Θέματα Αρχιτεκτονικής Web Εφαρμογών

Ανάπτυξη Διαδικτυακών Συστημάτων & Εφαρμογών

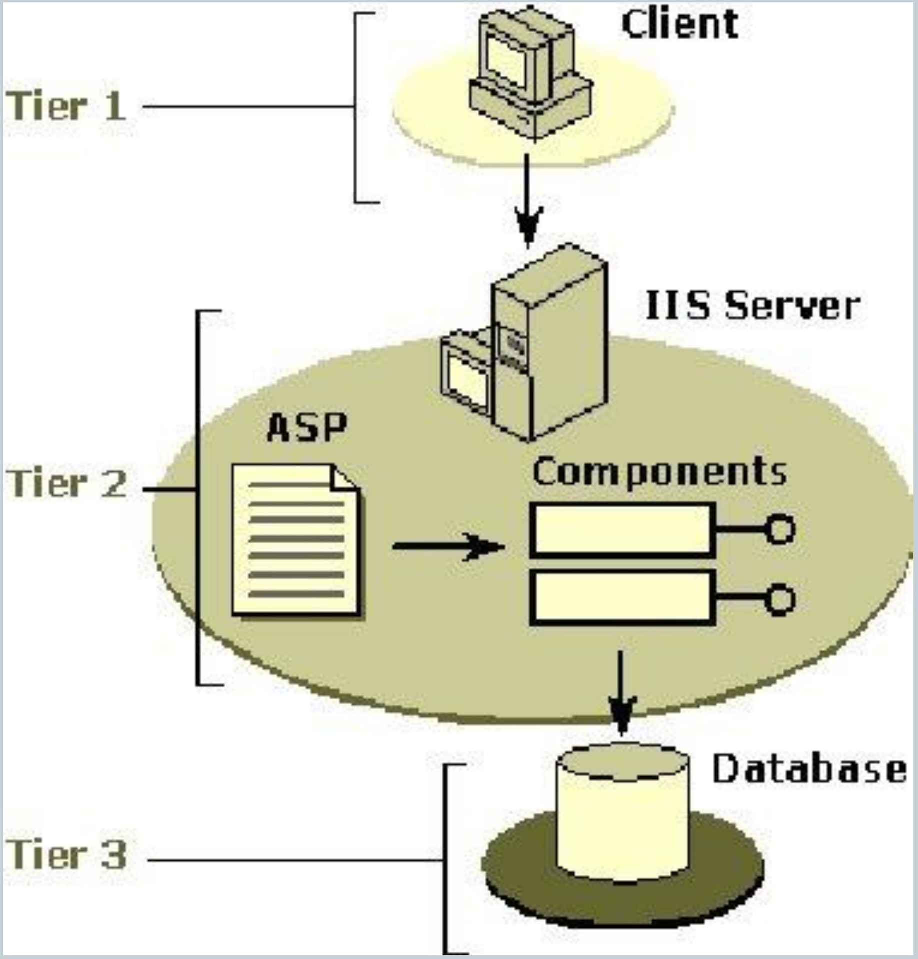
Τμ. Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

ΔιΠαΕ

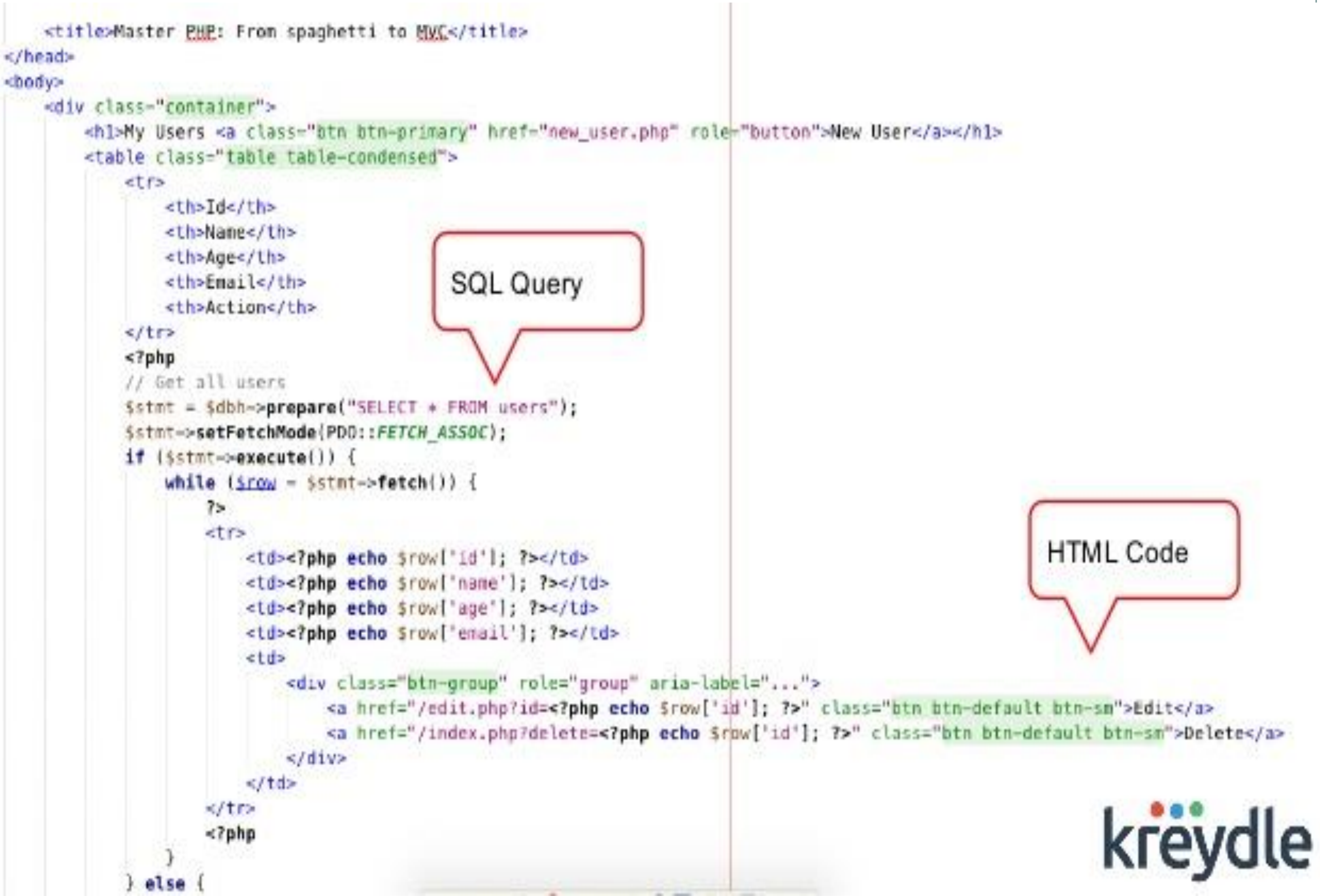
Αντώνης Σιδηρόπουλος

Talking Productivity

Step 1: Separate Business Logic

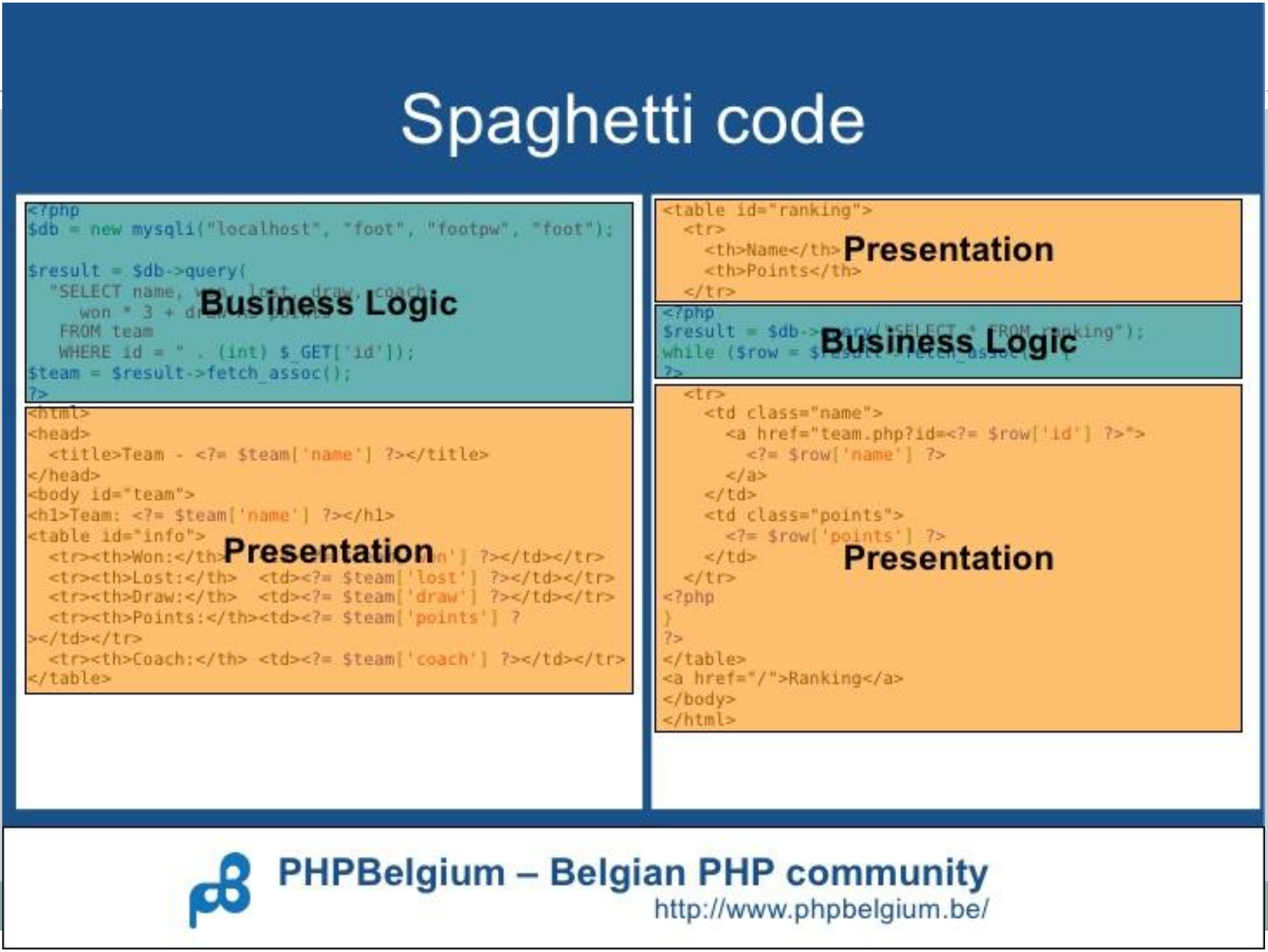


Προβλήματα – Spaggeti code



```
    <title>Master PHP: From spaghetti to MVC</title>
</head>
<body>
    <div class="container">
        <h1>My Users <a class="btn btn-primary" href="new_user.php" role="button">New User</a></h1>
        <table class="table table-condensed">
            <tr>
                <th>Id</th>
                <th>Name</th>
                <th>Age</th>
                <th>Email</th>
                <th>Action</th>
            </tr>
            <?php
            // Get all users
            $stmt = $dbh->prepare("SELECT * FROM users");
            $stmt->setFetchMode(PDO::FETCH_ASSOC);
            if ($stmt->execute()) {
                while ($row = $stmt->fetch()) {
                    ?>
                    <tr>
                        <td><?php echo $row['id']; ?></td>
                        <td><?php echo $row['name']; ?></td>
                        <td><?php echo $row['age']; ?></td>
                        <td><?php echo $row['email']; ?></td>
                        <td>
                            <div class="btn-group" role="group" aria-label="...">
                                <a href="/edit.php?id=<?php echo $row['id']; ?>" class="btn btn-default btn-sm">Edit</a>
                                <a href="/index.php?delete=<?php echo $row['id']; ?>" class="btn btn-default btn-sm">Delete</a>
                            </div>
                        </td>
                    </tr>
                    <?php
                }
            } else {
```

Προβλήματα – Spaggeti code



```
<?php
$db = new mysqli("localhost", "foot", "footpw", "foot");

$result = $db->query(
  "SELECT name, won, lost, draw, coach,
     won * 3 + draw AS points
   FROM team
   WHERE id = " . (int) $_GET['id']);
$team = $result->fetch_assoc();
?>
<html>
<head>
  <title>Team - <?= $team['name'] ?></title>
</head>
<body id="team">
<h1>Team: <?= $team['name'] ?></h1>
<table id="info">
  <tr><th>Won:</th>[illegible]n'] ?></td></tr>
  <tr><th>Lost:</th>  <td><?= $team['lost'] ?></td></tr>
  <tr><th>Draw:</th>  <td><?= $team['draw'] ?></td></tr>
  <tr><th>Points:</th><td><?= $team['points'] ?
></td></tr>
  <tr><th>Coach:</th> <td><?= $team['coach'] ?></td></tr>
</table>
```

```
<table id="ranking">
  <tr>
    <th>Name</th>
    <th>Points</th>
  </tr>
<?php
$result = $db->query("SELECT * FROM ranking");
while ($row = $result->fetch_assoc()) {
?>
  <tr>
    <td class="name">
      <a href="team.php?id=<?= $row['id'] ?>">
        <?= $row['name'] ?>
      </a>
    </td>
    <td class="points">
      <?= $row['points'] ?>
    </td>
  </tr>
<?php
}
?>
</table>
<a href="/">Ranking</a>
</body>
</html>
```

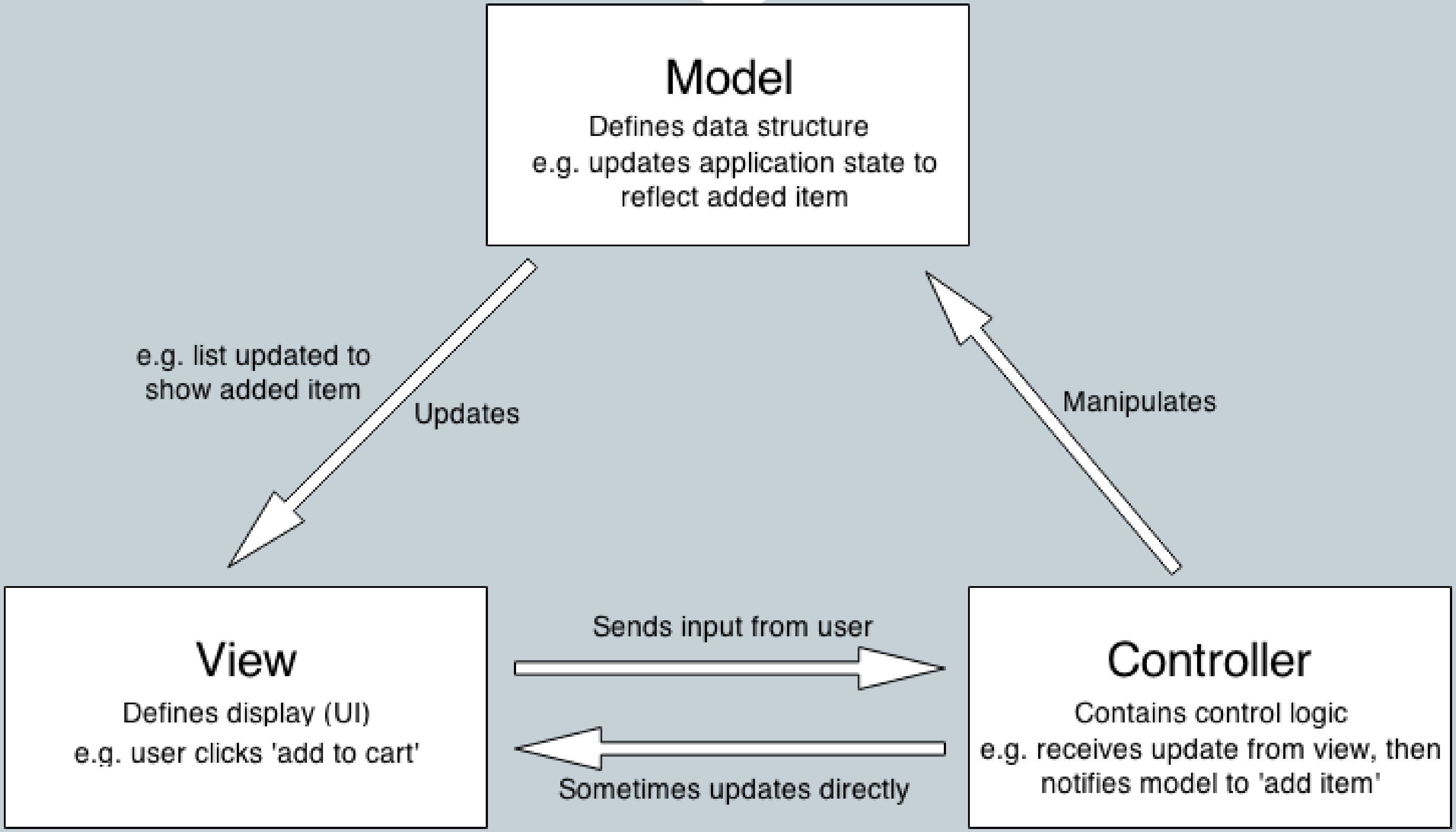
“Classic” Web Technologies

- Classic ASP is one of the web technology introduced by Microsoft.
- Biggest pain point with the classic ASP was spaghetti code and maintainability.
- Same is PHP, JSP
- **You know what’s the biggest problem with this scenario?**
  - You have to do lots of stuffs by your own.

Model View Controller (MVC)

- is a software architecture pattern, commonly used to implement user interfaces.
- it is therefore a popular choice for architecting web apps.
- In general, it separates out the application logic into three separate parts, promoting modularity and ease of collaboration and reuse.

MVC


Model
Defines data structure
e.g. updates application state to reflect added item
e.g. list updated to show added item
Updates
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view, then notifies model to 'add item'

The Model

- The model defines what data the app should contain. If the state of this data changes, then the model will usually notify the view (so the display can change as needed) and sometimes the controller (if different logic is needed to control the updated view).

The View

- The view defines how the app's data should be displayed.

The Controller

- The controller contains logic that updates the model and/or view in response to input from the users of the app.
- So for example, a shopping application could have input forms and buttons that allow us to add or delete items. These actions require the model to be updated, so the input is sent to the controller, which then manipulates the model as appropriate, which then sends updated data to the view.
- You might however also want to just update the view to display the data in a different format, e.g., change the item order to alphabetical, or lowest to highest price. In this case the controller could handle this directly without needing to update the model.

Application programming interface (API)

- is a set of subroutine definitions, protocols, and tools for building application software.
- In general terms, it is a set of clearly defined methods of communication between various software components.
- A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.
- An API may be for a web-based system, operating system, database system, computer hardware or software library.

APIs

- Libraries and frameworks
  - An API is usually related to a software library. The API describes and prescribes the expected behavior (a specification) while the library is an actual implementation of this set of rules.
- Operating systems
  - An API can specify the interface between an application and the operating system (e.g. POSIX)
- Remote APIs
  - Remote APIs allow developers to manipulate remote resources through protocols, specific standards for communication that allow different technologies to work together, regardless of language or platform. For example, the Java Database Connectivity API.
- Web APIs
  - APIs with interaction through the web (HTTP)

Web API

- Συνήθως χρησιμοποιείται στην πλευρά του “Model”. Έτσι έχουμε καλύτερο διαχωρισμό των στοιχείων MVC.

Representational State Transfer (REST)

- Representational state transfer (REST) or RESTful web services is a way of providing interoperability between computer systems on the Internet.

- REST + API = REST API

Resource

```
{    "ID": "1",
    "Name": "A Sidiropoulos",
    "Email": "asidirop@gmail.com",
    "Country": "Greece"
}
```

JSON representation of a resource.

```
<Person>
<ID>1</ID>
<Name>A Sidiropoulos</Name>
<Email>asidirop@gmail.com</Email>
<Country>Greece</Country>
</Person>
```

XML representation of a resource.

Addressing Resources (διευθυνσιοδότηση πόρων)

- REST απαιτεί κάθε πόρος να έχει (αντιστοιχεί) σε ένα τουλάχιστον URI.
- Μια υπηρεσία RESTful χρησιμοποιεί μια ιεραρχία φακέλων για την διευθυνσιοδότηση των πόρων.
- Η χρήση του URI είναι να διευθυνσιοδοτεί έναν πόρο ή μια συλλογή από πόρους.
- Η ενέργεια ορίζεται από ένα HTTP verb (HTTP ρήμα). Το URI δεν πρέπει να περιέχει πληροφορία για την ενέργεια επάνω στον πόρο.
- Έτσι μπορούμε να χρησιμοποιούμε το ίδιο URI με διαφορετικές ενέργειες μέσω του HTTP.

- Έστω ότι έχουμε μια βάση από Persons και θέλουμε να την «δημοσιοποιήσουμε» μέσα από ένα Web Service. Για κάθε Person (πόρος) μπορεί να αντιστοιχηθεί ένα URI:

  http://MyService/Persons/1

- Το URI έχει την μορφή: Protocol://ServiceName/ResourceType/ResourceID

Query Parameters in URI

Τα προηγούμενα URI μπορούμε να τα εμπλουτίσουμε με ορίσματα:

http://MyService/Persons?id=1

ή

http://MyService/Persons/1?format=xml&encoding=UTF8

ή

http://MyService/Persons/1?format=json&encoding=UTF8

Ή να ενσωματώσουμε τα ορίσματα στην διαδρομή μέσα στο URI :

http://MyService/Persons/1/json/UTF8

Uniform Interface (ομοιόμορφη διεπαφή)

Τα συστήματα RESTful πρέπει να έχουν ομοιόμορφη διεπαφή. Το HTTP 1.1 παρέχει ένα σύνολο μεθόδων (called verbs), για αυτόν τον σκοπό. Τα περισσότερο συχνά είναι:

Method	**Operation performed on server**
GET	Read a resource.
PUT	Insert a new resource or update if the resource already exists.
POST	Insert a new resource. Also can be used to update an existing resource.
DELETE	Delete a resource .
OPTIONS	List the allowed operations on a resource.
HEAD	Return only the response headers and no response body.

HTTP Requests

- Παράδειγμα HTTP αιτήσεων:

~~GET http://MyService/DeletePersons/1 HTTP/1.1~~
~~HOST: MyService~~

```
DELETE http://MyService/Persons/1 HTTP/1.1
HOST: MyService
```

API

- Το σύνολο των resources της εφαρμογής μας αποτελεί το API.

Resource	Methods	URI	Description
Person	`GET, POST, PUT, DELETE`	`http://MyService/Persons/{PersonID}`	Contains information about a person {PersonID} is optional Format: text/xml
Club	`GET, POST, PUT`	`http://MyService/Clubs/{ClubID}`	Contains information about a club. A club can be joined my multiple people {ClubID} is optional Format: text/xml
Search	`GET`	`http://MyService/Search?`	Search a person or a club Format: text/xml Query Parameters: Name: String, Name of a person or a club Country: String, optional, Name of the country of a person or a club Type: String, optional, Person or Club. If not provided then search will result in both Person and Cubs

10 κανόνες για την δημιουργία REST API

1. Use nouns but no verbs
2. GET method and query parameters should not alter the state
3. Use plural nouns
4. Use sub-resources for relations
5. Use HTTP headers for serialization formats
6. Use HATEOAS
7. Provide filtering, sorting, field selection and paging for collections
8. Version your API
9. Handle Errors with HTTP status codes
10. Allow overriding HTTP method

https://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/

1. Use nouns but no verbs

Resource	GET read	POST create	PUT update	DELETE
/cars	Returns a list of cars	Create a new car	Bulk update of cars	Delete all cars
/cars/711	Returns a specific car	Method not allowed (405)	Updates a specific car	Deletes a specific car

Do not use verbs:

/getAllCars
/createNewCar
/deleteAllRedCars

2. GET method and query parameters should not alter the state

- Use PUT, POST and DELETE methods instead of the GET method to alter the state.
- Do not use GET for state changes:

  GET /users/711?activate or

  GET /users/711/activate

3. Use plural nouns

- Do not mix up singular and plural nouns. Keep it simple and use only plural nouns for all resources.

/cars instead of /car

/users instead of /user

/products instead of /product

/settings instead of /setting

4. Use sub-resources for relations

- If a resource is related to another resource use subresources.

```
GET /cars/711/drivers/
     Returns a list of drivers for car 711

GET /cars/711/drivers/4
     Returns driver #4 for car 711
```

5. Use HTTP headers for serialization formats

- Both, client and server, need to know which format is used for the communication. The format has to be specified in the HTTP-Header.

- Content-Type defines the request format. (το στέλνει ο server)
- Accept defines a list of acceptable response formats. (το στέλνει ο client μέσα στο request)

6. Use HATEOAS

- **H**ypermedia **as** **the** **E**ngine **of** **A**pplication **S**tate is a principle that hypertext links should be used to create a better navigation through the API.

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
    <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
    <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
    <link rel="close" href="https://bank.example.com/accounts/12345/close" />
</account>
```

6. Use HATEOAS (b)

- Η ίδια αίτηση σε επόμενο χρόνο…
- στην περίπτωση που έχει κλειδώσει ο λογαριασμός η μόνη πράξη που επιτρέπεται είναι «κατάθεση»

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">-25.00</balance>
    <link rel="deposit" href="https://bank.example.com/account/12345/deposit" />
</account>
```

7. Provide filtering, sorting, field selection and paging for collections (a)

Filtering:

Use a unique query parameter for all fields or a query language for filtering.

```
GET /cars?color=red
```

Returns a list of red cars

```
GET /cars?seats<=2
```

Returns a list of cars with a maximum of 2 seats

7. Provide filtering, sorting, field selection and paging for collections (b)

Sorting:

Allow ascending and descending sorting over multiple fields.

```
GET /cars?sort=-manufactorer,+model
```

This returns a list of cars sorted by descending manufacturers and ascending models.

7. Provide filtering, sorting, field selection and paging for collections (c)

Field selection

Mobile clients display just a few attributes in a list. They don’t need all attributes of a resource. Give the API consumer the ability to choose returned fields. This will also reduce the network traffic and speed up the usage of the API.

```
GET
/cars?fields=manufacturer,model,id,color
```

7. Provide filtering, sorting, field selection and paging for collections (d)

Paging

Use limit and offset. It is flexible for the user and common in leading databases. The default should be limit=20 and offset=0

```
GET /cars?offset=10&limit=5
```

To send the total entries back to the user use the custom HTTP header: X-Total-Count.

Links to the next or previous page should be provided in the HTTP header link as well. It is important to follow this link header values instead of constructing your own URLs.

```
Link:
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=15&limit=5>;
rel="next",
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=50&limit=5>;
rel="last",
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=0&limit=5>;
rel="first",
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=5&limit=5>;
rel="prev"
```

8. Version your API

Make the API Version mandatory and do not release an unversioned API. Use a simple ordinal number and avoid dot notation such as 2.5.

We are using the url for the API versioning starting with the letter „v“

```
/blog/api/v1
```

9. Handle Errors with HTTP status codes

It is hard to work with an API that ignores error handling. Pure returning of a HTTP 500 with a stacktrace is not very helpful.

The HTTP standard provides over 70 status codes to describe the return values. We don't need them all, but there should be used at least a mount of 10.

- 200 – OK – Eyerything is working
- 201 – OK – New resource has been created
- 204 – OK – The resource was successfully deleted
- 304 – Not Modified – The client can use cached data
- 400 – Bad Request – The request was invalid or cannot be served. The exact error should be explained in the error payload. E.g. „The JSON is not valid“
- 401 – Unauthorized – The request requires an user authentication
- 403 – Forbidden – The server understood the request, but is refusing it or the access is not allowed.
- 404 – Not found – There is no resource behind the URI.
- 422 – Unprocessable Entity – Should be used if the server cannot process the enitity, e.g. if an image cannot be formatted or mandatory fields are missing in the payload.
- 500 – Internal Server Error – API developers should avoid this error. If an error occurs in the global catch blog, the stracktrace should be logged and not returned as response.

9. Handle Errors with HTTP status codes (b)

Use error payloads

All exceptions should be mapped in an error payload. Here is an example how a JSON payload should look like.

```
{  "errors": [
    {
        "userMessage": "Sorry, the requested resource does not
exist",
        "internalMessage": "No car found in the database",
        "code": 34,
        "more info":
        "http://dev.mwaysolutions.com/blog/api/v1/errors/12345"
    } ]
}
```

10. Allow overriding HTTP method

- Some proxies support only POST and GET methods. To support a RESTful API with these limitations, the API needs a way to override the HTTP method.

- Use the custom HTTP Header

```
X-HTTP-Method-Override
```

to overrider the POST Method.

Web API

- Εάν ένα API δεν πληροί κάποιες από τις προηγούμενες προδιαγραφές, τότε δεν είναι REST API. Όμως:
  - Δεν σημαίνει ότι δεν είναι λειτουργικό.
  - Δεν σημαίνει ότι δεν είναι Web API.