Internet Applications Design and Implementation

2016 - 2017 - 2nd edition

(Lecture 7 - Server side programming)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

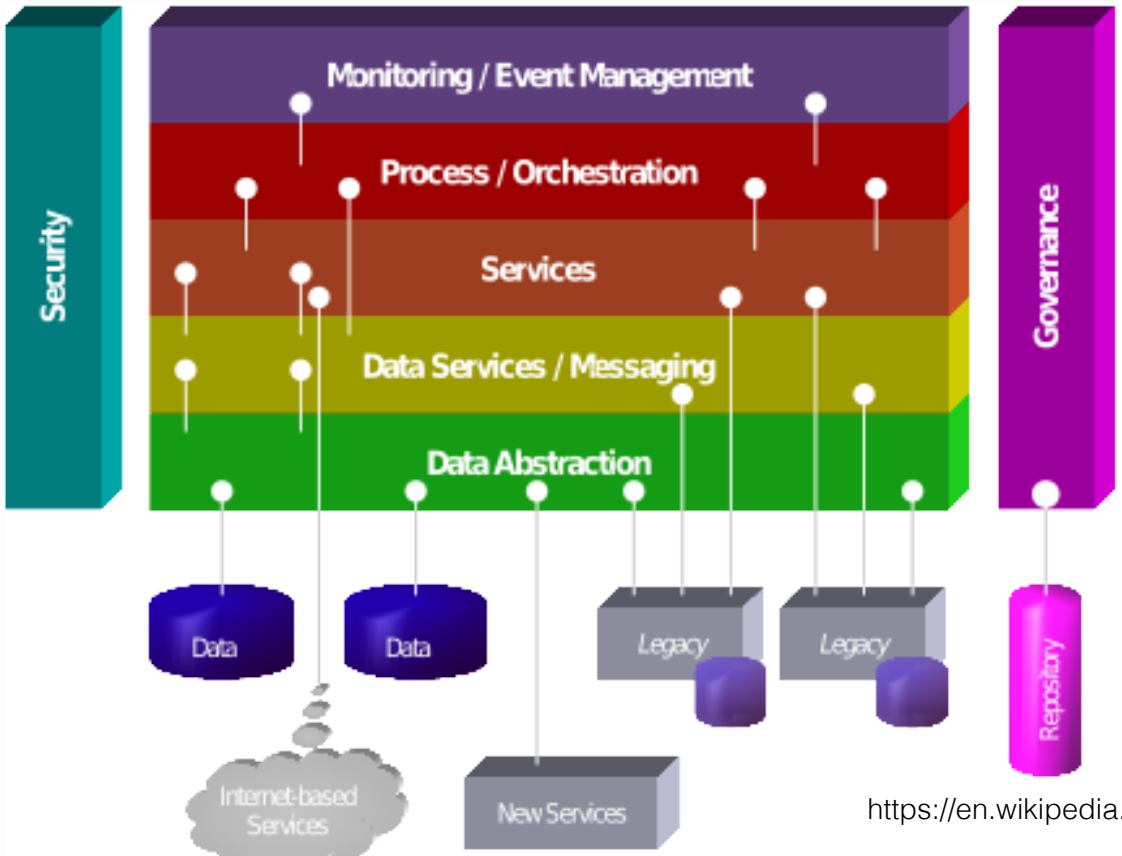
Lecture Plan (draft)

Week	Lecture	Lab
1	Introduction. Software Architectures. Web development frameworks.	-
2	Client technologies (HTML, CSS)	
3	Client application technologies (JS + JQuery)	
4	Reactive Client applications (ReactJS)	
5	Specification of client internet apps (IFML)	
6	Implementation of client internet apps (IFML+REACT)	
7	Process languages and Service-Oriented architectures	#project1
8	Data abstractions (JDBC, JPA, ORM, …) - part I	#midterm
9	Data abstractions - part II	
10	Performance and scalability	
11	Authentication and security models	
12	Specification and implementation of security policies	
13	Language based tools for web applications	#projectfinal
14	…	#final

Connecting Applications
Services and Processes

(web & local) Services

- Service oriented architectures are another way of decoupling implementation from use.
- Services are implemented based on a “contract” or interface and provided by a broker.



https://en.wikipedia.org/wiki/Service-oriented_architecture

Service Orchestration Languages

- Spring Web Flow

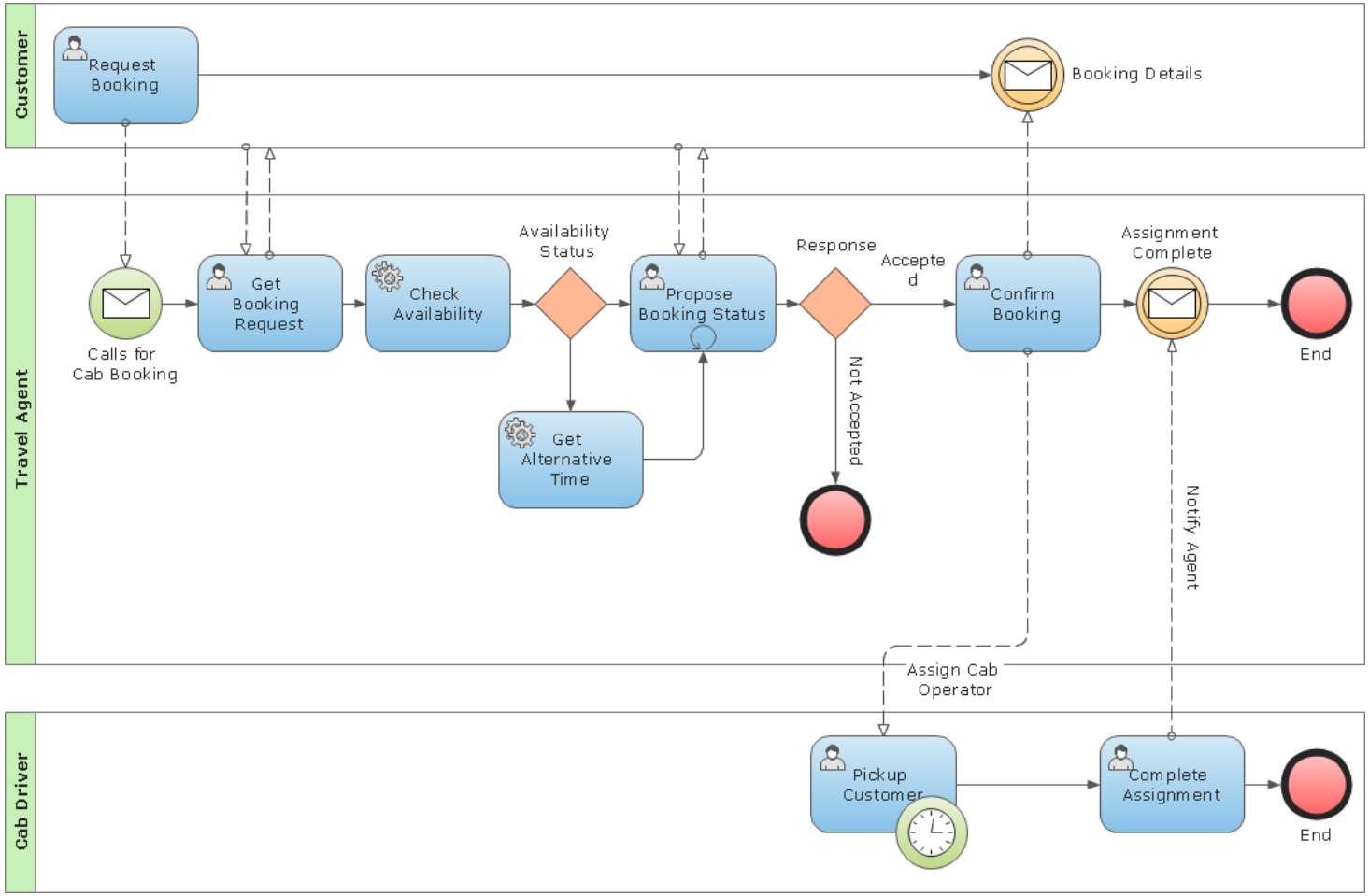
```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
                          http://www.springframework.org/schema/webflow/spring-
webflow-2.0.xsd"


        <start-state="welcome">

        <view-state id="welcome" view="/welcome.jsp">
            <transition on="continue" to="finish"/>
            <transition on="cancel" to="cancelled"/>
```

Process Specification Languages

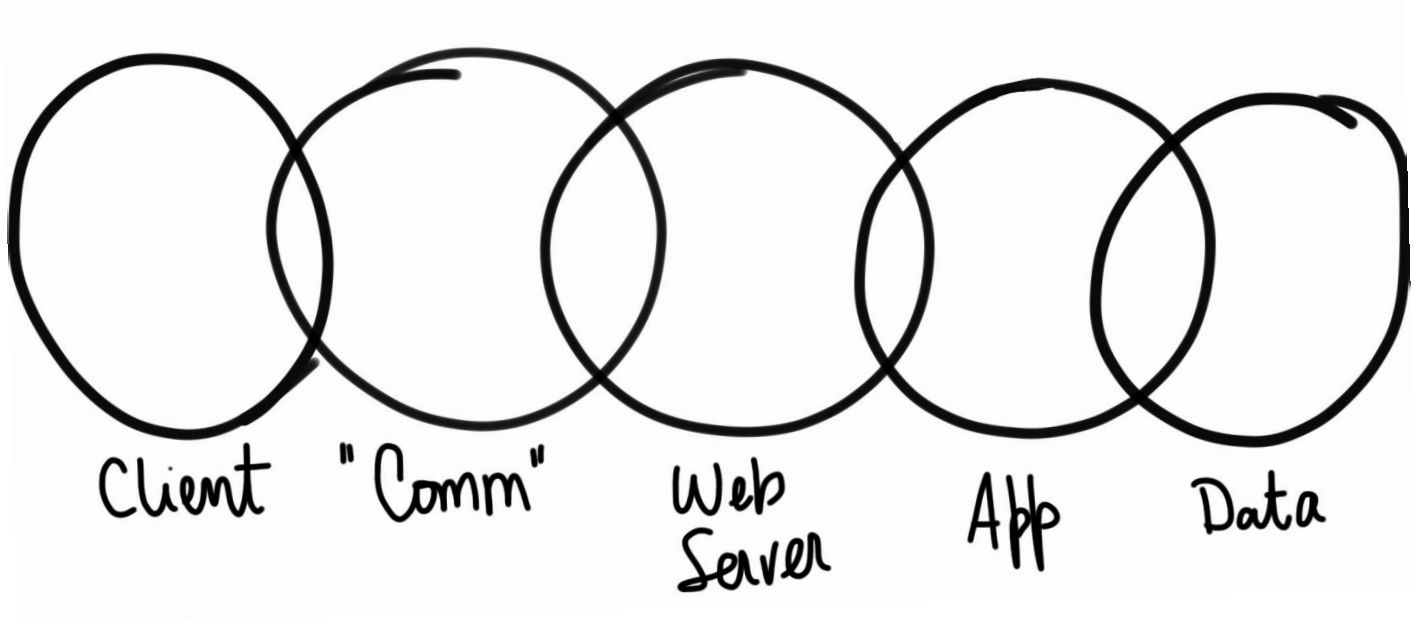
- From processes to code…



Developing Internet Applications
MVC Architecture

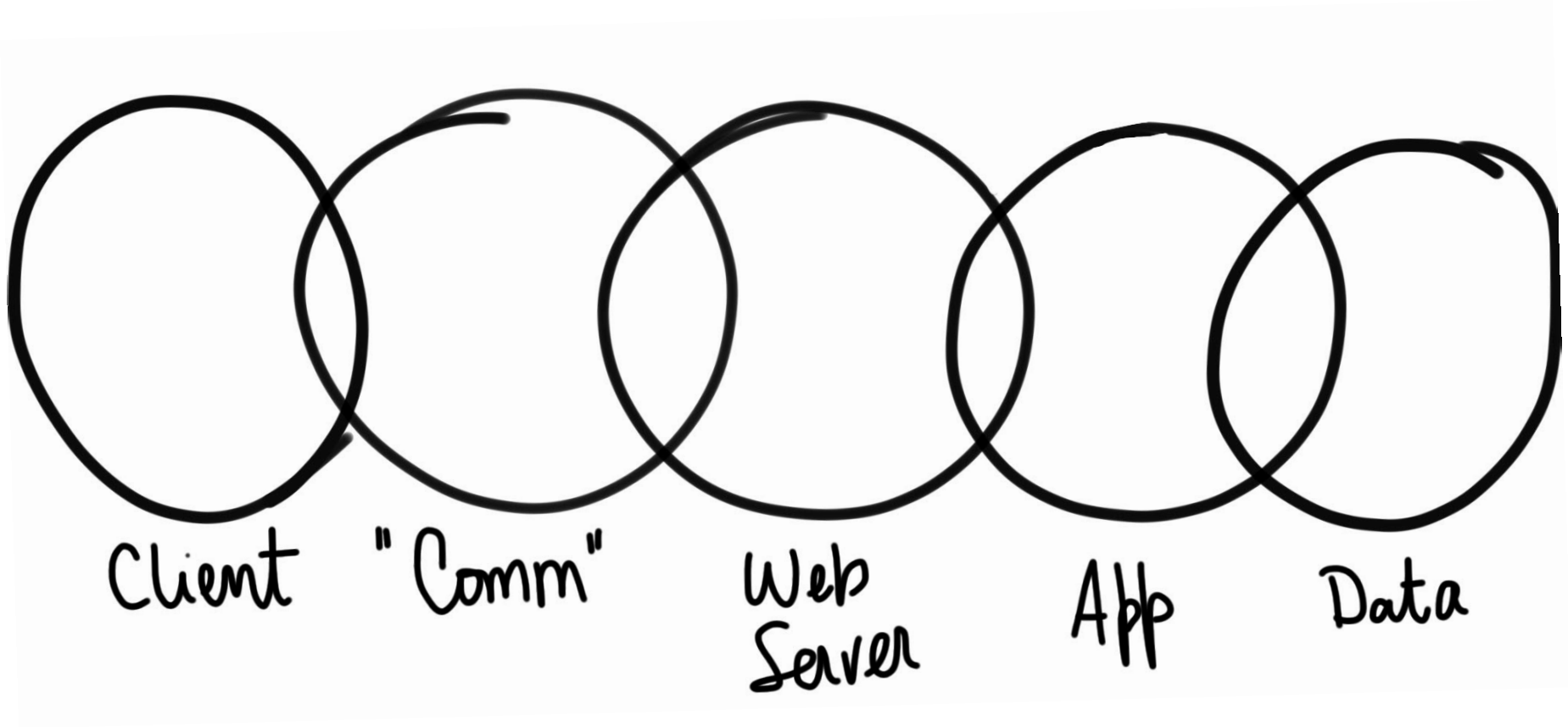
Web architectures, patterns and styles

- Most common web applications follow the MVC architectural pattern.
  - Model layer - isolate the representation of persistent data and its operations, validations and conditions
  - Controller - contains the core application logic implementing the application interface (e.g. ad-hoc URL mapping, REST convention)
  - View - defines the way in which responses are formed (e.g. HTML, JSON)



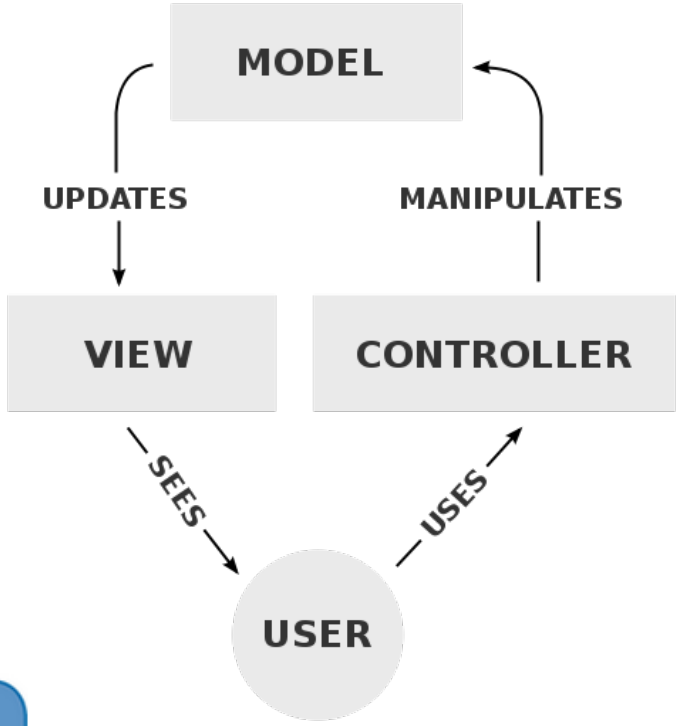
Summary - Web Frameworks

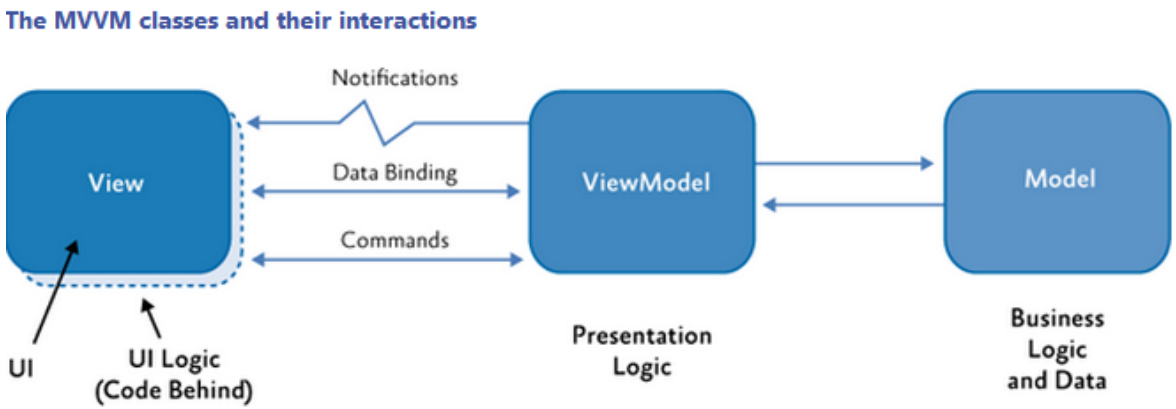
- Web Frameworks are “languages” that carry libraries and abstractions that get compiled to run on the “web virtual machine”.



The classic MVC design pattern

- The Model-View-Controller (Reenskaug'79, JOT'88)
  - designed to develop GUI
  - popular in web applications' context
- Variants of the MVC Architecture (Separation of Concerns)
  - MVP, PM (Fowler), MVVM (Microsoft)

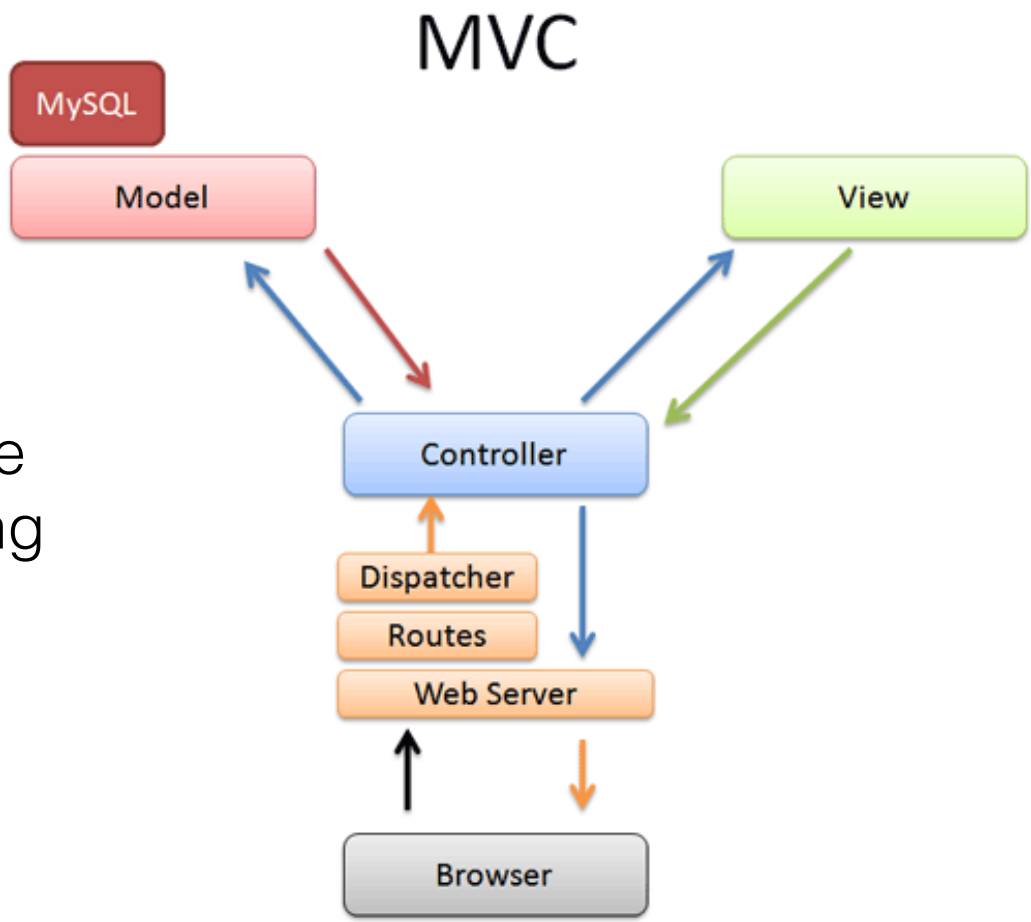




https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/

Frameworks and MVC Architecture

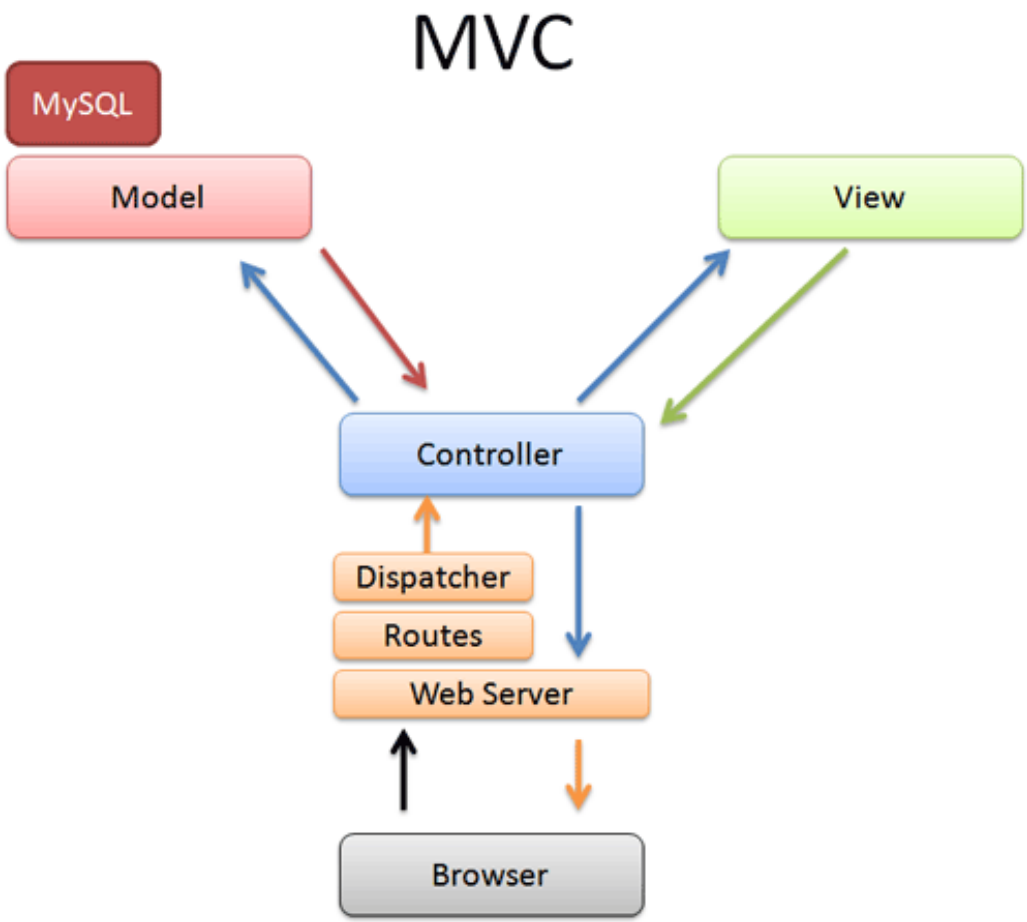
- Frameworks help to implement and maintain architectures.
- Rails (2005):
  - conventions on folder, file, and class names
  - A flexible OO prog language (Ruby) supports data sharing between model, controller, and view objects.



https://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Frameworks and MVC Architecture

- Frameworks help to implement and maintain architectures.
- Django (2005):
  - views are controllers
  - templates are views
  - models are models



https://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and let the components implement the “logic” of applications.
- How spring implements the MVC
  - Dependency Injection (inversion of control)
  - Aspect-Oriented Programming including Spring's declarative transaction management
  - Spring MVC web application and RESTful web service framework
  - Foundational support for JDBC, JPA, JMS
  - …

https://spring.io/guides/

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@SpringBootApplication
public class Application {

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

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@Configuration
@EnableAutoConfiguration
@EnableWebMvc
@ComponentScan
public class Application {

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

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@Controller
public class GreetingController {

    private static final String template = "Hello, %s! %d”;
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name,
                             Model model) {
        long c = counter.incrementAndGet();
        model.addAttribute(“message”, String.format(template, name, c));
        return “greeting”;
    }
}
```

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {
        return new Greeting(counter.incrementAndGet(),
                            String.format(template, name));
    }
}
```

Add-ons to the MVC Framework

- Resource control: DB connection & transactions

```
@Component
public class BookingService {

    private final static Logger logger = LoggerFactory.getLogger(BookingService.class);

    private final JdbcTemplate jdbcTemplate;

    public BookingService(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Transactional
    public void book(String... persons) {
        for (String person : persons) {
            logger.info("Booking " + person + " in a seat...");
            jdbcTemplate.update("insert into BOOKINGS(FIRST_NAME) values (?)", person);
        }
    }

    public List<String> findAllBookings() {
        return jdbcTemplate.query("select FIRST_NAME from BOOKINGS",
                (rs, rowNum) -> rs.getString("FIRST_NAME"));
    }

}
```

Add-ons to the MVC Framework

- Across application concerns: security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("user").password("password").roles("USER");
    }
}
```

Connecting Applications
REST Services

Restful interface design (Recap)

- Follows an architectural style (convention)
  - Architectural style that promotes a simpler and more efficient way of providing and connecting web services. Built on top of basic HTTP
- Promotes the decoupling from Data-centric server side applications and client user-centric applications
- Implementations provides (convenient) flavours
  - Web-service style pure JSON/XML Data
  - Complete/partial HTML view responses
  - Javascript code responses (e.g. Rails AJAX responses)

REST - Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (optional)

Representational State Transfer

- Resource Based
  - vs Action Based
  - Nouns and not verbs to identify data in the system
  - Identified (represented) by URI
  - Aliasing is admissible
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
  - JSON or XML representation of the state of a given resource transferred between client and server at a given verb in a given URL.
  - Well identified interface (the information retrieved at an URL — the type)
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
  - standard HTTP verbs (GET, PUT, POST, DELETE)
  - standard HTTP response (status code, info in the response body)
  - Uniform structure of URIs with a name, identifying the resource
  - References inside responses must be complete.
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
  - Server does not hold session state
  - Messages are self contained
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
  - Responses can be tagged as cacheable (in the server)
  - (also) Bookmarkable
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Layered System
  - Establishes an API between a client and a “database”
- Code on Demand (not talking about it)

EXAMPLES

6. Real REST Examples

Here's a very partial list of service providers that use a REST API. Note that some of them also support a WSDL (Web Services) API, in addition, so you can pick which to use; but in most cases, when both alternatives are available, REST calls are easier to create, the results are easier to parse and use, and it's also less resource-heavy on your system.

So without further ado, some REST services:

- The Google Glass API, known as "Mirror API", is a pure REST API. Here is an excellent video talk about this API. (The actual API discussion starts after 16 minutes or so.)
- Twitter has a **REST API** (in fact, this was their original API and, so far as I can tell, it's still the main API used by Twitter application developers),
- **Flickr**,
- **Amazon.com** offer several REST services, e.g., for their S3 storage solution,
- **Atom** is a RESTful alternative to RSS,
- **Tesla Model S** uses an (undocumented) REST API between the car systems and its Android/iOS apps.

in … http://rest.elkstein.org/2008/02/real-rest-examples.html

Mirror API - Google Glasses

Contacts

For Contacts Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/mirror/v1, unless otherwise noted		
delete	`DELETE /contacts/id`	Deletes a contact.
get	`GET /contacts/id`	Gets a single contact by ID.
insert	`POST /contacts`	Inserts a new contact.
list	`GET /contacts`	Retrieves a list of contacts for the authenticated user.
patch	`PATCH /contacts/id`	Updates a contact in place. This method supports patch semantics.
update	`PUT /contacts/id`	Updates a contact in place.

in … https://developers.google.com/glass/v1/reference/

Mirror API - Google Glasses

Timeline

For Timeline Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/mirror/v1, unless otherwise noted		
delete	`DELETE /timeline/id`	Deletes a timeline item.
get	`GET /timeline/id`	Gets a single timeline item by ID.
insert	`POST https://www.googleapis.com/upload/mirror/v1/timeline` and `POST /timeline`	Inserts a new item into the timeline.
list	`GET /timeline`	Retrieves a list of timeline items for the authenticated user.
patch	`PATCH /timeline/id`	Updates a timeline item in place. This method supports patch semantics.
update	`PUT https://www.googleapis.com/upload/mirror/v1/timeline/id` and `PUT /timeline/id`	Updates a timeline item in place.

Mirror API - Google Glasses

Timeline.attachments

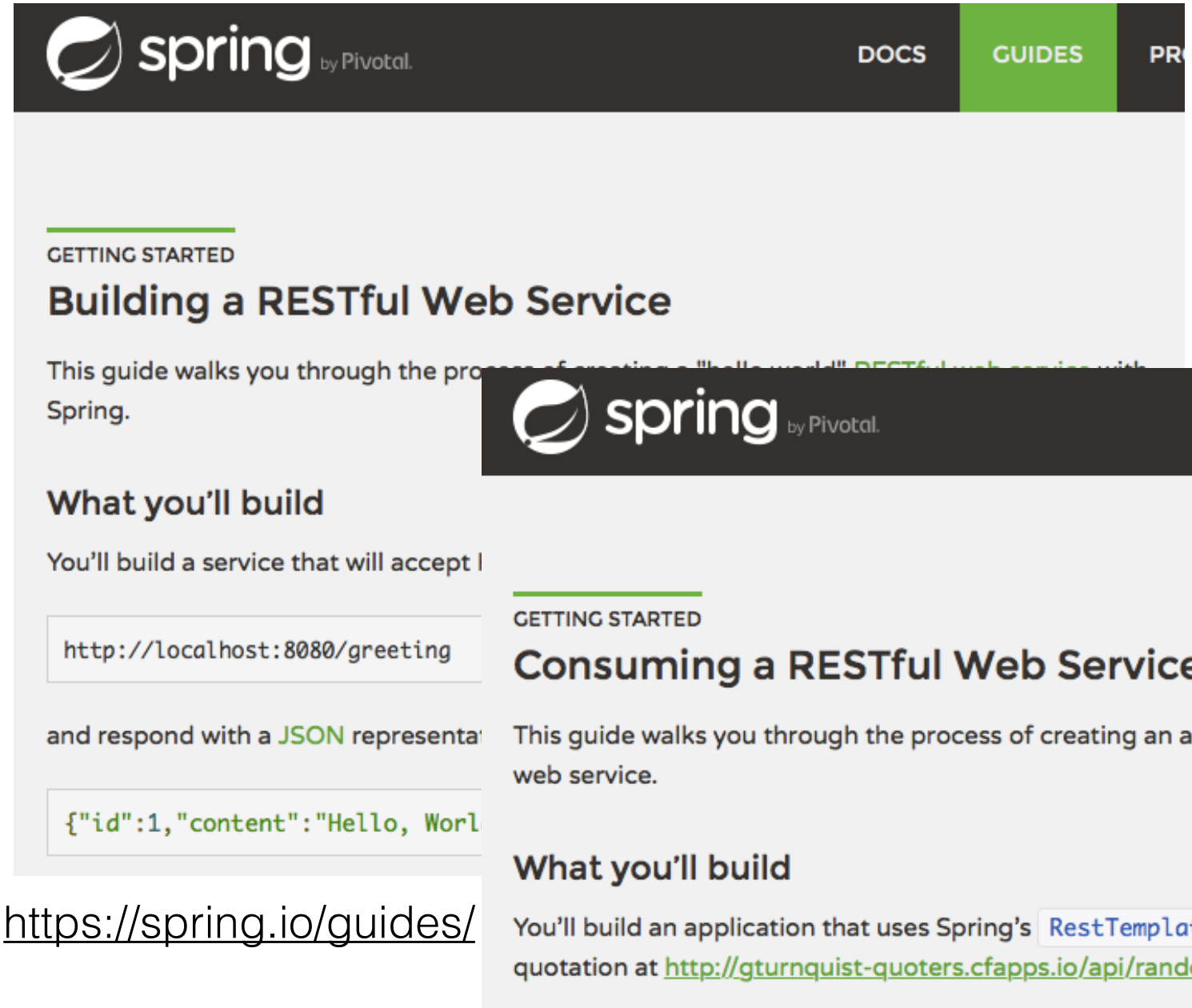
For Timeline.attachments Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/mirror/v1, unless otherwise noted		
delete	`DELETE /timeline/itemId/attachments/attachmentId`	Deletes an attachment from a timeline item.
get	`GET /timeline/itemId/attachments/attachmentId`	Retrieves an attachment on a timeline item by item ID and attachment ID.
insert	`POST https://www.googleapis.com/upload/mirror/v1/timeline/itemId/attachments`	Adds a new attachment to a timeline item.
list	`GET /timeline/itemId/attachments`	Returns a list of attachments for a timeline item.

in … https://developers.google.com/glass/v1/reference/

Connecting Applications
Rest in Spring

Rest in Spring



https://spring.io/guides/