JÖNKÖPING UNIVERSITY

School of Engineering

ASP.NET WEB API

Server Side Web Development

TPWK16 Spring 2016

Peter Larsson-Green

JÖNKÖPING UNIVERSITY
School of Engineering

AWS EC2 INSTANCES

Guide:

- How to connect to AWS EC2 instances

API

Application Programming Interface

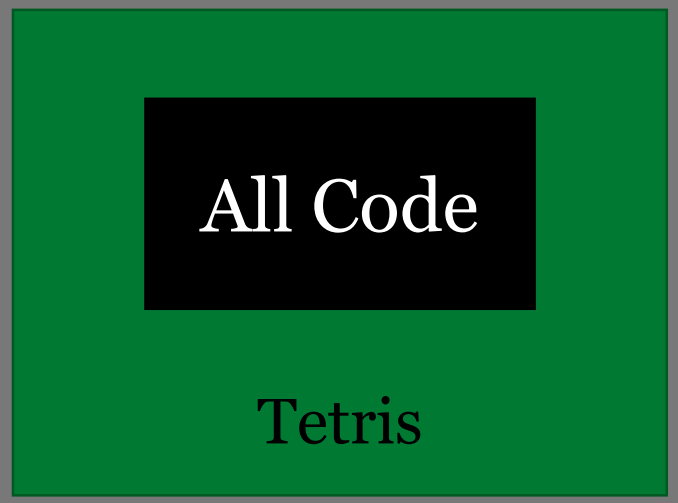
1. Build an application containing the logic.
2. Add interfaces through which it can be controlled.
   - Command Line Interfaces - for nerds.
   - Graphical User Interfaces - for average people.
   - Application Programming Interfaces - for other programs.

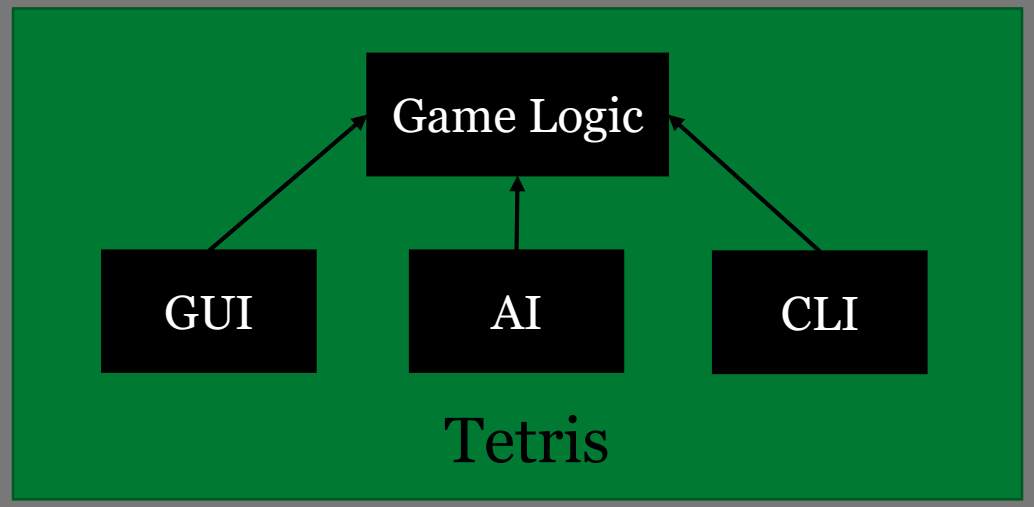
INTERFACES ARE GREAT!

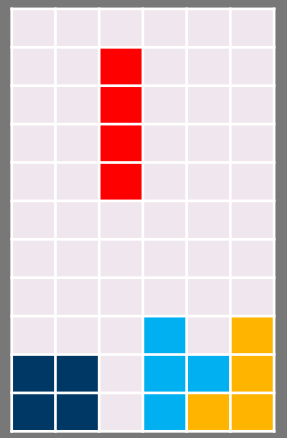
When designing, always try to create modules with well defined interfaces.

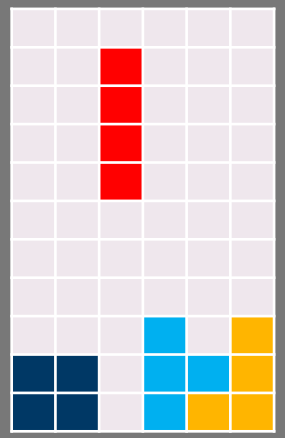
- Your code will be well structured.
- Your code will be reusable.
- Your code will be testable.

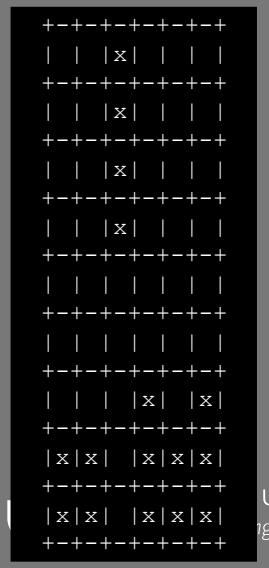
DESIGNING TETRIS











APIS FOR WEB APPLICATIONS

Messages are sent over HTTP.

- Old approach: SOAP - Simple Object Access Protocol.
  - http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example/
- Modern approach: REST - REpresentational State Transfer.
  - Is data centric and builds on HTTP:
    - Use URIs to identify resources.
    - Use the HTTP methods to apply operations on the resources.
      - Create: POST
      - Read: GET
      - Update: PUT
      - Delete: DELETE
  - Is an architectural style, not a specification.

REST EXAMPLE

A server with information about users.

- The GET method is used to retrieve resources.
  - `GET /users`
  - `GET /users/61`
  - `GET /users/pages/1` `GET /users?page=1`
  - `GET /users/gender/female` `GET /users?gender=female`
  - `GET /users/age/18` `GET /users?age=18`
  - `GET /users/???` `GET /users?gender=female&age=18`
  - `GET /users/61/name`
  - `GET /users/61/pets`

REST EXAMPLE

A server with information about users.

- The GET method is used to retrieve resources.
  - Which data format?
    - Specified in the `Accept` header!

```
GET /users HTTP/1.1
Host: the-website.com
Accept: application/json
```

`application/xml` was popular before JSON.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 66

[
  {"id": 1, "name": "Human A"},
  {"id": 2, "name": "Human B"}
]
```

REST EXAMPLE

A server with information about users.

- The POST method is used to create resources.
  - Which data format? Specified in the `Accept` and `Content-Type` header!

```
POST /users HTTP/1.1
Host: the-website.com
Accept: application/json
Content-Type: application/xml
Content-Length: 40

<human>
  <name>Human C</name>
</human>
```

```
HTTP/1.1 201 Created
Location: /users/3
Content-Type: application/json
Content-Length: 28

{"id": 3, "name": "Human C"}
```

REST EXAMPLE

A server with information about users.

- The PUT method is used to update an entire resource.

```
PUT /users/2 HTTP/1.1
Host: the-website.com
Content-Type: application/xml
Content-Length: 53

<human>
  <id>2</id>
  <name>Human X</name>
</human>
```

```
HTTP/1.1 204 No Content
```

PUT can also be used to create a resource if you know which URI is should have in advance.

REST EXAMPLE

A server with information about users.

- The DELETE method is used to delete a resource.

```
DELETE /users/2 HTTP/1.1
Host: the-website.com
```

```
HTTP/1.1 204 No Content
```

REST EXAMPLE

A server with information about users.

- The PATCH method is used to update parts of a resource.

```
PATCH /users/1 HTTP/1.1
Host: the-website.com
Content-Type: application/xml
Content-Length: 40

<human>
  <name>Human X</human>
</human>
```

```
HTTP/1.1 204 No Content
```

The PATCH method is only a proposed standard.

REST EXAMPLE

A server with information about users.

- What if something goes wrong?
  - Use the HTTP status codes!

```
GET /users/999 HTTP/1.1
Host: the-website.com
Accept: application/json
```

```
HTTP/1.1 404 Not Found
```

  - Read more about the different status codes at:
    - http://www.restapitutorial.com/httpstatuscodes.html
  - Optionally include error messages in the response body.

DESIGNING A REST API

How should you think?

- Make it as easy to use as possible.

Facebook:

- Always return 200 OK.
- `GET /v2.7/{user-id}`
- `GET /v2.7/{post-id}`
- `GET /v2.7/{user-id}/friends`
- `GET /v2.7/{object-id}/likes`

DESIGNING A REST API

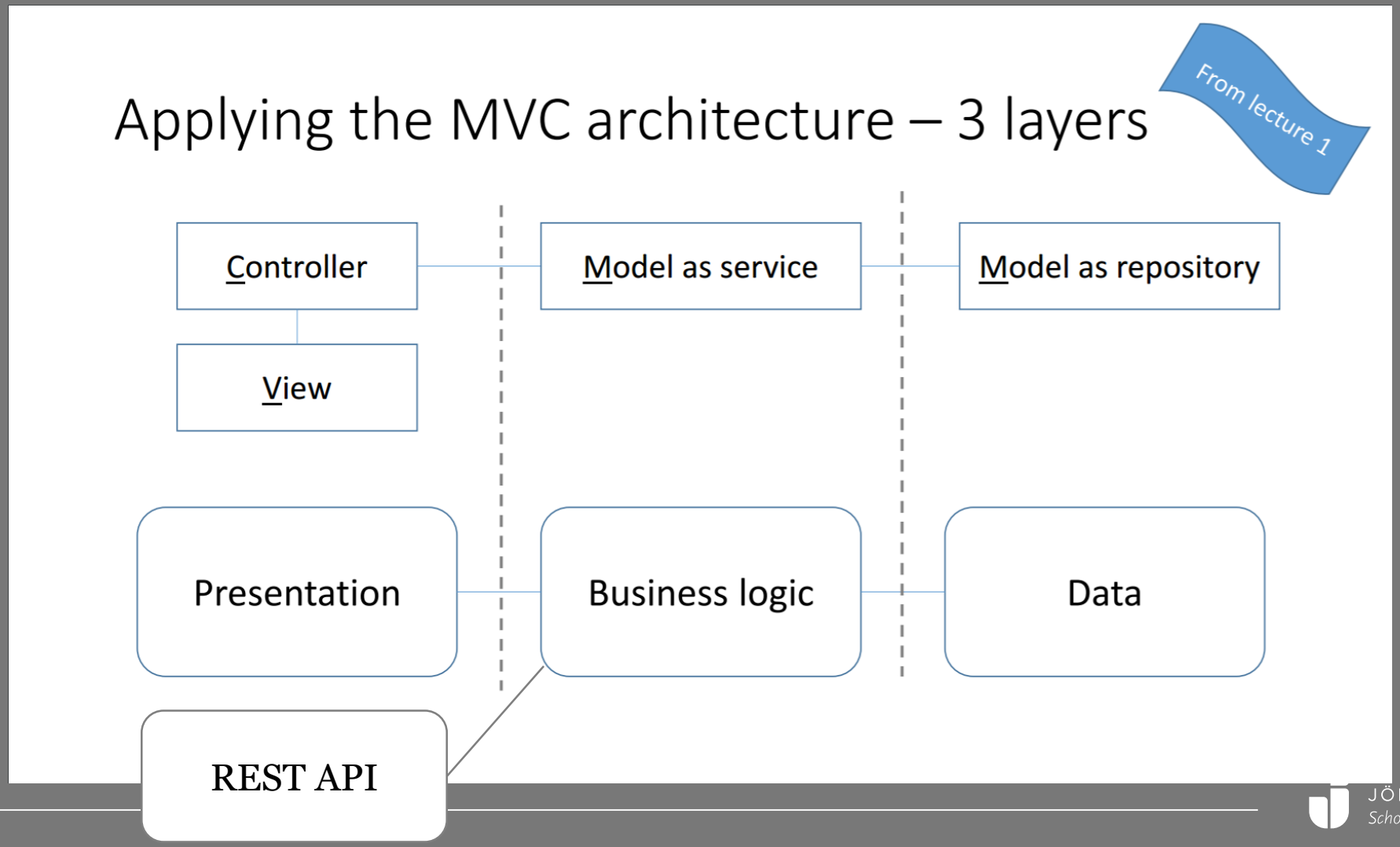
How should you think?

- Make it as easy to use as possible.

Twitter:

- Only use GET and POST.
- `GET  /1.1/users/show.json?user_id=2244994945`
- `POST /1.1/favorites/destroy.json?id=243138128959913986`

THREE LAYERED ARCHITECTURE



WEB API CONFIGURATION

```
GlobalConfiguration.Configure(config => {
  config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
  );
});
```

`{controller}Controller` will be used.

MAPPING METHODS 1

```
public class HumansController : ApiController{
  [HttpGet]
  public HttpResponseMessage ReturnListOfHumans(){
    // Code handling GET requests for /api/Humans.
  }
  [HttpPost]
  public HttpResponseMessage CreateHuman(){
    // Code handling POST requests to /api/Humans.
  }
}
```

```
public class Human{
  public int Id { get; set; }
  public string Name { get; set; }
  public int Age { get; set; }
}
```

MAPPING METHODS 2

```
public class HumansController : ApiController{
  [AcceptVerbs("GET", "HEAD")]
  public HttpResponseMessage ReturnListOfHumans(){
    // Code handling GET and HEAD requests for /api/Humans.
  }
  [AcceptVerbs("POST")]
  public HttpResponseMessage CreateHuman(){
    // Code handling POST requests to /api/Humans.
  }
}
```

MAPPING METHODS 3

```
public class HumansController : ApiController{
  public HttpResponseMessage GetListOfHumans(){
    // Code handling GET requests for /api/Humans.
  }
  public HttpResponseMessage PostHuman(){
    // Code handling POST requests to /api/Humans.
  }
}
```

CONSTRUCTING THE RESPONSE

```
public class HumansController : ApiController{
  public HttpResponseMessage GetListOfHumans(){
    HttpResponseMessage response = Request.CreateResponse(
                                     HttpStatusCode.OK,
                                     theListOfHumans
                                   );
    response.ReasonPhrase = "OK";
    return response;
  }
}
```

```
public static List<Human> theListOfHumans = new List<Human>{
  new Human{Id=0, Name="Agnes", Age=10},
  new Human{Id=1, Name="Bella",  Age=15},
  new Human{Id=2, Name="Cicela", Age=20},
  ...
}
```

RESPONSE IN XML

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Date: Wed, 24 Feb 2016 12:52:36 GMT
Content-Length: 879

<ArrayOfHuman
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://schemas.datacontract.org/2004/07/ApiDemo.Models">
  <Human><Age>10</Age> <Id>0</Id> <Name>Agnes</Name></Human>
  <Human><Age>15</Age> <Id>1</Id> <Name>Bella</Name></Human>
  <Human><Age>20</Age> <Id>2</Id> <Name>Cicela</Name></Human>
  ...
</ArrayOfHuman>
```

RESPONSE IN JSON

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Wed, 24 Feb 2016 12:55:21 GMT
Content-Length: 437

[
  {"Id": 0, "Name": "Agnes", "Age": 10},
  {"Id": 1, "Name": "Bella", "Age": 15},
  {"Id": 2, "Name": "Cicela", "Age": 20},
  ...
]
```

HANDLING MULTIPLE GET

```
public class HumansController : ApiController{
  public HttpResponseMessage GetListOfHumans(){
    // Code handling GET requests for /api/Humans.
  }
  public HttpResponseMessage GetHuman(int id){
    // Code handling GET requests for /api/Humans/5.
  }
}
```

```
GET /api/humans
GET /api/humans/5
```

```
/api/{controller}/{id}
```

HANDLING ERRORS

```
public class HumansController : ApiController{
  public HttpResponseMessage GetHuman(int id){
    if(id < 0){
      HttpError error = new HttpError("Negative id.");
      return Request.CreateResponse(
                HttpStatusCode.BadRequest,
                error);
    }
    return Request.CreateResponse(
              theListOfHumans[id]);
  }
}
```

```
Request.CreateErrorResponse(
  HttpStatusCode.BadRequest,
  "Negative id."
)
```

RESPONSE EXAMPLES

```
GET /api/Humans/5
```

```
{"Id": 5, "Name": "Frida", "Age": 35}
```

```
<Human xmlns:i=http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://schemas.datacontract.org/2004/07/ApiDemo.Models">
  <Age>35</Age>
  <Id>5</Id>
  <Name>Frida</Name>
</Human>
```

```
GET /api/Humans/-1
```

```
{"Message": "Negative id."}
```

```
<Error>
  <Message>Negative id.</Message>
</Error>
```

RETURNING OTHER VALUES

```
public class HumansController : ApiController{
  public Human Get(int id){
    if(id < 0)
      throw new HttpResponseException(
                  Request.CreateErrorResponse(
                    HttpStatusCode.BadRequest,
                    "Negative id."
                  )
                );
    return theListOfHumans[id];
  }
}
```

MODEL BINDING

```
public class HumansController : ApiController{
  public Human Put(int id, Human human){
  }
}
```

Simple types:

- Read value from URI (query string or placeholders).

Complex types:

- Read values from the body of the request

```
public Human Put([FromBody]int id, [FromUri]Human human){ }
```

HANDLING POST REQUESTS

```
public class HumansController : ApiController{
  public HttpResponseMessage Post(Human human){
    // Validate it (TODO (next slide)).
    human.Id = theListOfHumans.Count;
    theListOfHumans.Add(human);
    var response = Request.CreateResponse(
                        HttpStatusCode.Created, human);
    response.Headers.Location = new Uri(
                        "http://site.com/api/Humans/"+human.Id);
    return response;
  }
}
```

VALIDATING POST REQUESTS

```
var error = new HttpError("Something is wrong!");
if(human.Name.Length == 0)
  error.Add("Name", "Can't be empty.");
if(human.Age < 0)
  error.Add("Age", "Can't be negative.");
if(1 < error.Count())
  return Request.CreateResponse(
           (HttpStatusCode) 422,
           error
         );
```

POST EXAMPLE

```
POST /api/Humans
{"Name": "XXXX", "Age": 12}
```

```
{"Id": 14, "Name": "XXXX",  "Age": 12}
```

```
<Human xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://schemas.datacontract.org/2004/07/ApiDemo.Models">
  <Age>12</Age>
  <Id>14</Id>
  <Name>XXXX</Name>
</Human>
```

POST EXAMPLE

```
POST /api/Humans
{"Name": "", "Age": -12}
```

```
{
  "Message": "Something is wrong!",
  "Name": "Can't be empty.",
  "Age": "Can't be negative."
}
```

```
<Error>
  <Message>Something is wrong!</Message>
  <Name>Can't be empty.</Name>
  <Age>Can't be negative.</Age>
</Error>
```

VALIDATING POST REQUESTS 2

```
public class Human{
  public int Id { get; set; }
  [Required(AllowEmptyStrings=false)]
  public string Name { get; set; }
  [Range(1, 200)]
  public int Age { get; set; }
}
```

But validation should be in the business layer!

```
if(!ModelState.IsValid)
  return Request.CreateErrorResponse((HttpStatusCode) 422,
                                     ModelState);
```

POST EXAMPLE

```
POST /api/Humans
{"Name": "", "Age": -12}
```

```
{
  "Message": "The request is invalid.",
  "ModelState": {
    "human.Name": ["The Name field is required."],
    "human.Age": ["The field Age must be between 1 and 200."]
  }
}
```

```
<Error>
  <Message>The request is invalid.</Message>
  <ModelState>
    <human.Name>The Name field is required.</human.Name>
    <human.Age>The field Age must be between 1 and 200.</human.Age>
  </ModelState>
</Error>
```

HANDLING PUT REQUESTS

Common URI: `api/the-resource-collection/the-id`

Update, or attempt to create.

HANDLING PATCH REQUESTS

Common URI: `api/the-resource-collection/the-id`

Update (not necessarily the entire object).

HANDLING DELETE REQUESTS

Common URI: `api/the-resource-collection/the-id`

SUPPORTING JSONP

- Web browsers comply to the same-origin policy → AJAX can only fetch pages from the same domain.
- Does not comply to the `<script>`-tag!
  - But can only be used for JavaScript.

JSON → JSONP

```
{"a": 1, "b": 2}
```

```
aFunction({"a": 1, "b": 2})
```

- ASP.NET does not support JSONP by default.
  - But can be added through a NuGet Package.
    - https://github.com/WebApiContrib/WebApiContrib.Formatting.Jsonp

WEB API CONFIGURATION

```
GlobalConfiguration.Configure(config => {
  config.MapHttpAttributeRoutes();
});
```

```
public class HumansController : ApiController{
  [Route("api/animals/{numberOfLegs}/yellow")]
  public HttpResponseMessage Get(int numberOfLegs){
  }
}
```

POSTMAN

- A program to test web APIs.
- Webpage: https://www.getpostman.com/

RECOMMENDED READING

Web API Design - Crafting Interfaces that Developers Love

- https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf

In the course book *Pro ASP.NET MVC 5*:

- Chapter 24: Model Binding
- Chapter 25: Model Validation
- Chapter 27: Web API and Single-page Applications