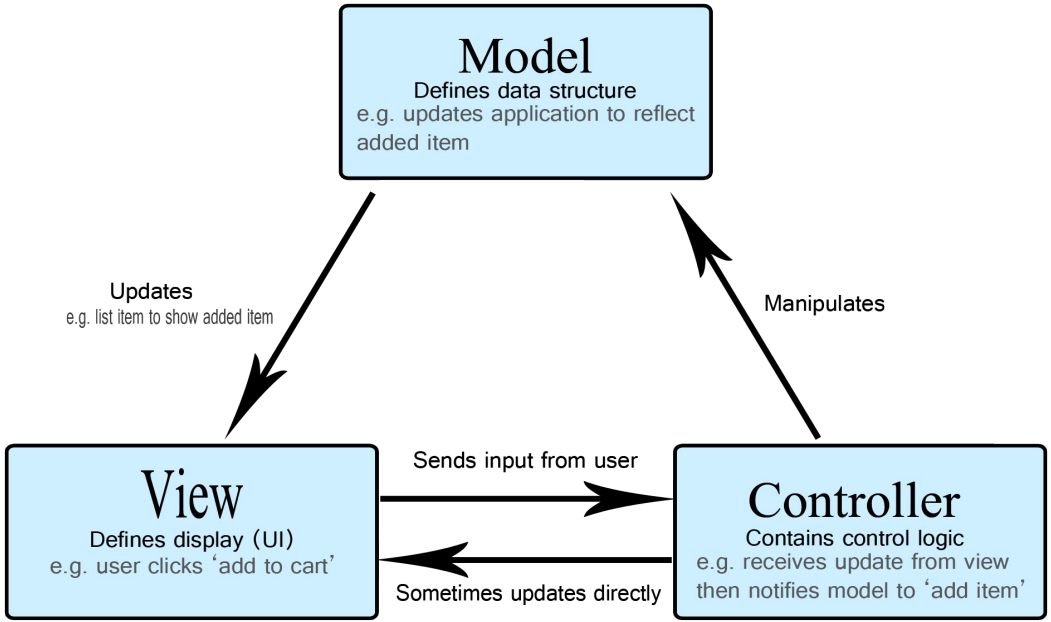
JAVA II

Spring MVC - Web pages, controllers, restfull api

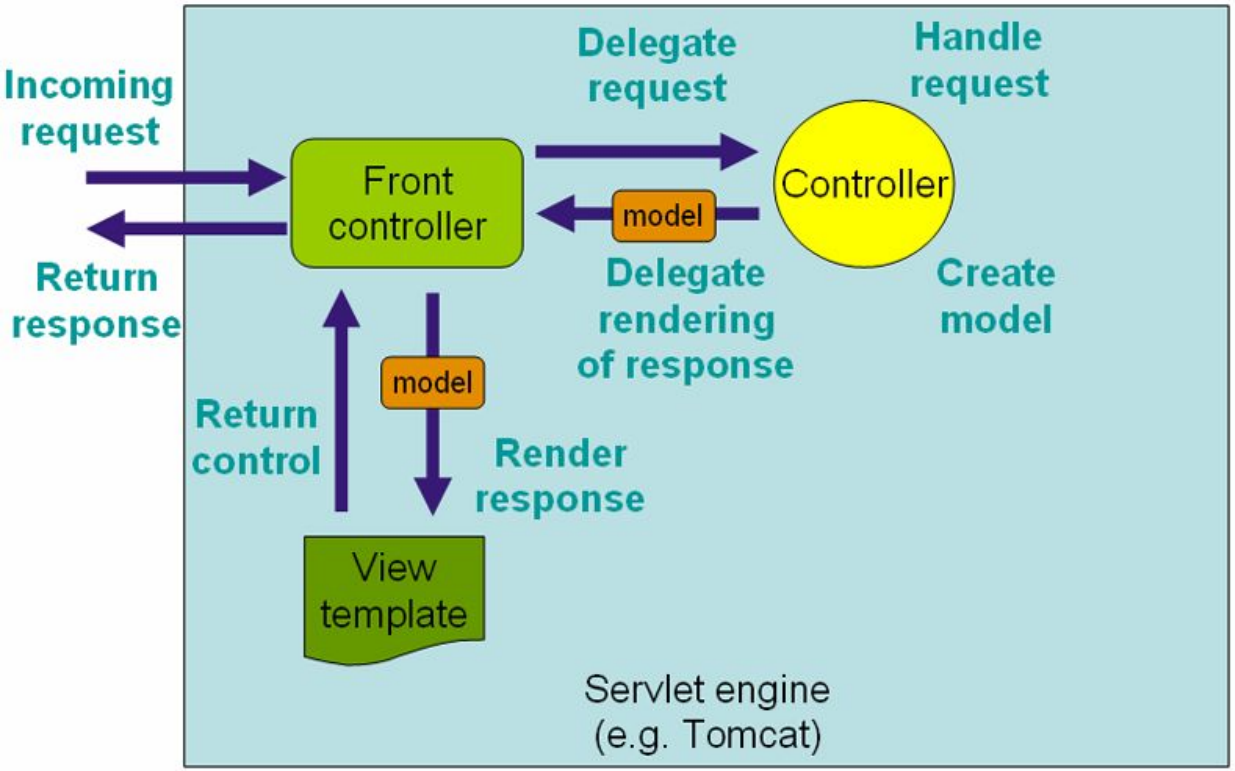
How to enable hot reload. Compile on app run

https://stackoverflow.com/questions/33869606/intellij-15-springboot-devtools-livereload-not-working

MVC architecture



How SPring MVC works (DispatcherServlet)



How to configure Spring MVC in application server (web.xml)

```
<web-app>

    <servlet>
        <servlet-name>example</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>example</servlet-name>
        <url-pattern>/example/*</url-pattern>
    </servlet-mapping>

</web-app>
```

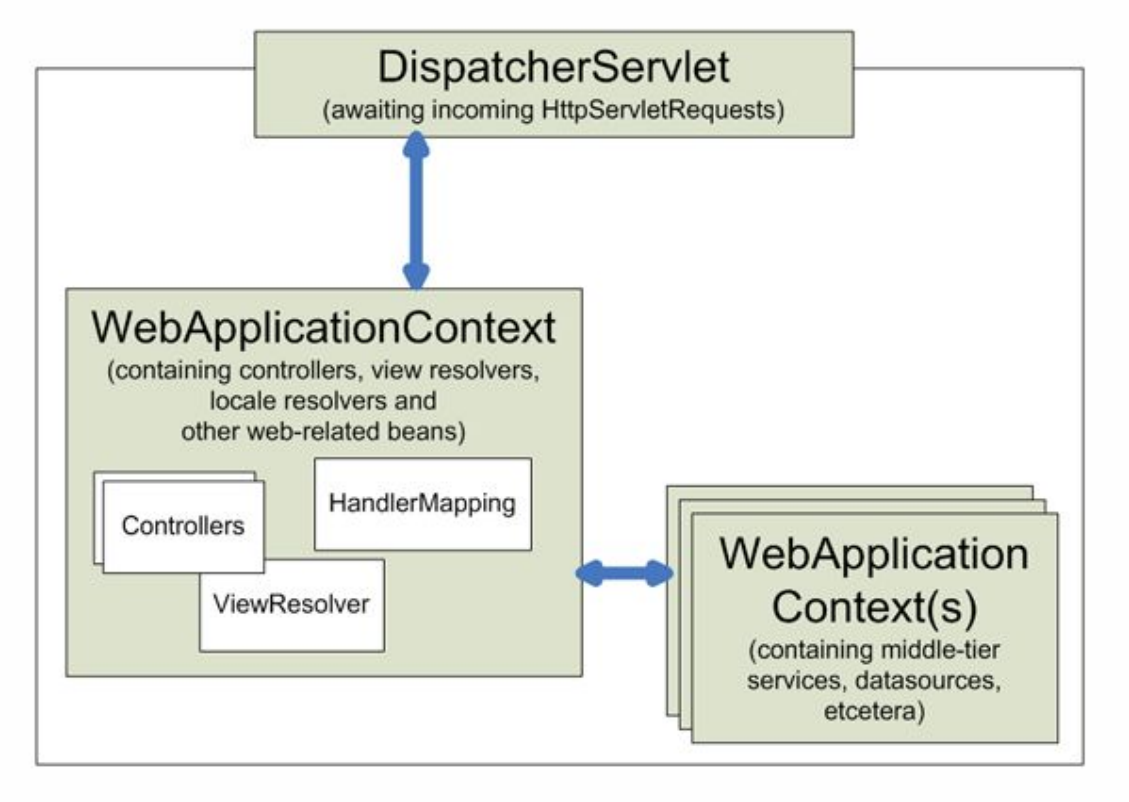
How to configure Spring MVC in application server (Java)

```java
public class MyWebApplicationInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext container) {
        ServletRegistration.Dynamic registration = container.addServlet("dispatcher", new DispatcherServlet());
        registration.setLoadOnStartup(1);
        registration.addMapping("/example/*");
    }

}
```

How dispatcher servlet works?



How to configure Spring MVC in Spring boot (I)

```
package lt.baltictalents.lesson4;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;


@SpringBootApplication
public class Lesson4Application {
   ApplicationContext applicationContext;

   public static void main(String[] args) {
      SpringApplication.run(Lesson4Application.class, args);
   }
}
```

How to configure Spring MVC in Spring boot (II)

```
package lt.baltictalents.lesson4.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@Configuration
@EnableWebMvc
public class MvcConfig implements WebMvcConfigurer {
  @Override
  public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
     configurer.enable();
  }

  @Override
  public void configureViewResolvers(ViewResolverRegistry registry) {
     InternalResourceViewResolver resolver = new InternalResourceViewResolver();
     resolver.setPrefix("/WEB-INF/view/");
     resolver.setSuffix(".jsp");
     resolver.setViewClass(JstlView.class);
     registry.viewResolver(resolver);
  }
}
```

Implementing controllers

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public String helloWorld(Model model) {
        model.addAttribute("message", "Hello World!");
        return "helloWorld";
    }
}
```

More practical Controller example

```java
@Controller
@RequestMapping("/model-and-view")
public class MyModelAndViewController {
    @RequestMapping(value={"", "/"})
    ModelAndView doGetModelAndView() {
        val modelAndView = new ModelAndView( viewName: "model-and-view");

        modelAndView.addObject( attributeName: "id",  attributeValue: 1);

        return modelAndView;
    }

    @RequestMapping("/{id}")
    ModelAndView doGetModelAndView(@PathVariable("id") Optional<Integer> id) {
        val modelAndView = new ModelAndView( viewName: "model-and-view");

        if (id.isPresent()) {
            modelAndView.addObject( attributeName: "id", id.get());
        }

        modelAndView.addObject(1);

        return modelAndView;
    }

    @RequestMapping("/model/{id}")
    String doGetModel(Model model, @PathVariable("id") Optional<Integer> id) {
        if (id.isPresent()) {
            model.addAttribute( attributeName: "id", id.get());
        }

        model.addAttribute( attributeName: "id", id.get());

        return "model-and-view";
    }
}
```

http://localhost:8080/model-and-view
http://localhost:8080/model-and-view/1
http://localhost:8080/model-and-view/5
http://localhost:8080/model-and-view/model/5
http://localhost:8080/model-and-view/model

What annotation and classes we will use with controllers?

- @Controller
- @RestController (for rest services)
- @RequestMapping
- @PathVariable
- @Autowired
- @MatrixVariable
- @RequestParam
- @ResponseBody
- and more :)

- HttpServletRequest
- HttpServletResponse
- ResponseEntity
- HttpHeaders
- HttpStatus
- Model
- ModelAndView
- RequestMethod
- and more :)

Examples, examples, examples ...

http://localhost:8080/examples

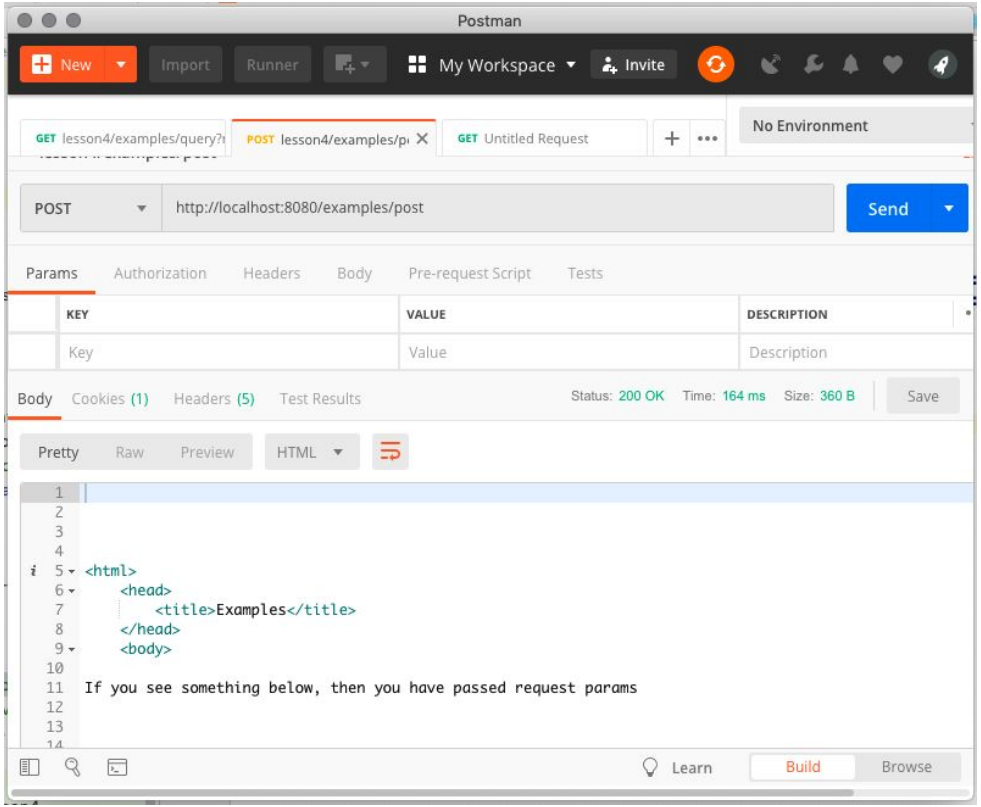
http://localhost:8080/examples/redirect

http://localhost:8080/examples/query?name1=value1&name2=value2

http://localhost:8080/examples/post (Use postman for testing post)

http://localhost:8080/examples/json

Postman as tool for debugging requests



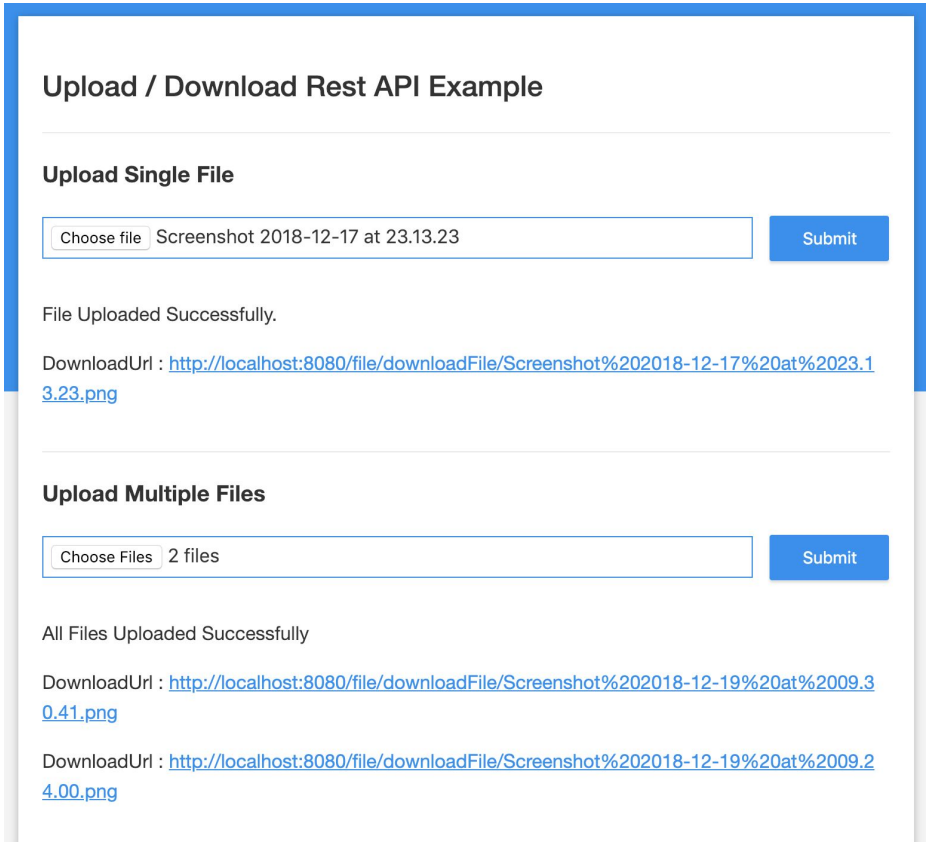
REST - Representational State Transfer

REST (Representational State Transfer) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. REST is an architectural style for designing distributed systems. It is not a standard but a set of constraints, such as being stateless, having a client/server relationship, and a uniform interface. REST is not strictly related to HTTP, but it is most commonly associated with it.

Principles of REST:

- Resources expose easily understood directory structure URIs.
- Representations transfer JSON or XML to represent data objects and attributes.
- Messages use HTTP methods explicitly (for example, GET, POST, PUT, and DELETE).
- Stateless interactions store no client context on the server between requests. State dependencies limit and restrict scalability. The client holds session state.

RESTless example with file upload :)



Resources

- https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html
- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc

Home work :)

1. Do analysis of **UploadContoller**, **FileRestController, FileStorageProperty and FileStorageService**. Write down all methods and annotations with all explanations. You need to know how to explain how **FileRestController, UploadContolle, FileStoratgeProperty and FileStorageService** works
2. Write UserInfoRestController which produce such information on requests:
   http://localhost:8080/user-info?name=Jonas&surname=Jonaitis&age=80 (GET method)
   Must return:

   {
   name: “Jonas”,
   surname: “Jonaitis”,
   age: 80
   }

   http://localhost:8080/user-info(POST method). You should pass json **{ name: “Jonas”, surname: “Jonaitis”, age: 80 }** by using RequestBody
   Must return:

   {
   name: “JonasJonas”,
   surname: “JonaitisJonaitis”,
   age: 160
   }

3. Do all steps and run project that you can find on https://spring.io/guides/gs/rest-service/ web page.