1. **Về Spring Boot**

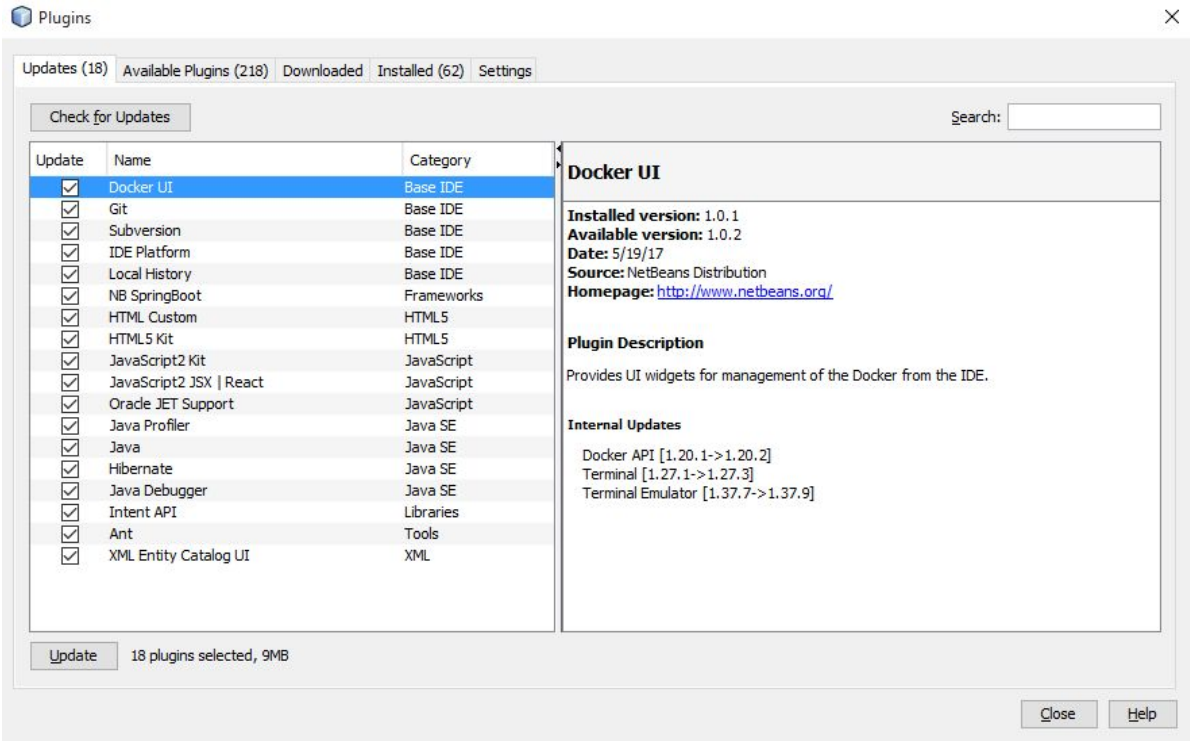
   Để tạo ứng dụng Web MVC với Spring vẫn thực sự rất khó khăn làm nản lòng người muốn theo học

   Spring Boot ra đời với mục đích làm giản tiện các bước cần phải cấu hình tay để giúp người ta có thể nhanh chóng tạo được 1 ứng dụng, nhanh chóng hơn, dễ thở hơn nhiều

   Các phần dưới đây sẽ giúp cho các bạn mới tiếp xúc có thể thực hiện tạo Spring Boot Project với Netbeans một cách thuận lợi từ đó làm bước đệm để phát triển các ứng dụng lớn hơn.

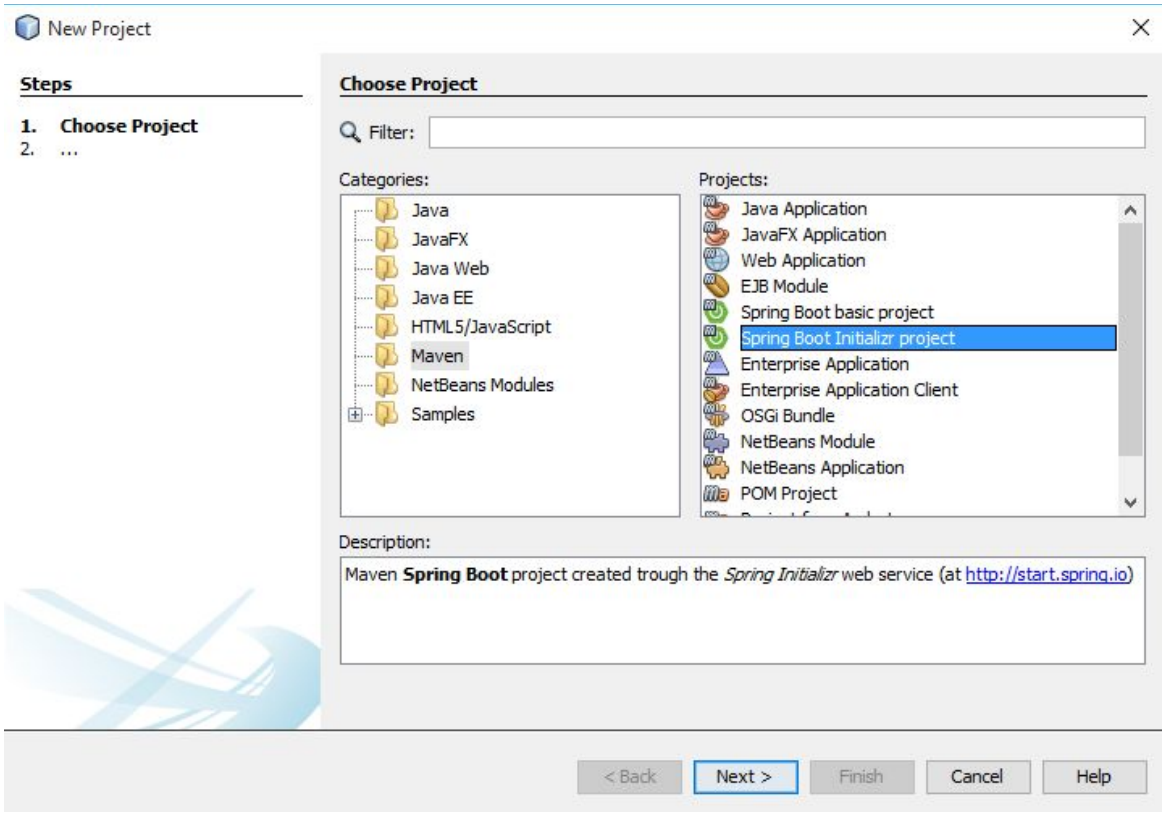
   Spring Boot đã tích hợp các thư viện tomcat bên trong nó, cho nên để chạy ứng dụng web ta không cần phải cài đặt thêm server web nào hết, chỉ cần chạy bằng lệnh java là được

2. **Cài đặt plugin cho Netbeans**
   - Trong Netbeans mở Toos > Plugins



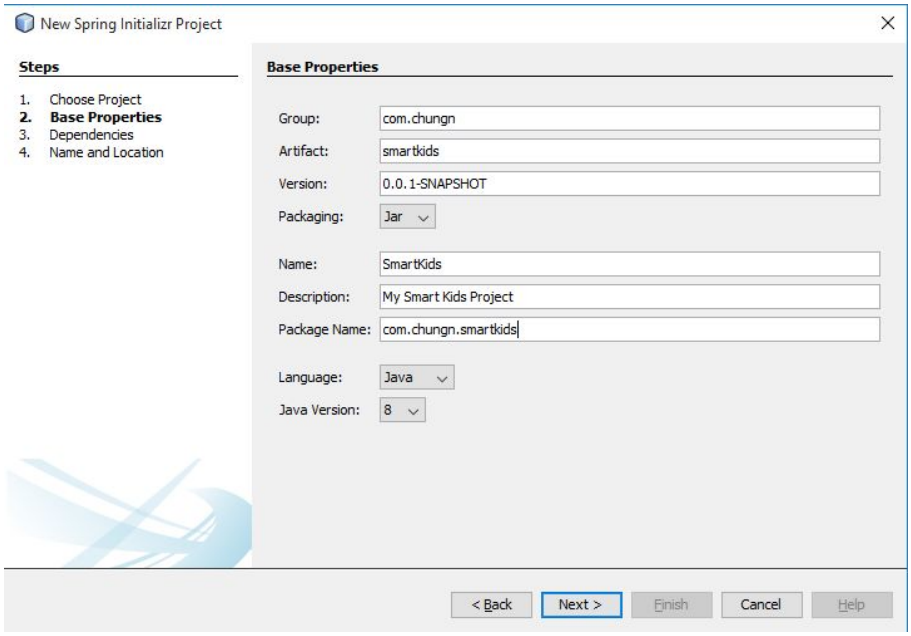

   - Chon tab Avaiable Plugin sau đó search NB Spring Boot, chọn và thực hiện install

3. **Tạo một Spring boot Project sử dụng bộ Initializer**
   - Chọn New Project

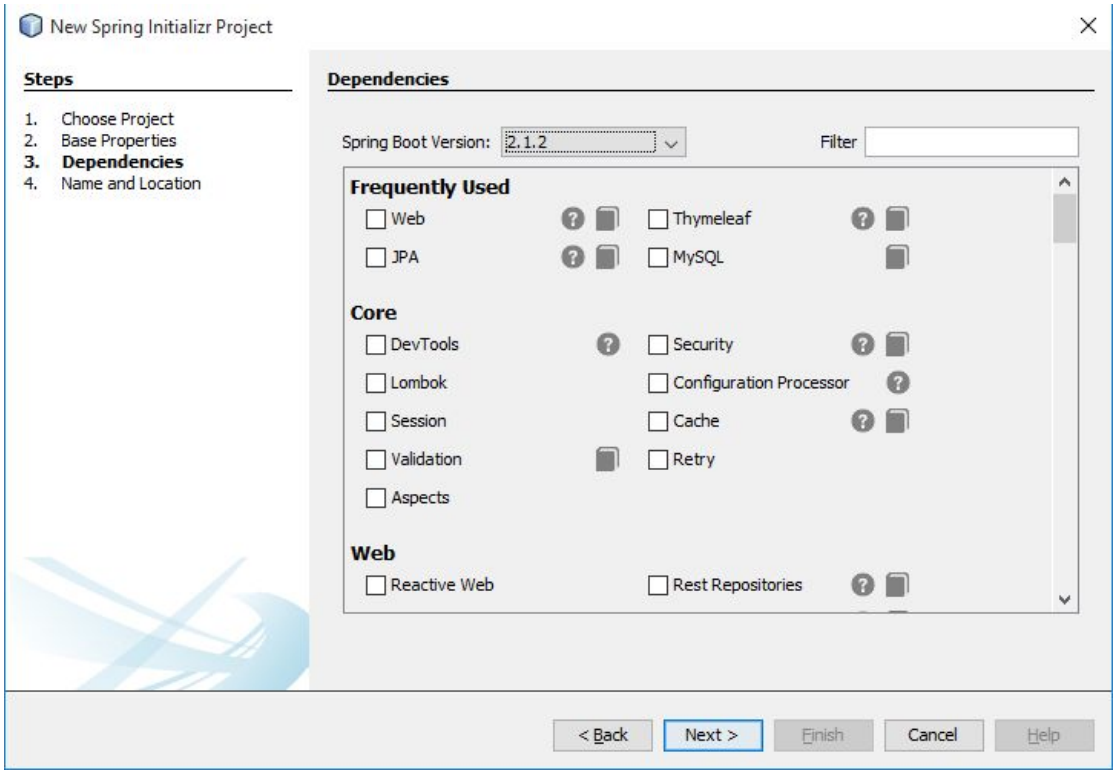



Chọn Maven > Spring Boot Initialzr project rồi nhấn Next

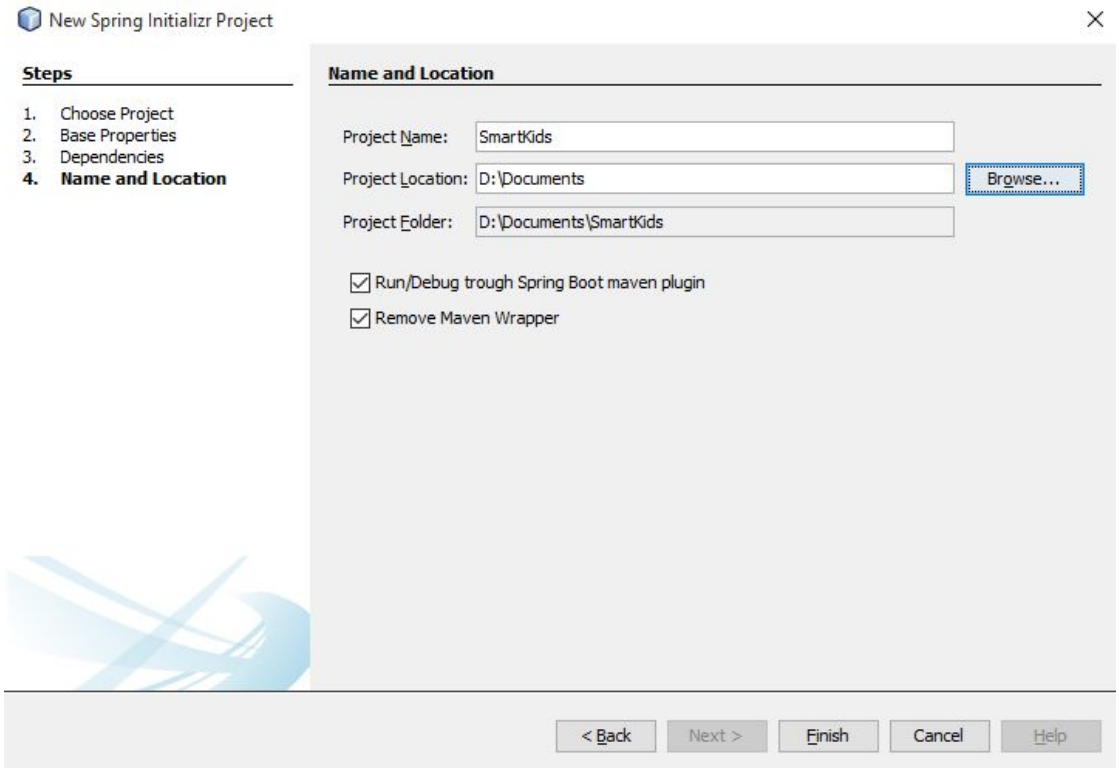
Việc thực hiện khởi tạo project này đòi hỏi phải request lên server springio nên cần đợi 1 lúc để nó thực hiện xong




Điền các thông tin dự án và nhấn Next

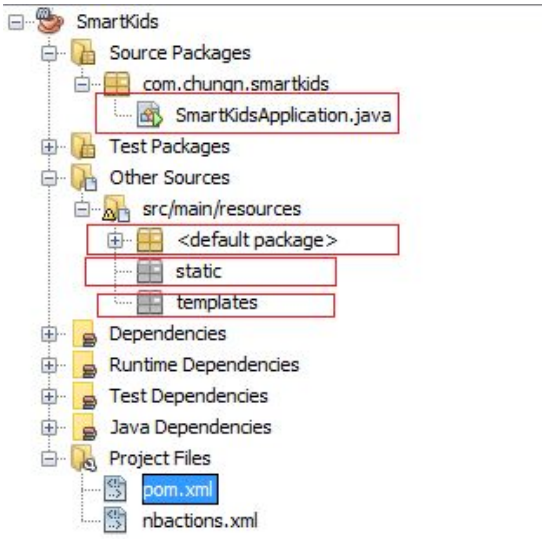



Đây là màn hình chọn các thư viện sẽ được sử dụng trong project. Như bạn có thể thấy, có vô vàn thư viện, tuy nhiên trong phạm vi hướng dẫn này ta sẽ chọn: Web, JPA, Thymeleaf, Mysql. Các bạn có thể tự tìm thêm thông tin về các thư viện này để làm gì :D Nhấn Next để chuyển sang màn hình tiếp theo



Chọn tên và folder, sau đó nhấn Finish
Đợi 1 chút để Netbeans khởi tạo cấu trúc project. Chú ý: tắt bỏ việc transfer với maven central repository, vì nó rất là nặng, extract ra đầy ổ cứng mất.

Cấu trúc của project sẽ được tạo ra như hình sau




File SmartKidsApplication.java sẽ là file chứa function main của ứng dụng, khi chạy sẽ chạy file này đầu tiên.
Các file quan trọng như

- application.properties chứa các cấu hình cũng được tạo sẵn (trong default package)
- Static: folder chứa các file ảnh, css, js
- Template: Chứa các file template thymeleaf

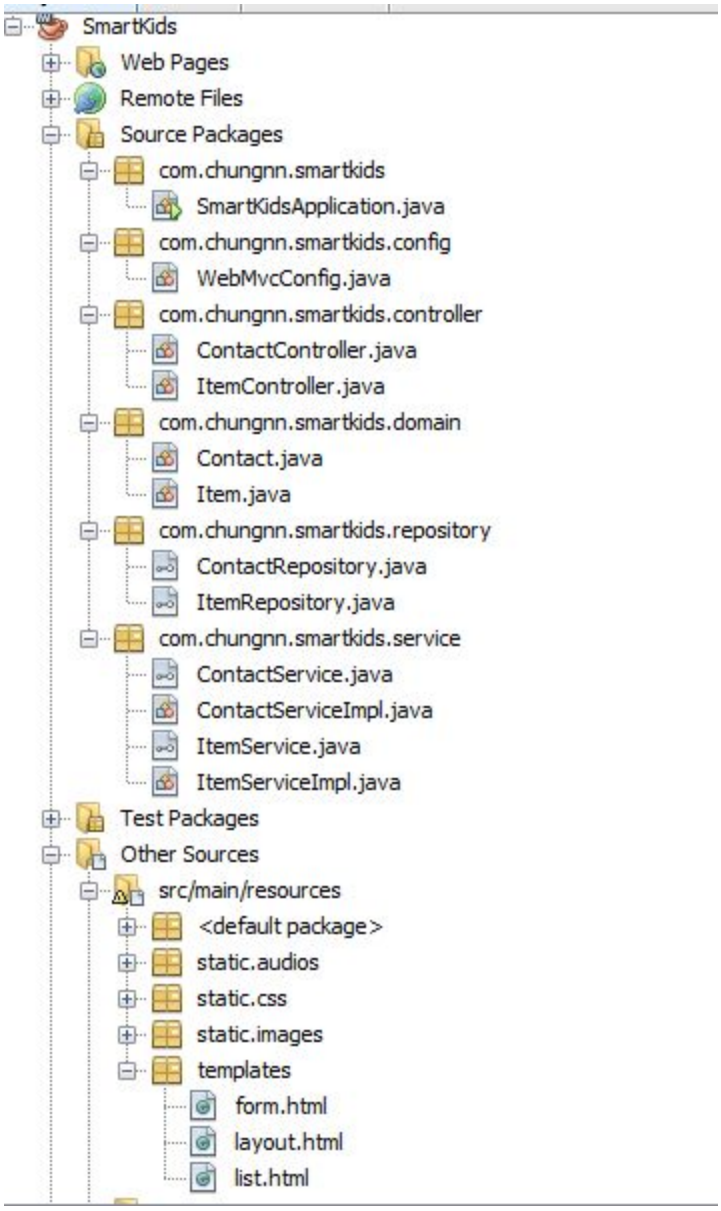
Tiếp theo ta sẽ xây dựng ứng dụng gồm các chức năng sau

- Hiển thị danh sách liên hệ
- Thêm liên hệ mới
- Sửa liên hệ
- Xóa liên hệ

Thực hiện tạo bảng

```
CREATE TABLE `contact` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `name` varchar(50) NOT NULL,
 `email` varchar(50) NULL,
 `phone` varchar(20) NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Tiếp theo ta sẽ xây dựng ứng dụng với cấu trúc thư mục như sau:




**Cấu hình kết nối csdl**

Để thực hiện cấu hình kết nối csdl, ta chỉnh sửa file application.properties

```
# ================================
# THYMELEAF
# ================================
spring.thymeleaf.cache=false

# ================================
```

```
# DATASOURCE
# ==============================

# Set here configurations for the database connection

# Connection url for the database "mycontact"
spring.datasource.url=jdbc:mysql://10.1.7.11:3306/mydb?useSSL=false

# MySQL username and password
spring.datasource.username=root
spring.datasource.password=vietis@12345

# Keep the connection alive if idle for a long time (needed in production)
spring.datasource.dbcp.test-while-idle=true
spring.datasource.dbcp.validation-query=SELECT 1

# ==============================
# JPA / HIBERNATE
# ==============================

# Use spring.jpa.properties.* for Hibernate native properties (the prefix is
# stripped before adding them to the entity manager).

# Show or not log for each sql query
spring.jpa.show-sql=true

# Hibernate ddl auto (create, create-drop, update): with "update" the database
# schema will be automatically updated accordingly to java entities found in
# the project
spring.jpa.hibernate.ddl-auto=update

# Naming strategy
spring.jpa.hibernate.naming.strategy=org.hibernate.cfg.ImprovedNamingStrategy

# Allows Hibernate to generate SQL optimized for a particular DBMS
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

**Tạo file POJO**

Tạo package com.chungnn.smartkids.domain

Sau đó tạo class: Contact như sau

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.chungnn.smartkids.domain;
```

```
import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;

@Entity
@Table(name = "contact")
public class Contact implements Serializable {

    private static final long serialVersionUID = 2L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", nullable = false)
    private int id;

    @NotEmpty
    @Column(name = "name", nullable = false)
    private String name;

    @Email
    @Column(name = "email")
    private String email;

    @Column(name = "phone")
    private String phone;

    public Contact() {
        super();
    }

    public Contact(int id, String name, String email, String phone) {
        super();
        this.id = id;
        this.name = name;
        this.email = email;
        this.phone = phone;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

}
```

**Tạo các class Repository**

Tạo package com.chungnn.smartkids.repository, sau đó tạo interface ContactRepository như sau:

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.chungnn.smartkids.repository;

import com.chungnn.smartkids.domain.Contact;
import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface ContactRepository extends CrudRepository<Contact, Integer> {

    List<Contact> findByNameContaining(String q);
```

```
}
```

**Tạo các class Service**

Tạo package com.chungnn.smartkids.service, rồi tạo interface ContactService như sau

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.chungnn.smartkids.service;

import com.chungnn.smartkids.domain.Contact;
import java.util.List;



public interface ContactService {

    Iterable<Contact> findAll();

    List<Contact> search(String q);

    Contact findOne(int id);

    void save(Contact contact);

    void delete(int id);

}
```

Sau đó tạo class ContactServiceImpl để implement các interface đã định nghĩa

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.chungnn.smartkids.service;

import com.chungnn.smartkids.domain.Contact;
```

```
import com.chungnn.smartkids.repository.ContactRepository;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;


@Service
public class ContactServiceImpl implements ContactService {

    @Autowired
    private ContactRepository contactRepository;

    @Override
    public Iterable<Contact> findAll() {
        return contactRepository.findAll();
    }

    @Override
    public List<Contact> search(String q) {
        return contactRepository.findByNameContaining(q);
    }

    @Override
    public Contact findOne(int id) {
        Optional<Contact> optinalEntity  =  contactRepository.findById(id);
        return optinalEntity.get();
    }

    @Override
    public void save(Contact contact) {
        contactRepository.save(contact);
    }

    @Override
    public void delete(int id) {
        contactRepository.deleteById(id);
    }

}
```

**Tạo các file templates**

Ứng với mỗi trang trong ứng dựng web ta sẽ tạo một file .html ở trong folder templates, tuy nhiên các trang có thể sử dụng chung phần header, footer, nên ta tạo file layout.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:fragment="head">
<meta charset="utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<!-- The above 3 meta tags *must* come first in the head; any other head
content must come *after* these tags -->

<title>Spring MyContact</title>

<link href="../static/images/logo.png" th:href="@{/images/logo.png}"
    rel="shortcut icon" />

<!-- Bootstrap -->
<link

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    th:href="@{/webjars/bootstrap/3.3.7/css/bootstrap.min.css}"
    rel="stylesheet" />
<!-- Custom style -->
<link href="../static/css/style.css" th:href="@{/css/style.css}"
    rel="stylesheet" />

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script
    src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"
    th:src="@{/webjars/jquery/1.12.4/jquery.min.js}"></script>
<!-- Include all compiled plugins (below), or include individual files as
needed -->
<script
    src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
    th:src="@{/webjars/bootstrap/3.3.7/js/bootstrap.min.js}"></script>

<!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media
queries -->
<!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
<!--[if lt IE 9]>
      <script
src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
      <script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
</head>
<body>
    <nav th:fragment="header"
        class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed"
                    data-toggle="collapse" data-target="#navbar"
aria-expanded="false"
                    aria-controls="navbar">
```

```
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="#">Spring MyContact</a>
            </div>
        </div>
    </nav>

    <h1>Main content</h1>

    <footer th:fragment="footer" class="container">
        ALoha
    </footer>
</body>
</html>
```

Bạn là người mới, bạn cần dành một chút thời gian để quan sát file này, các thẻ html sẽ được ghép với các thẻ của thymeleaf.
Tại đây chúng ta có sử dụng đến thư viện jquery, bootstrap trong webjars, nên phải khai báo thêm dependencies trong file pom.xml

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>3.3.7</version>
</dependency>
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>jquery</artifactId>
    <version>1.12.4</version>
</dependency>
```

**Trang danh sách**
Tạo class ContactController trong package com.chungnn.smartkids.controller

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.chungnn.smartkids.controller;
```

```
import com.chungnn.smartkids.domain.Contact;
import com.chungnn.smartkids.service.ContactService;
import javax.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller
public class ContactController {

    @Autowired
    private ContactService contactService;

    @GetMapping("/contact")
    public String index(Model model) {
        model.addAttribute("contacts", contactService.findAll());
        return "list";
    }

}
```

Sau đó tạo file view: list.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head th:replace="layout::head"></head>
    <body>
        <nav th:replace="layout::header"></nav>
        <h1>AAAAAAAAAAA</h1>
        <h1>AAAAAAAAAAA</h1>
        <h1>AAAAAAAAAAA</h1>
        <h1>AAAAAAAAAAA</h1>
        <a th:href="@{/contact/create}" class="btn btn-success pull-left">
            <span class="glyphicon glyphicon-plus"></span> Add new contact
        </a>
        <div th:if="${success}" class="row alert alert-success
alert-dismissible">
            <button type="button" class="close" data-dismiss="alert"
aria-label="Close"><span aria-hidden="true">&times;</span></button>
            <span th:text="${success}"></span>
        </div>
        <div class="container main-content list">
            <th:block th:if="${#lists.isEmpty(contacts)}">
```

```
                <h3>No contacts</h3>
            </th:block>

            <th:block th:unless="${#lists.isEmpty(contacts)}">
                <div class="row">

                    <form class="form-inline pull-right">
                        <div class="form-group">
                            <input type="text" class="form-control"
name="criteria"
                                   placeholder="Type contact name..." />
                        </div>
                        <button type="submit" class="btn
btn-primary">Search</button>
                    </form>
                </div>
                <div class="row">
                    <table class="table table-bordered table-hover">
                        <thead>
                            <tr>
                                <th>NO</th>
                                <th>Name</th>
                                <th>Email</th>
                                <th>Phone</th>
                                <th>Edit</th>
                                <th>Delete</th>
                            </tr>
                        </thead>
                        <tbody>
                            <tr th:each="contact,iterStat : ${contacts}">
                                <td th:text="${iterStat.count}"></td>
                                <td th:text="${contact.name}"></td>
                                <td th:text="${contact.email}"></td>
                                <td th:text="${contact.phone}"></td>
                                <td><a
th:href="@{/contact/{id}/edit(id=${contact.id})}"><span class="glyphicon
glyphicon-pencil"></span></a></td>
                                <td><a
th:href="@{/contact/{id}/delete(id=${contact.id})}"><span class="glyphicon
glyphicon-trash"></span></a></td>
                            </tr>
                        </tbody>
                    </table>
                </div>
            </th:block>
        </div>
        <!-- /.container -->

        <footer th:replace="layout::footer"></footer>
    </body>
</html>
```

**Trang thêm mới**

Bổ sung thêm file resource chứa các thông báo lỗi. Tạo file messages.properties

```
NotEmpty.contact.name=Vui lòng nhập tên cho liên hệ
Email.contact.email=Vui lòng nhập đúng định dạng email
```

Tạo class WebMvcConfig trong package com.chungnn.smartkids.config để khai báo bean cho việc lấy nội dung thông báo lỗi

```
package com.chungnn.smartkids.config;

import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.context.support.ReloadableResourceBundleMessageSource;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
    @Bean
    public MessageSource messageSource() {
        ReloadableResourceBundleMessageSource bean = new
ReloadableResourceBundleMessageSource();
        bean.addBasenames("classpath:messages");
        return bean;
    }

}
```

Thêm vào file ContactController

```
    @GetMapping("/contact/create")
    public String create(Model model) {
        model.addAttribute("contact", new Contact());
        return "form";
    }

    @PostMapping("/contact/save")
    public String save(@Valid Contact contact, BindingResult result,
RedirectAttributes redirect) {
        if (result.hasErrors()) {
            return "form";
```

```
        }
        contactService.save(contact);
        redirect.addFlashAttribute("success", "Saved contact successfully!");
        return "redirect:/contact";
    }
```

Tạo file form.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head th:replace="layout::head"></head>
    <body>
        <nav th:replace="layout::header"></nav>

        <div class="container main-content form">
            <div class="row">
                <form action="#" th:action="@{/contact/save}"
th:object="${contact}"
                      method="POST" novalidate="novalidate">
                    <input type="hidden" th:field="*{id}" />
                    <div class="form-group">
                        <label>Name</label>
                        <input type="text" class="form-control"
                               th:field="*{name}" th:errorclass="field-error"
/>
                        <em th:if="${#fields.hasErrors('name')}"
th:errors="*{name}"></em>
                    </div>
                    <div class="form-group">
                        <label>Email</label>
                        <input type="email" class="form-control"
                               th:field="*{email}" th:errorclass="field-error"
/>
                        <em th:if="${#fields.hasErrors('email')}"
th:errors="*{email}"></em>
                    </div>
                    <div class="form-group">
                        <label>Phone</label>
                        <input type="text" class="form-control"
th:field="*{phone}" />
                    </div>
                    <button type="submit" class="btn
btn-primary">Save</button>
                </form>
            </div>
        </div><!-- /.container -->

        <footer th:replace="layout::footer"></footer>
    </body>
</html>
```

**Trang cập nhật & Xóa**

Thêm đoạn code vào trong controller

```
    @GetMapping("/contact/{id}/edit")
    public String edit(@PathVariable int id, Model model) {
        model.addAttribute("contact", contactService.findOne(id));
        return "form";
    }

    @GetMapping("/contact/{id}/delete")
    public String delete(@PathVariable int id, RedirectAttributes redirect) {
        contactService.delete(id);
        redirect.addFlashAttribute("success", "Deleted contact
successfully!");
        return "redirect:/contact";
    }
```

Như vậy là chúng ta đã hoàn thành webapp đơn giản thực hiện CRUD với 1 bảng contact

**4. Thiết lập để có thể thực hiện chạy service từ file jar sau khi đóng gói ứng dụng spring boot:**

Đầu tiên, sửa lại trong file pom để khi đóng gói IDE sẽ đưa thêm file manifest

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <executable>true</executable>
                </configuration>
            </plugin>
        </plugins>
        <finalName>The_jar_name</finalName>
    </build>
```

```
$ sudo chmod +x /path/to/your-app.jar
$ sudo ln -s /path/to/your-app.jar /etc/init.d/your-app
$ sudo service your-app start
```