# TIE-23500, Web Software Development (Web-ohjelmointi)

## Notes by Daniel "3ICE" Berezvai <http://tut.3ice.hu/>

Finnish and English versions of this course are taught together. Feel free to choose either. Exam answers should be in the language you signed up for.

TA helps with project work in the second term. And maybe with the individual exercises in the first term.

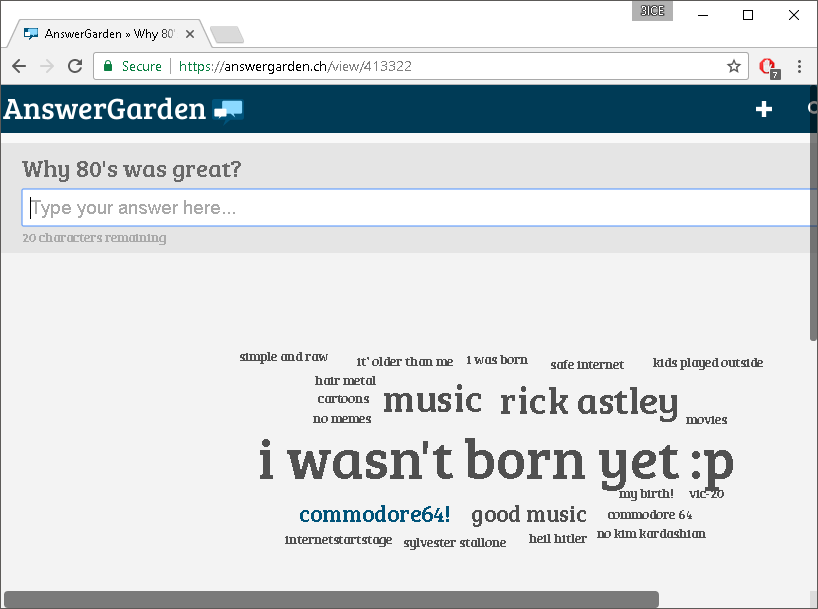
Flipped classroom teaching methodology will be used.

## Why is this subject important

Web is the most common programming platform. In addition, web has affected programming in general.

### Why were the 80's great?

<https://answergarden.ch/view/413322>



In the 80s software was created mostly by following the Waterfall method. And delivered on cassettes.

Requirements → Design → Implementation → Testing → Maintenance

In the 90s, operations and maintenance were moved to the internet.

In 2000 speed became more important. Customers were able to choose, our product had to stay fast enough.

### Developers vs Operations

What can developers do? Nothing (☺), Design, Program. Change!

Operations' aim: Uptime (if it's not broken, don't fix it). Stability!

DevOps approach was created to alleviate the tension between the two. Just like agile development fixes the gap between Business and Development. Combines with Quality assurance as well.

### Lean startups

Have ideas, build product, measure something about it, learn from the data, get new ideas. Spin this cycle as fast as possible.  
MVP → MVP → MVP (?)

### AB-testing

Almost all big web software can easily divide users into groups seamlessly so that we can test if we add a new feature how it affects users.

## Course arrangements

### Learning outcomes

Understand distributed nature, session management, modern app framework, etc.

### Django

We are not using Node.js anymore. (Sorry.)

By learning one framework properly, we will be able to apply that knowledge to other frameworks easily.

Why the change? It was easy to do a hack job in Node.js (Teacher has a harder time assessing our work)

Basic communication in Python, and of course JavaScript is needed anyhow.

This is almost a full stack course.

There are plenty of frameworks. These are familiar: Express, ASP.net, Web2py, RubyOnRails, Flask, Symphony, Nette (PHP framework), etc.

Use whatever for the individual project work, but stick to one framework in groups.

### Content

User <-**-> Browser <--> Web Service <-**-> Database

We are focused on the part in **Bold**.

### Prerequisites and plagiarism

Programming and Databases knowledge required.

Do not copy.

### Outline

First half of the course: Personal exercises and lectures.

Exam between the two terms (3 and 4)

Small group project in second term. Last year group size was 3, this time we have twice the students so might be bigger groups.

### Exercises on GitLab and TUT+

Fork the slides

TUT+ exercise access only opens next Monday <https://plus.cs.tut.fi/seitti/>

### Project

Most of the grade comes from there, most of the work invested in there. The rest is just fail/pass:

Exercise (0/1), exam (0/1), and project (0/1/2/3) added together.

### Tools

I already have my favorite editor thank you.

### Git

All exercise submission in this course are performed using GitLab. Upload our code there, and in TUT+ we just say "my code is in git, please check it"

-Break-

## HTTP

OSI-8 application level protocol, which is Stateless

Follows client-server model, always initiated by client.

### What does statelessness mean?

Every request has to contain enough information for the server to be able to identify it. (Session cookie, for example.)

The server can maintain the state. HTTP, the protocol itself, cannot.

Request: GET/POST/PUT/DEL… HTTP/1.1  
Host: 3ice.hu  
Other-headers: etc.

Reply:

HTTP/1.1 200 OK  
Date: …  
Last modified:

<Body> or Payload

Caching uses that Last modified header, to save bandwidth.

### Safe HTTP methods

No side effect on server-side. GET and HEAD are safe.

Idempotent: They behave the same way no matter how many times we execute it. Their side effect is the same if executed again. This is the case with PUT (upload) and DELETE.  
(Of course the webserver likely won't always follow these guidelines.)

If we submit a form we use a POST request instead. (Non-safe.)

### Response codes

1xx: Information  
2xx: Success  
3xx: Redirect  
4xx: Client error  
5xx: Server error

Use these response codes to indicate what sort of error happened. Don't say "200 OK There was an error.".

How would you implement an "article limit" in online newspapers? In this case, again the server needs to have a state. We could use cookies to identify the client. "This is Ville" We check in the database, how many articles Ville has read. Deny access after so many articles. Or easier: have the client count articles for themselves. Less secure but who cares.

### Layers, tiers

#### LAYERED ARCHITECTURES

In the context of web applications, there are three commonly used layers:

Presentation Layer: Manipulating the view

Business Logic Layer: Behavioral logic

Data Layer: Accessing and editing the data

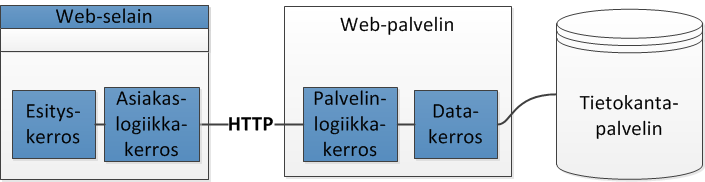
#### TIERS VS. LAYERS

In addition to logical separation, there is also the physical separation. N-tier architecture refers for the physical separation.

Tiers: physical separation

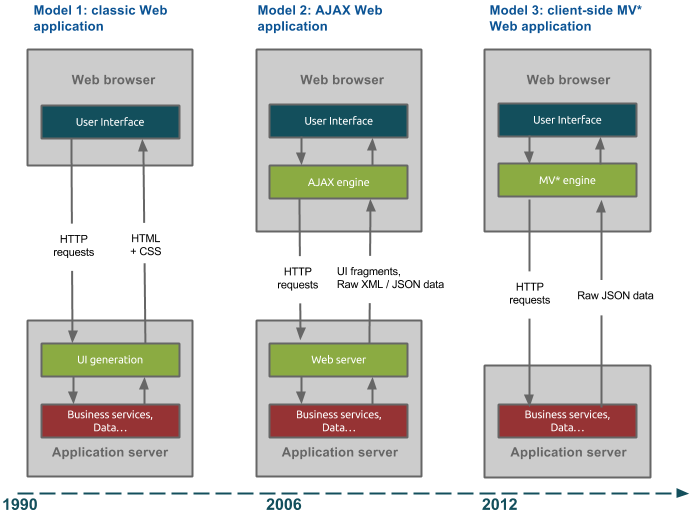
Layer: logical separation

Layer and tiers can get mixed.



### Evolution

Many modern web applications move business logic to the client side. Logic happens in the browser, not on the server anymore. There's just an API for data access.



Obvious counter example: Debugging server application from the same computer / Localhost developing.

Not so obvious: mobile applications where we have an HTML5 wrapper but there we have two servers. One runs on the phone, the other in the cloud.

HTML, CSS next Monday.

Read the rest as homework. see \Web\lectures\02\_http\_html\_css.html

Three layers: Content (HTML), style (CSS), behavior (JS)

The end.