Short Keys in Eclipse

1. **Cmd + option + S - source code generator**
2. Cmd + shift + O - organize imports
3. Cmd + shift + M - add imports
4. Cmd + option + j - for file comment
5. Cmd + 1 - for auto suggestion
6. Cmd + option + T - extract class, interface, other
7. Cmd + option + T - Move file to another package
8. Cmd + space - auto suggestion

Hello world API

CRUD Api Implementation in Spring Boot

Create your first api locally with DB :-

1. *After creating the project , create a file named TestController or StudentController and write these lines of code.*
2. *Add annotation and all eg code.*
3. *Open your application.java file and run as Spring boot app*
4. *Create a POJO Model without annotation and return it as it is and it will automatically convert to JSON.*
5. *Open browser and hit api localhost:8080/test*
6. *Done*
7. *Optional on error - to forcing the build https://stackoverflow.com/questions/6021899/eclipse-maven-multiple-annotations-found-at-this-line/18800625*
8. *Optional on JRE - select project=> option(right click)=> configure build path => Library => set jdk to one that you selected at project initialization*

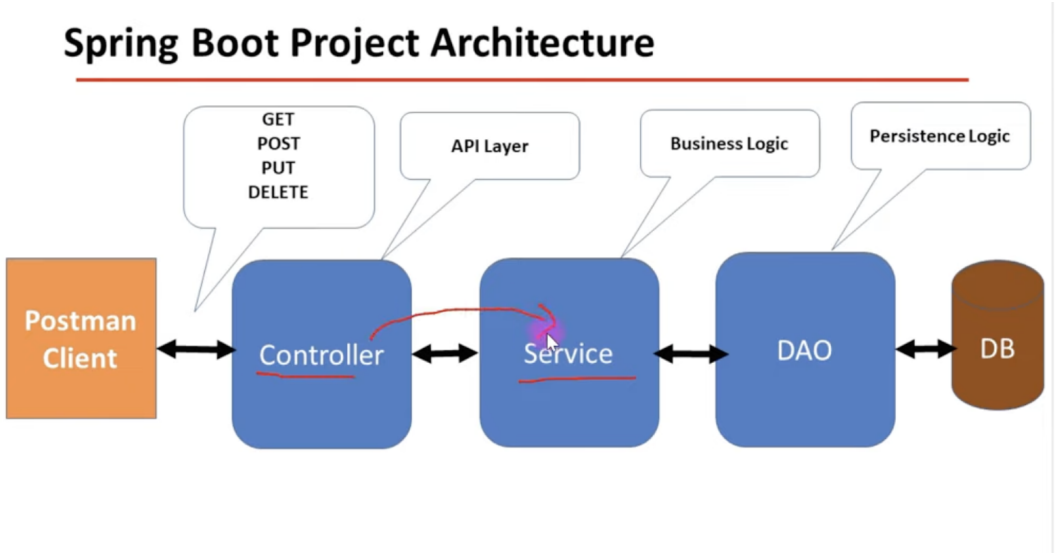
Code examples :)

```
package com.example.test2;

import java.util.ArrayList;
import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
```

```java
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class StudentController {

	@GetMapping("/student")
	public Student getStudent() {
		return new Student("Ramesh","Kumar");
	}
	@GetMapping("/students")
	public List<Student> getStudents() {
		List<Student> studentList = new ArrayList<>();
		studentList.add(new Student("name1","last"));
		studentList.add(new Student("name2","last"));
		studentList.add(new Student("name3","last"));
		studentList.add(new Student("name4","last"));
		studentList.add(new Student("name5","last"));
		studentList.add(new Student("name6","last"));
		studentList.add(new Student("name7","last"));
		studentList.add(new Student("name8","last"));
		return studentList;
	}

	@GetMapping("/student/{firstName}/{lastName}")
	public Student studentPathVariable(
			@PathVariable("firstName") String firstName,
			@PathVariable("lastName") String lastName) {
		return new Student("akash", "soni");
	}

	@GetMapping("/student/query")
	public Student studentQueryParam(
			@RequestParam(name = "firstName") String firstName,
			@RequestParam(name = "lastName") String lastName
			) {

		return new Student(firstName, lastName);
	}
```

}

CRUD Api With MySql DB



Setup Workspace

1. *Create new starter project*
2. *Add **spring web, Lombok, Spring data jpa, mysql driver***
3. *Optional on error - to forcing the build https://stackoverflow.com/questions/6021899/eclipse-maven-multiple-annotations-found-at-this-line/18800625*
4. ***Optional on JRE - select project=> option(right click)=> configure build path => Library => set jdk to one that you selected at project initialization***
5. *Configure mysql db*
    a. *Open application.properties file and add datasource and hibernate dialect properties*
        i. *spring.datasource.url=jdbc:mysql://localhost:3306/ems?useSSL=false*
        ii. *spring.datasource.username=root*
        iii. *spring.datasource.password=root*
        iv. *Hibernate properties*
        v. *spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect*

vi. *Create, create-drop*
vii. *spring.jpa.hibernate.ddl-auto=update*

b. *Jdbc driver*
c. *Jdbc url*
d. *Jdbc username*
e. *Jdbc paw*

6. *Create package structure for your workspace*
   a. ***Controller***
   b. ***Exception***
   c. ***Model***
   d. ***Repository***
   e. ***Service***
   f. ***Service.impl***
7. *Optional - Remember Controllers created outside the main package won't work. So create all packages inside your main package e.g. (com.test.Main)*
8. *Run query on **MySQL Workbench create database ems**; Refresh and check the db is created in ems or not*
9. *Configure in spring boot project in **application.properties** and add /ems in **datasource.url***
10. *Configure mysql db*
    a. *Open application.properties file and add datasource and hibernate dialect properties*
       i. *spring.datasource.url=jdbc:mysql://localhost:3306/ems?useSSL=false*
       ii. *spring.datasource.username=root*
       iii. *spring.datasource.password=root@1234*
       iv. *#Hibernate properties*
       v. *spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect*
       vi. *#Create, create-drop*
       vii. *spring.jpa.hibernate.ddl-auto=update*
    b. *Jdbc driver*
    c. *Jdbc url*
    d. *Jdbc username*
    e. *Jdbc paw*
11. *Now run your project and check everything work fine - Successfully setup*

Create JPA Entity

1. Create a model named Employee.java
2. **@Data** - lombok => **Add lombok class notation** with a class that will create a Getter Setter automatically. Or create source from constructors, getter and setters **short key 1**, lombok reduce boilerplate code like getter, setter, constructor, toString ...
3. **@Entity - javax.persistence** - from persistence class that specify that **class is an entity**, jpa entity
4. **@Table - javax.persistence** - this annotation have property name that will define the table name, if not then it will automatically take the class name
5. Id
    a. Use **@Id to create primary** key
    b. Use **@GenerateValue(strategy = GenerationType.IDENTITY)** to mek it auto generated id of long type.
    c. Lets define
6. Firstname - now add column details for other fieldss
    a. @Column(name= “first_name”, nullable = false)
7. Do this for other type as well
    a. @Column(name= “email”)
    b. @Column(name= “last_name”)
8. Hibernate will create these properties automatically. Let’s run and see if the table is created in MySql or not. Open workbench tap on home and tap on your sql workspace
9. Done - we have successful created our employee jpa entity

```
package com.poc.springboot.Model;

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

@Entity
@Table(name="employee")
public class Employee{

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private long id;

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

        @Column(name="last_name")
        private String lastName;

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

```
	public Employee() {
		super();
	}

	public Employee(long id, String firstName, String lastName, String email) {
		super();
		this.id = id;
		this.firstName = firstName;
		this.lastName = lastName;
		this.email = email;
	}

	public long getId() {
		return id;
	}

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

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public String getEmail() {
		return email;
	}

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



}
```

Create Repository

1. Name the interface EmoployeeRepository extends JPARepository<Employee, Long>
2. Theory -
    a. Reqpo is responsible to our CRUD operations
    b. This is a interface extends to our JPARepository
    c. Spring Data JPA internally provides @Repository annotation so we don't need to add @repository annotation to Employee Repository interface
    d. @transactional is also defined in jap repo

Create Exception

1. **Create Exception class** in Exception package name it **ResourceNotFoundException** that extends **RuntimeException**
    e. Resource not found exception
    f. **extends** RuntimeException
    g. Add annotation **@ExceptionStatus(value = HttpStatus.NOT_FOUND)**
    h. private static final long serialVersionUID = 1L;
    i. Add property private **String resourceName,fieldName, Object fieldValue**;
    j. Generate constructors for all the fields and add default message with format
    k. super(String.format("%s not found with %s : '%s'",resourceName, fieldName, fieldValue));
    l. Generate getters
    m. Done

```
package com.poc.springboot.Exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException{

	private static final long serialVersionUID = 1L;
	private String resourceName,fieldName;
	private Object fieldValue;
	public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {
		super(String.format("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));
		this.resourceName = resourceName;
		this.fieldName = fieldName;
		this.fieldValue = fieldValue;
	}
	public String getResourceName() {
```

```
		return resourceName;
	}
	public void setResourceName(String resourceName) {
		this.resourceName = resourceName;
	}
	public String getFieldName() {
		return fieldName;
	}
	public void setFieldName(String fieldName) {
		this.fieldName = fieldName;
	}
	public Object getFieldValue() {
		return fieldValue;
	}
	public void setFieldValue(Object fieldValue) {
		this.fieldValue = fieldValue;
	}

}
```

Create Service

1. Create new interface in Service package name EmployeeService - don't forget to change package from com.poc.springboot.Service.impl to **com.poc.springboot.Service**
2. Create new class in Service.impl package name EmployeeServiceImpl
3. EmployeeServiceImpl Implement EmployeeService
4. **@Service annotation EmployeeServiceImpl**
5. Define a method in interface EmployeeService - `Employee saveEmployee(Employee employee);`
6. Implement the method in class EmployeeServiceImpl
7. Theory
   In Employee impl - There are two type of dependency injection
   1. Setter based dependency injection
   2. Constructor based dependency injection
   We will use constructor base constructor based dependency
   Spring 4.3 if a class configured as a spring bean has only one constructor, @autowired can be omitted and spring will use that constructor and inject all necessary dependency
8. Add auto generated method stub in impl file
9. Create private EmployeeRepository employeeRepository;
10. Create its constructor
11. Call save method from repository in override method and return Employee.

Create Controller

1. Create class EmployeeController in controller package
2. Add **@RestController** annotation
    a. Theory
    b. RestController is a convenient annotation that combines @controller and @ResponseBody, which eliminates the need to annotation every request handling method of the controller class with the @Response/body annotation.
3. Create property for service - private EmployeeService employeeService;
4. Create constructor
5. Create endpoint method that will hold request and response paths - **saveEmployee**
6. Use return type as **ResponseEntity** - to add status and header automatically
7. Add @postmaping and @RequestBody to this **@postMapping public ResponseEntity<Employee> saveEmployee(@RequestBody Employee employee) {**
8. **return new object of ResponseEntity have two param service.save(employee), code as CREATED}**

```
package com.poc.springboot.Controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

import com.poc.springboot.Model.Employee;
import com.poc.springboot.Service.EmployeeService;

@RestController
public class EmployeeController {
	private EmployeeService employeeService;

	public EmployeeController(EmployeeService employeeService) {
		super();
		this.employeeService = employeeService;
	}

	@PostMapping("/save")
	public ResponseEntity<Employee> saveEmployee(Employee employee){
		return new ResponseEntity<Employee>(employeeService.saveEmployee(employee),
HttpStatus.CREATED);
	}

}
```

PostMan

Testing url
http://localhost:8080/api/mobile/employee/test

http://localhost:8080/api/mobile/employee/save

Create api post type
body raw type and JSON
Create body


```
{
  "fristName":"akash",
  "lastName":"soni",
  "email":"akashsoni@gmail.com"
}
```


Hit api
And hit


```
{
  "timestamp": "2021-11-28T07:46:58.485+00:00",
  "status": 404,
  "error": "Not Found",
  "path": "/api/mobile/employee/save"
}
```


Further API creation Step

1. Create Service
2. Create service impl
3. Goto controller add method and path
4. Done

Get employee by id example

Create Service

```
Employee findById(long id);
```

Create service impl

```
@Override
public Employee findById(long id) {
	// TODO Auto-generated method stub
	/*
	Optional<Employee> employee = employeeRepository.findById(id);
	if (employee.isPresent()){
		return employee.get();
	}else {
		throw new ResourceNotFoundException("Employee", "Id", employee);
	}
	 * */

	// or use lambda function

	return employeeRepository.findById(id).orElseThrow( () -> new
ResourceNotFoundException("Employee", "Id", id));
}
```

Goto controller add method and path

```
@GetMapping("{id}")
public ResponseEntity<Employee> getEmployeeById(@PathVariable("id") long id){
	return new ResponseEntity<Employee>(employeeService.findById(id), HttpStatus.OK);
}
```

Done

Update Employee by id

```
Employee updateEmployeeById(Employee employee, long id);
```

```
@Override
public Employee updateEmployeeById(Employee employee, long id) {
	// TODO Auto-generated method stub
	Employee existingEmployee = employeeRepository.findById(id).orElseThrow( () -> new
ResourceNotFoundException("Employee", "Id", id));
	existingEmployee.setFirstName(employee.getFirstName());
```

```
		existingEmployee.setLastName(employee.getLastName());
		existingEmployee.setEmail(employee.getEmail());
		employeeRepository.save(existingEmployee);
		return existingEmployee;
	}



	@PutMapping("{id}")
	public ResponseEntity<Employee> updateEmployeeById(@RequestBody Employee employee,
@PathVariable("id") long id){
		return new ResponseEntity<Employee>(employeeService.updateEmployeeById(employee,id),
HttpStatus.OK);
	}
```

Delete Employee by id

```
	Employee deleteEmployeeById(long id);

	@Override
	public Employee deleteEmployeeById(long id) {
		// TODO Auto-generated method stub
		Employee existingEmployee = employeeRepository.findById(id).orElseThrow( () -> new
ResourceNotFoundException("Employee", "Id", id));
		employeeRepository.delete(existingEmployee);
		return existingEmployee;
	}

	@DeleteMapping("{id}")
	public ResponseEntity<Employee> deleteEmployeeById(@PathVariable("id") long id){
		return new ResponseEntity<Employee>(employeeService.deleteEmployeeById(id),
HttpStatus.OK);
	}
```

What is REST?
What is SOAP?
Spring boot vs Spring MVC?
Sorting boot - basic question
System design - no
Spring boot - everything, service integration (get, put, post), security

Transient variable -

Variables may be marked transient to indicate that they are not part of the persistent state of an object.

Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:

1. **Autoconfiguration**
    a. Autoconfiguration means that applications are initialized with pre-set dependencies that you don't have to configure manually. As Java Spring Boot comes with built-in autoconfiguration capabilities, it automatically configures both the underlying Spring Framework and third-party packages based on your settings (and based on best practices, which helps avoid errors). Even though you can override these defaults once the initialization is complete, Java Spring Boot's auto configuration feature enables you to start developing your Spring-based applications fast and reduces the possibility of human errors.
2. **An opinionated approach to configuration**
    a. Spring Boot uses an opinionated approach to adding and configuring starter dependencies, based on the needs of your project. Following its own judgment, Spring Boot chooses which packages to install and which default values to use, rather than requiring you to make all those decisions yourself and set up everything manually.
    b. You can define the needs of your project during the initialization process, during which you choose among multiple starter dependencies—called *Spring Starters*—that cover typical use cases. You run Spring Boot Initializr by filling out a simple web form, without any coding.

c. For example, the ‘Spring Web’ starter dependency allows you to build Spring-based web applications with minimal configuration by adding all the necessary dependencies—such as the Apache Tomcat web server—to your project. ‘Spring Security’ is another popular starter dependency that automatically adds authentication and access-control features to your application.
d. Spring Boot includes over 50 Spring Starters, and many more third-party starters are available.

3. **The ability to create standalone applications**
    a. Spring Boot helps developers create applications that *just run*. Specifically, it lets you create standalone applications that run on their own, without relying on an external web server, by embedding a web server such as Tomcat or Netty into your app during the initialization process. As a result, you can launch your application on any platform by simply hitting the Run command. (You can opt out of this feature to build applications without an embedded Web server.)

These features work together to provide you with a tool that allows you to set up a Spring-based application with minimal configuration and setup.

Gradle vs. Maven

Let's discuss some key differences between Gradle and Maven:

Gradle	***Maven***
*It is a build **automation system** that uses a **Groovy-based DSL** (domain-specific language)*	*It is a software project **management system** that is primarily used for **java projects**.*

*It does not use an **XML** file for declaring the **project configuration**.*	*It uses an **XML** file for declaring the project, its **dependencies**, the build order, and its required plugin.*
*It is **based on a graph** of task dependencies that do the work.*	*It is based on the phases of the **fixed and linear model**.*
*In Gradle, the **main goal** is to add **functionality to the project**.*	*In maven, the main goal is related to the **project phase**.*
*It avoids the work by tracking input and output tasks and **only runs** the tasks that have been **changed**. Therefore it gives a faster performance.*	*It does not use the build cache; thus, its **build time is slower than Gradle**.*
*Gradle is **highly customizable**; it provides a wide range of IDE support custom builds.*	*Maven has a limited number of parameters and requirements, so **customization is a bit complicated**.*
Gradle avoids the compilation of Java.	*The compilation is mandatory in Maven.*

Setup in Eclipse

Download mysql server or any DB like postgres
Download mysql workbench or Dbviewer
Db user : root
Db psw : root@1234
Connection name : Local instance

brew install eclipse-ide
Took reference from this website

1. Install eclipse ide
2. Install spring boot tools latest - go to help - marketplace - search spring
3. Create a new project for spring boot api - *File* > *New* > *Project*… and select the option *Spring starter Project.*
4. *Add project name, package name,* ***web app*** *add in dependency.*
5. *Done*

Setup Spring boot in Visual Studio Code

Step 1:
https://code.visualstudio.com/docs/java/java-spring-boot
Step 2:
https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-spring-initializr
Step 3:
Create a ***Maven*** *project Cmd+shift+p*
Step 4:
Version latest, add package name com….myshcedule, add artifect id myschedule
Step 5:
Choose jar, choose latest version of java,
Step 6: choose dependency

1. *spring* ***web,***
2. ***Spring data jpa,***
3. ***Lombok,***
4. ***mysql driver***

Instead of creating getter and setter for our classes we use lombok annotation to reduce the boilerplate code.

Spring Boot in 25 Minutes Udemy

[in28minutes/spring-microservices: Spring Microservices using Spring Cloud]

1. Init project
2. Spring Web, Spring Boot tools, Spring Data JPA, H2 Database
3.