta tick chọn vào ô checkbox. Kết quả nhận được:

Lúc này ta thấy 2 công việc “write report” và “edit report” đã được tick xanh và có gạch ngang. Chứng tỏ api updateTodo() đã hoạt động.

Kế tiếp, ta thử xóa “edit report” bằng cách nhấn vào dấu “x”. Lúc này sẽ có 1 dialog hiện tại hỏi rằng ta có muốn xóa công việc này hay không. Nếu ta chọn “OK” ta sẽ được kết quả là công việc “edit report” đã bị xóa tức là api removeTodo() đã hoạt động

