# Problem 1 – Concert

*Mandy was hired to accept the group applications for the upcoming concert. She thought that it would be easy but now she is in trouble. She needs a programmer to make an application that will help her to safe the concert. She needs you.*

You will read commands until you receive **"start of concert"** command.

There are **two types** of commands:

* "**Add;** **{bandName}; {member 1}, {member 2}, {member 3}"** - applies a band and a **list of members** to the concert. All members must be **unique** so don't add duplicates. If you receive the **same band** twice add only those members that **aren't present** in the list.
* **"Play; {bandName}; {time}"** – the band with the given name plays an **amount of time** on the stage. If you receive a **band** that has **already** **applied** in the concert, just **increase** the band **time.**

If in both commands the band **does not exist**, add it.

At the end you have to print the **total time** and the bands ordered by the **time** on stage in **descending** order, then by **band name** in **ascending** order.

Also the **final input line** will be "**{bandName}**" and you have to print **all members** for this band in **insertion order**.

## Input / Constraints

* The **time** of the bands – **integer** in range **[0 – 231 - 1]**
* There will always be at least one **member** in the group
* The bands will always have **time** on stage
* The final input line will always contain an **existing** band name
* Input will always be valid and in the range specified. You don’t need to check it

## Output

**Total time: {totalTime}**

**{firstBandName} -> {firstBandTime}**

**{secondBandName} -> {secondBandTime}**

**{thirdBandName} -> {thirdBandTime}**

**…**

**{bandName}**

**=> {firstMemberName}**

**=> {secondMemberName}**

**=> {thirdMemberName}**

**…**

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Play; The Beatles; 2584  Add; The Beatles; John Lennon, Paul McCartney, George Harrison, Ringo Starr  Add; Eagles; Glenn Frey, Don Henley, Bernie Leadon, Randy Meisner  Play; Eagles; 1869  Add; The Rolling Stones; Brian Jones, Mick Jagger, Keith Richards  Add; The Rolling Stones; Brian Jones, Mick Jagger, Keith Richards, Bill Wyman, Charlie Watts, Ian Stewart  Play; The Rolling Stones; 4239  start of concert  The Rolling Stones | Total time: 8692  The Rolling Stones -> 4239  The Beatles -> 2584  Eagles -> 1869  The Rolling Stones  => Brian Jones  => Mick Jagger  => Keith Richards  => Bill Wyman  => Charlie Watts  => Ian Stewart |
| Add; The Beatles; John Lennon, Paul McCartney  Add; The Beatles; Paul McCartney, George Harrison  Add; The Beatles; George Harrison, Ringo Starr  Play; The Beatles; 3698  Play; The Beatles; 3828  start of concert  The Beatles | Total time: 7526  The Beatles -> 7526  The Beatles  => John Lennon  => Paul McCartney  => George Harrison  => Ringo Starr |

**Problem 02. Song Encryption**

*Now that you've helped Mandy to accept the group applications it's time to assist her with a security problem. You are tasked to encrypt all songs from the set list so that if someone steals it they won't be able to leak it online.*

Your task is to write a program that encrypts information about artists and their songs.

**Until** you receive the command **"end"** you should read lines in following format :"**{artist}:{song}",** and check if it's **valid**, considering the following rules:

* **Artist** – **starts** with **capital letter**, followed by **lowercase letters.**
  + It can also **contains apostrophe ( ' ),** and **whitespace " ";**

**Valid group name:** Red hot chili peppers, Eminem, Guns n' roses

**Invalid group name**: ReD Hot CiLly PePers, sLipKnot, guns n'roses

* **Song – contains only capital letters, and whitespaces.**

**Valid songs:** BACK IN BLACK, BLEED IT OUT, KILLSHOT

**Invalid songs:** #BaCk IN black, BLEED $IT$ OUTt, &KILLSHoT

After you validate the lines of an input, you should encrypt the information. In order to do that, you have to follow the rules below:

* **First you need to find a key for encryption.**
  + Your key is the **length** of the **artist (e.g. "Eminem" – key: 6)**
* You have to **increment** the **ASCII value** of each symbol of the input, with the **key length**

**(**e.g. "char" **'a'** -> **'g')**

* + Be careful if your key length is **bigger** than the ASCII value of a letter **'z'** or **'Z'**. In this case you should start from a letter **'a'. (**e.g. key:6 letter – '**x**', encrypted letter – '**d**'**)**
* You should **NOT ENCRYPT** the following characters: **whitespaces**, and **apostrophes**
* You also should **replace** **':'** with the sign **'@'**

**Input / Constraints**

Until you receive **"end"** command you should read from the console.

* **Line of input** – Artist name and song, **separated** by **":"**, containing only **ASCII symbols.**
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

**Output**

After every line of input, you should print:

* **If** line is valid – **encrypted information** in following format:

**"Successful encryption:** **{encryptedArtist}@{encryptedSong}".**

* **If** line is not valid – print the following message: **"Invalid input!"**

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| Eminem:VENOM  Linkin park:NUMB  Drake:NONSTOP  Adele:HELLO  end | Successful encryption: Ksotks@BKTUS  Successful encryption: Wtyvty alcv@YFXM  Successful encryption: Iwfpj@STSXYTU  Successful encryption: Fijqj@MJQQT | All lines of input **are valid**, so we  encrypt the information, change the charracter ":" with the sign **"@"**, and print the output of encription.  *Example*: **Eminem-> key 6**, adding a key to the ASCII value of each charracter except the **whitespace**, **apostrophe**, and our **delimiter**(":") and receive an encrypted name – **Ksotks@BKTUS**, then we do the same with the song. |
| **Input** | **Output** | **Comments** |
| Michael Jackson:ANOTHER PART OF ME  Adele:ONE AND ONLY  Guns n'roses:NOVEMBER RAIN  Christina Aguilera: HuRt  end | Invalid input!  Successful encryption: Fijqj@TSJ FSI TSQD  Successful encryption: Sgze z'daeqe@ZAHQYNQD DMUZ  Invalid input! | First line in not valid, because in the name of Michael Jackson we have more than one capital letter. Next two are valid, and the last is not valid, because the song does not contain only capital letters. |

**03. Basic CRUD – Band Register**

You have been tasked to create a simple application for a Band Register. The application should hold **bands**, which are the main app **entities**. The app is called **BandRegister**.

The functionality of the application should support **creating**, **listing, editing**, **deleting** bands.

The application should **persist** the data into a **database**.

**Overview**

Your application should be built on **one** of the **following technologies**:

**JavaScript**

* **NodeJS** + **ExpressJS** frameworks
* **Handlebars.js** view engine
* **Mongoose** ORM
* **MongoDB**

**PHP**

* **Symfony** framework
* **Twig** view engine
* **Doctrine** ORM
* **MySQL** database

**Java**

* **Spring** framework (**Spring MVC** + **Spring Boot** + **Spring Data**)
* **Thymeleaf** view engine
* **JPA** / **Hibernate ORM** + **Spring Data** data access
* **MySQL** database

**C#**

* **ASP.NET Core** framework (**ASP.NET MVC** + **Entity Framework Core**)
* **Razor** view engine
* **Entity Framework Core** ORM
* **SQL Server** database\

**Data Model**

The **Band** entity holds **5 properties**:

* **id** – technology-dependent identifier (**ObjectID** for JavaScript, **int** for all other technologies)
* **name** – non-empty text
* **members** – non-empty text
* **honorarium** – non-null floating-point number
* **genre** – non empty text

**Project Skeletons**

You will be given the applications’ **skeletons**, which holds about **90%** of the logic. You’ll be given some **files** (**controllers**, **models** etc.). The files will have **partially implemented logic**, so you’ll need to write some code for the application to **function properly**.

The application’s views will be given to you fully implemented. You only need to include them in your business logic.

**Each technology** will have its **own skeleton**, and the **different** **skeletons** may **differ** in **terms** of **what is given to you** and **what is to be implemented**.

Everything that has been given to you inside the skeleton is **correctly implemented** and if you write your code **correctly**, the application should work just fine. You are free to change anything in the Skeleton on your account.

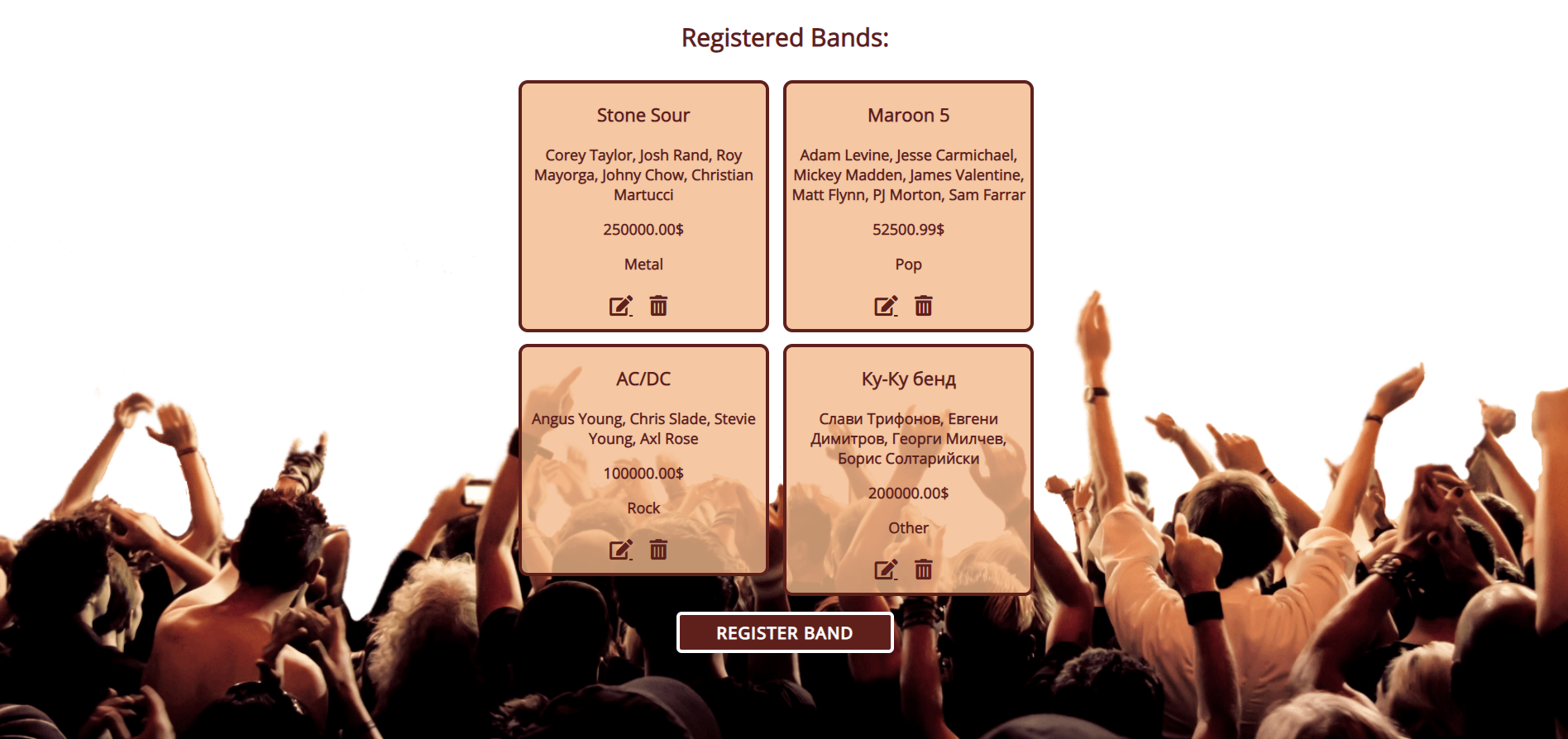
**User Interface**

This is the user interface or how the application’s pages should look in their final form (fully implemented). You have several pages, described below:

**Index Page**

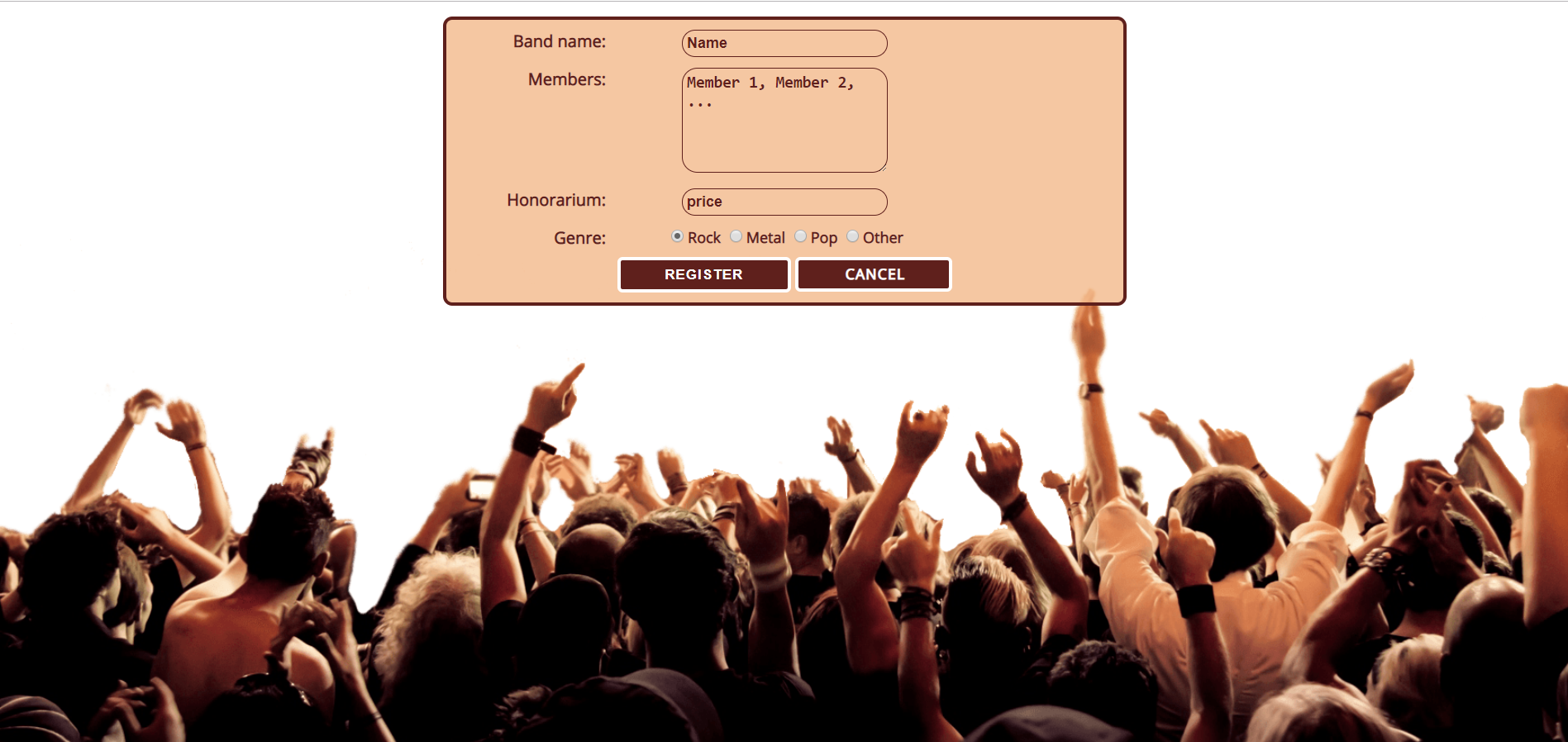
**Route: "/"**

Displays **all** the **bands** from the database with **options** to **edit** or **delete** them.



**Create Page**

**Route: "/create"**



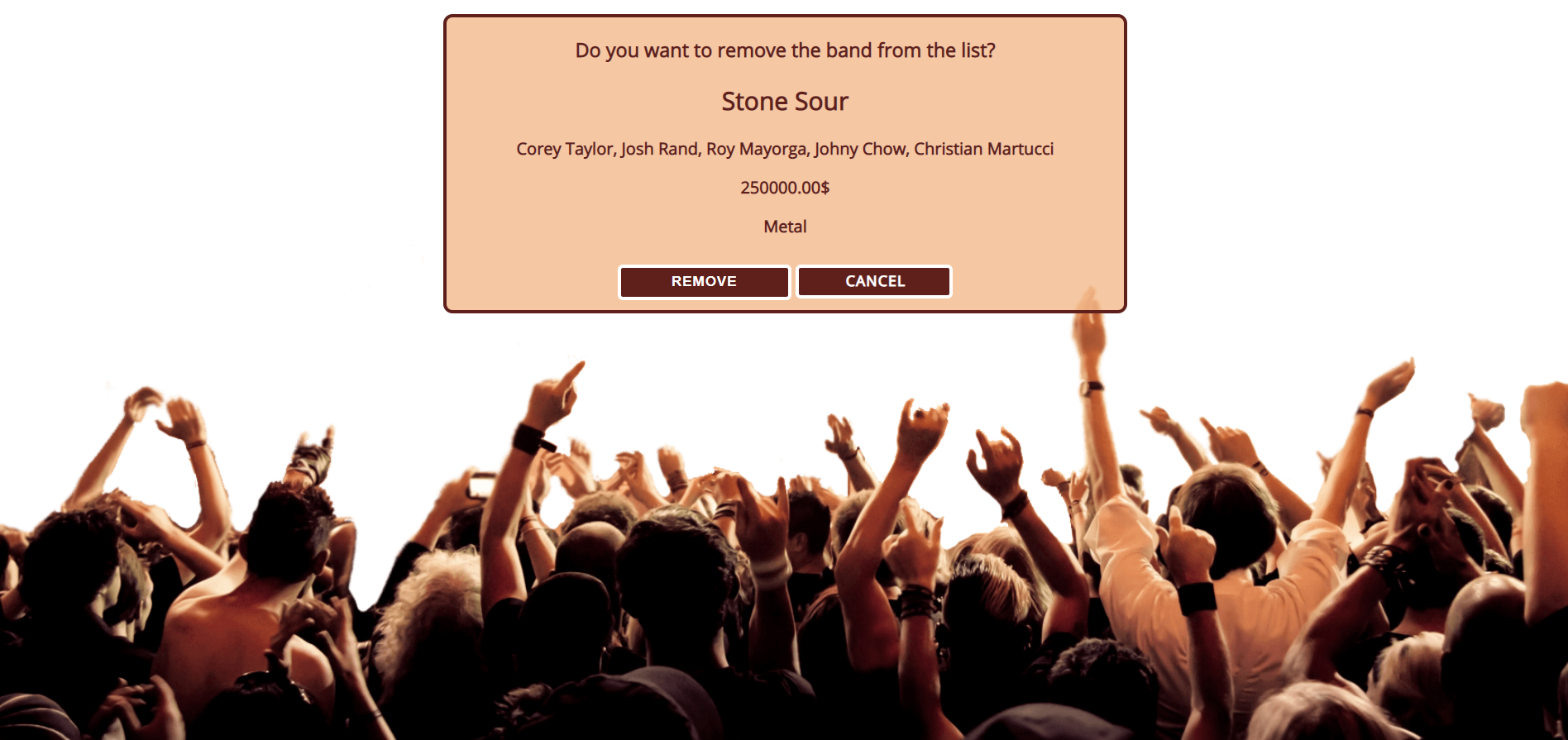
**Edit Page**

**Route: "/edit/{id}"**



**Delete Page**

**Route: "/delete/{id}"**



**Problem**

As you can see the different pages are on different routes. Most of the routing logic will be given to you in the **Skeleton**, but you should make sure that the application **works properly**.

Implement the "**Band Register" app** using only **your technology.**

**Setup**

Before you start working, make sure you **download all the dependencies** (packages) required for your technology and **set up** the **databases**! Below are instructions on how to do this:

**PHP and Symfony**

1. Make sure you've started your **MySQL server** (either from **XAMPP** or standalone)
2. Open a **Terminal in PHPStorm** or **shell** / **command prompt** / **PowerShell** window in the **root directory**: [Shift] + [Right click] 🡺 [Open command window here]
3. Enter the "**composer install**" command to restore its **Composer dependencies**   
   (described in **composer.json**)
4. Enter the "**php bin/console doctrine:database:create --if-not-exists**" command
5. Done!

**JavaScript and Node.js**

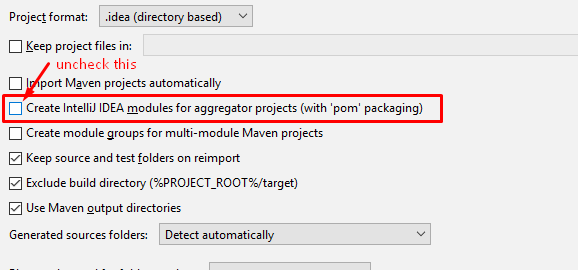
1. Go into the **root directory** of the project (where the **index.js** file is)
2. Open a **shell** / **command prompt** / **PowerShell** window in the **root directory**: [Shift] + [Right click] 🡺 [Open command window here]
3. Enter the “**npm install**” command to restore its **Node.js dependencies** (described in **package.json**)
4. Type **node index.js** to start the server
5. Done!

**C# and ASP.NET**

The C# project will automatically resolve its **NuGet dependencies** (described in **packages.config**) using the NuGet package restore when the project is built.

**Java and Spring MVC**

When you import your project, you should **uncheck "**Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)**"**:



This project is **set up to use Java jdk 1.8.** If your version is different, you can change it in **Maven dependencies** like this:



The Java project will automatically resolve its **Maven dependencies** (described in **pom.xml**) when the project is built.