Prof. Dr. Stefan Traub

Distributed and Webbased Systems

Hinweise

Inhaltsverzeichnis

1. Introduction

Let's start with some organizational topics:

1.1. Registration and information about the lecture

You must be registered for the lecture because access to different systems is based on the official student list. If you are not registered yet, please do so in LSF.

1.1.1. Prereqs

This is what you need to know:

- Operating Systems (Linux, Console)
- Computer Networks (Linux)
- Programming Language (Java or C++)

1.1.2. System Access

The access to different systems is granted with the help of public and private keys. To create a key pair, use `ssh-keygen -f <yourloginname>`. Do NOT set a password (passphrase). This command creates two files: `<yourloginname>` contains your private key. Please protect this file and do not share it with anyone. The file `<yourloginname>.pub` contains your public key. Copy this file into the following directory: `p:\traub\i-linux-01.`

Options to copy the file to my public folder. Always replace <username> (including the "<" and ">") with your regular loginname.

Inside our university, use the regular copy command.

```
copy <username>.pub p:\traub\i-linux-01
```

From outside using windows (notice the backtick symbol at the end of the second line):

```
$c=Get-Credential
Get-Content <username>.pub | Invoke-WebRequest -Uri https://fs.thu.de/public/traub/i-linux-01/<username>.pub `
 -Method PUT -Credential $c
```

From outside using Linux or Mac-OS:

```
curl -u <username> https://fs.thu.de/public/traub/i-linux-01/<username>.pub -X PUT --data @<username>.pub
```

Then convert the private key (file without extension) to the putty version using `putygen`. Start Putty without arguments and register this file at the Putty settings under `Connection/SSH/Auth`. Under `Connection/Data Auto-login name`, enter your username.

If you want to use the windows ssh client, use:

```
ssh -l <username> -i <username> i-linux-01.hs-ulm.de
```

Make sure, that the argument after `-i` points to your private keyfile. If you put your private keyfile into your `.ssh` directory and name is "id_rsa", then the `-i` option can be omitted.

1.1.3. Students-VM and Overlaynetwork

Please download the following VM-image from my public folder: https://fs.thu.de/public/traub/VM/vm3.zip.

Please change the default password opsys!

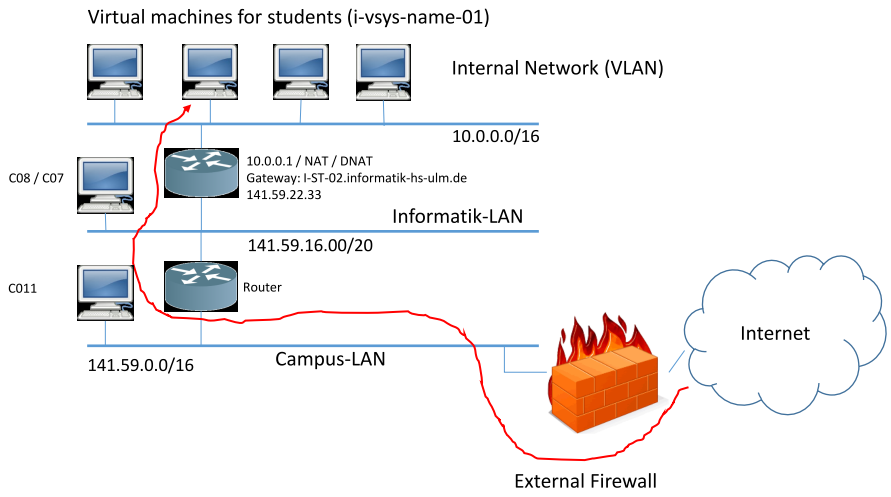
It is the same image, which you might know from our lecture "operating systems". Loginname and password are the same. Please extract the file and run the vm locally.

We want to establish some kind of a virtual laboratory, which should give us the same environment as if this would be a real lab. The connection shall be done at iso/osi layer 2.

Here are the options and the required steps:

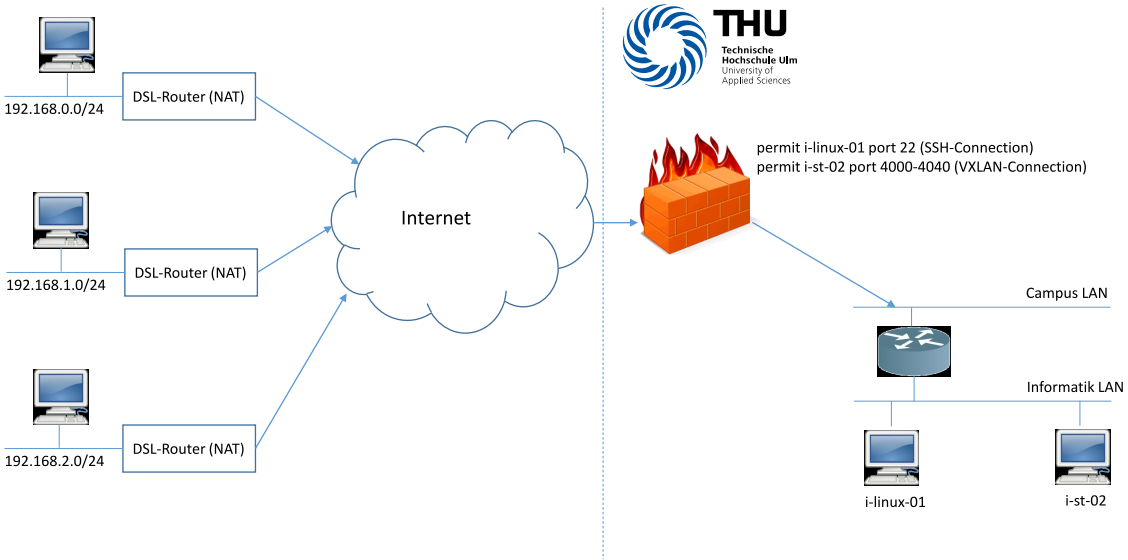
This is an excerpt of our campus net.

THU Campus, Informatik and Internal VLAN



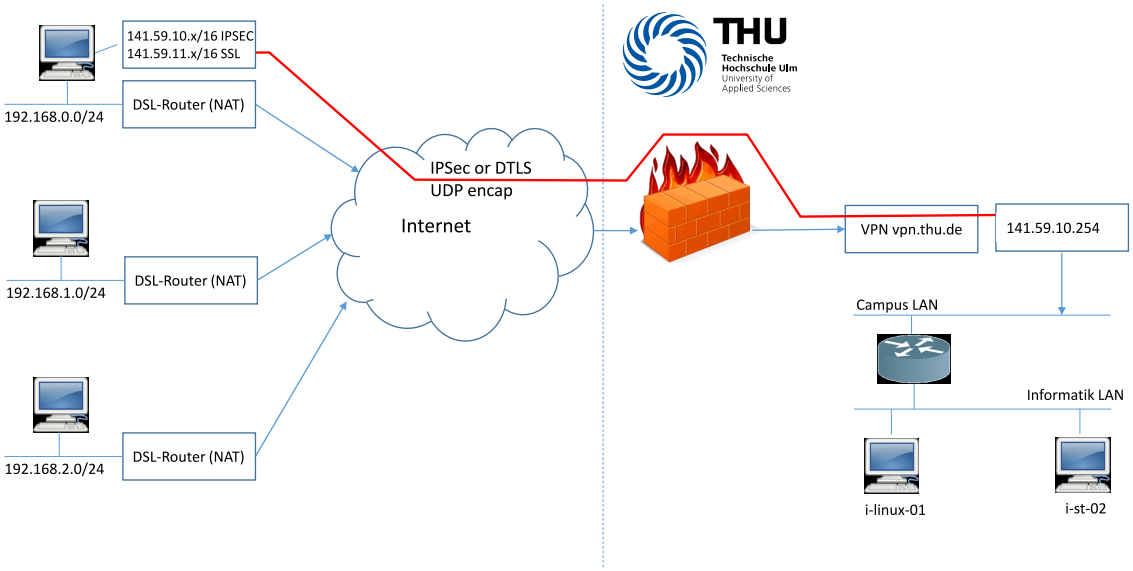
How to connect from your home to our campus.

Connection from Students Home to THU



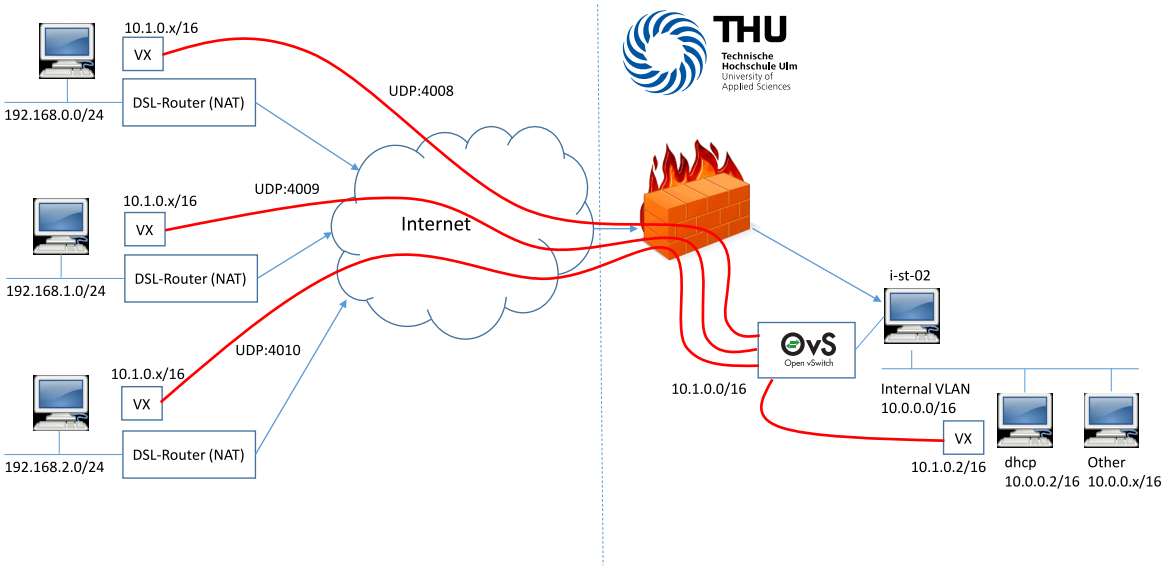
How to connect from your home to our campus with VPN.

VPN Connection



We can establish an vxlan based overlay network. This means, we have a private layer 2 network which includes all students vm.

VXLAN Layer 2 Overlay Network



To actually connect to this network use the following steps:

- Start our vm and login as user opsys.
- Execute the command: `sudo vx start`. Enter your default THU credentials when requested. They will not be stored somewhere.
- Check the connectivity with: `ping 10.1.0.2`.
- To close the session use: `sudo vx stop`.

If you do not want to reenter your credential all the time do the following:

- Change the default password of the opsys account!
- Create the file "`.netrc`". Content: `machine i-st-02.informatik.hs-ulm.de login xxxxxx password xxxxxxx`.
- Use the command "`vx -r start`".

What this "vx" command does:

- It connects to the webservice "https://i-st-02.informatik.hs-ulm.de:4001/create". The password is encrypted (https).
- It then authenticates you against active directory using LDAPS. Again the password is encrypted.
- It searches for a free port.
- It creates a vxlan device on the server.
- It creates a firewall rule for you to allow to send to your port.
- It connects the serverside vxlan device to the ovs switch.
- It creates the client-side vxlan device: `ip link add vx type vxlan id 1 remote $IP ...`
- It sets the vx mac address to the portnumber (for easy debugging)
- It requests an ip address using dhcp: `dhclient vx`

You can easily inspect whats happening. Open "usr/local/bin/vx". It's a bash script.

If you need to update the vx script, please use the following commands:

```
curl -o vx -u <yourloginname> https://fs.thu.de/public/traub/VM/vx.txt
sudo mv vx /usr/local/bin/
sudo chmod +x /usr/local/bin/vx
```

Of course, replace "`<yourloginname>`" with your THU-account, including the "<" and ">" symbols.

1.1.4. Git Repository

If we use git, the location is: p:\traub\dwsys\git or via web: https://fs.hs-ulm.de/public/traub/dwsys/git

1.1.5. Fileshares

Most information about the lecture can be found in my public directory:

`\\hs-ulm\fs\public\traub`

1.1.6. Software

If you use your own laptop in class, please install the following software:

- SSH Client, e.g. Putty
- Git
- Visual Studio Community Edition
- Visual Studio Code

If you are using Mac-OS git should already be present and as SSH client use the Terminal App. The Visual Studio version for Mac may not have all the projects we use, but in any case the basic features. Then install dotnet-core:

- Dotnet Core for Mac
- Powershell: `brew cast install powershell`
- FTP-Client: `brew install ncftp`

If you are using Linux, install git through your Packet Manager. SSH should already be installed. Visual Studio code is available for Linux, but unfortunately not Visual Studio. Install dot-net-core:

- Dotnet Core for Linux
- Webdav support: `apt install davfs2`

- CIFS support: `apt install cifs-utils`

To use Visual Studio Community Edition you need a Microsoft account. Please register here: https://account.microsoft.com.

1.2. Definitions

First some possible definitions:

Definition of a Distributed System (1)

A distributed system is:

A collection of independent computers that appears to its users as a single coherent system.

Definition of a Distributed System (1a)

A distributed system is

one in which the failure of a computer you didn't even know existed can render your own computer unusable

(Leslie Lamport)

Definition of a Distributed System (1b)

A distributed system is: A collection of independent computers that interact with each other in order to achieve a common goal (Wikipedia).

Middleware

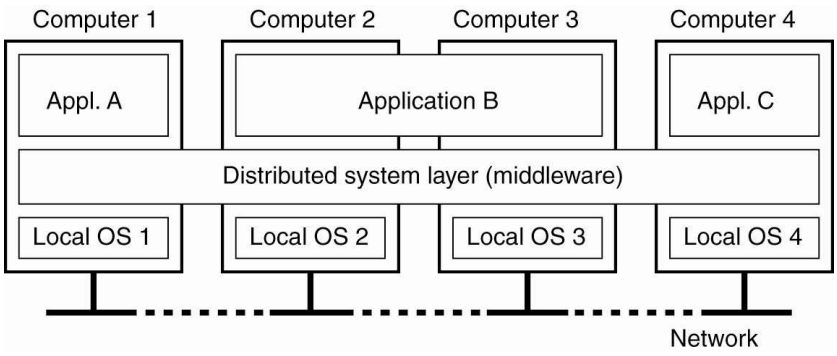


Figure 1-1. A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

1.3. Challenges

Major challenges in a distributed system.

1.3.1. Transparency

Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995).

1.3.2. Goals

Common Goals

- Openness
- Interoperability
- Portability
- extensible
- Separating Policy from Mechanism

1.3.3. Scaling

Always avoid centralized artefacts.

Scalability Problems

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Figure 1-3. Examples of scalability limitations.

Scalability Problems

Characteristics of decentralized algorithms:

- No machine has complete information about the system state.
- Machines make decisions based only on local information.
- Failure of one machine does not ruin the algorithm.
- There is no implicit assumption that a global clock exists.

Scaling Techniques (1)

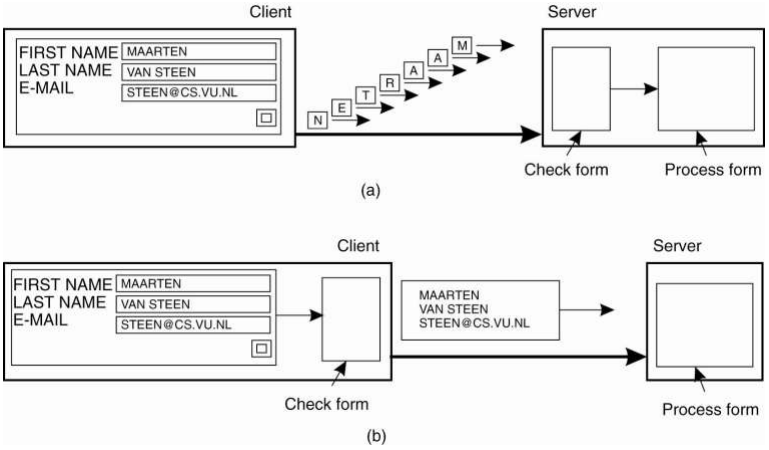


Figure 1-4. The difference between letting (a) a server or (b) a client check forms as they are being filled.

Scaling Techniques (2)

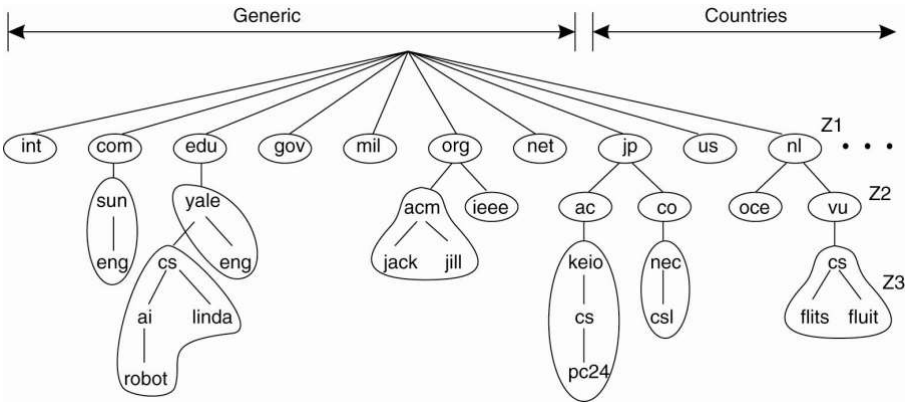


Figure 1-5. An example of dividing the DNS name space into zones.

1.3.4. Pitfalls

Pitfalls when Developing Distributed Systems

False assumptions made by first time developer:

- The network is reliable.
- The network is secure.
- The network is homogeneous.
- The topology does not change.
- Latency is zero.
- Bandwidth is infinite.
- Transport cost is zero.
- There is one administrator.

1.3.5. Services

Here's a small list of important services within a distributed system.

1.3.5.1. Naming

Naming normally runs in the background. But whenever you click on a link on a browser page the first thing what the system does, is to resolve a hyperlink into an IP address. The reason why we use naming is that humans like to remember strings while computers like to remember numbers. Thus we must convert a string into a number. A typical example is the Domain-Name-System (DNS), which a DNS name into an IP address.

We have basically three options to structure a name.

- Flat
- Hierarchical
- Attribute based

1.3.5.2. Time

The main problem is, that no global clock exist. Nevertheless you can try to synchronize the clock time.

1.3.5.2.1. Physical Clock

The official clock is derived from an atomic clock and then synchronized using the NTP-Protocoll. see: Physikalisch-Technische Bundesanstalt

Network Time Protocol

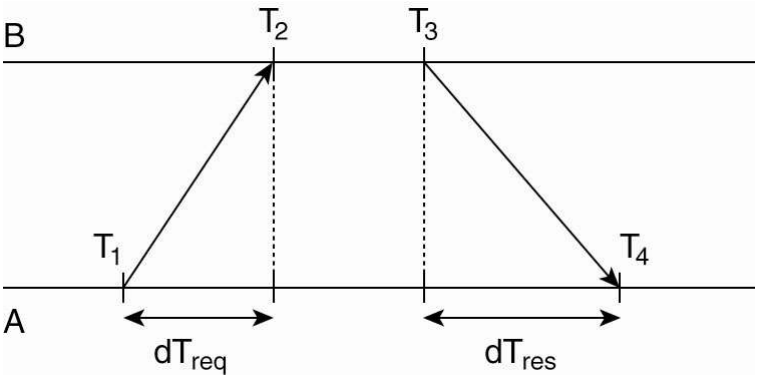


Figure 6-6. Getting the current time from a time server.

1.3.5.2.2. Logical Clock

Sometimes it is not even necessary to use a real clock. We switch to a logical clock, which is simply a counter.

Lamport's Logical Clocks (1)

The "happens-before" relation → can be observed directly in two situations:

- If *a* and *b* are events in the same process, and *a* occurs before *b*, then $a \rightarrow b$ is true.
- If a is the event of a message being sent by one process, and *b* is the event of the message being received by another process, then $a \rightarrow b$

Lamport's Logical Clocks (2)

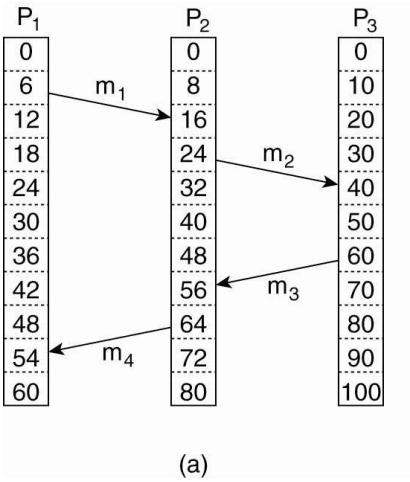


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates.

Lamport's Logical Clocks (3)

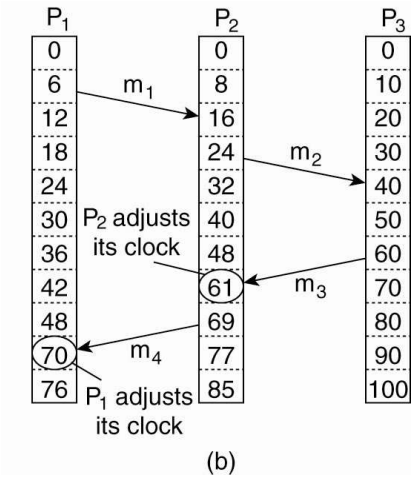


Figure 6-9. (b) Lamport's algorithm corrects the clocks.

Lamport's Logical Clocks (4)

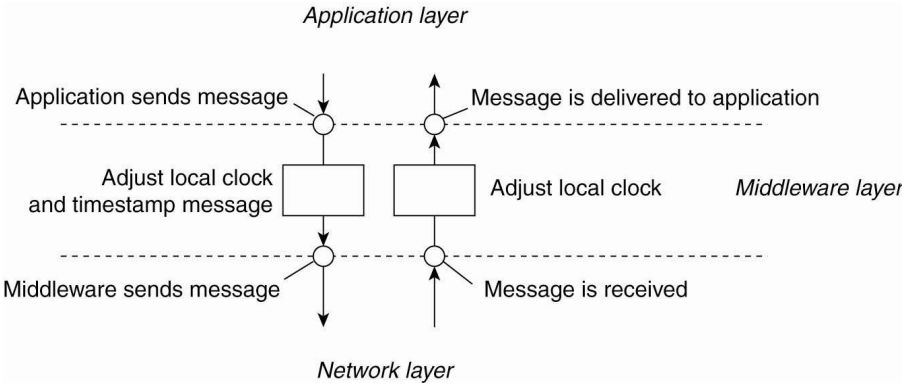


Figure 6-10. The positioning of Lamport's logical clocks in distributed systems.

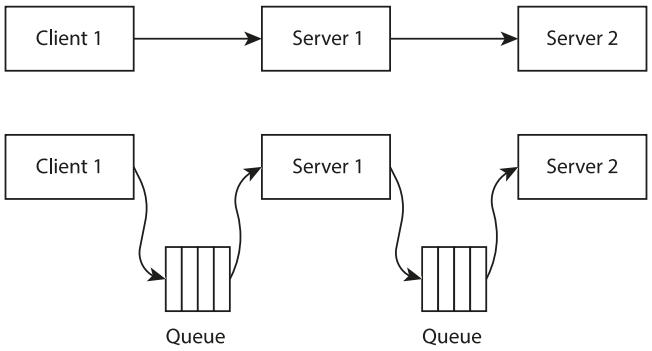
Lamport's Logical Clocks (5)

Updating counter C_i for process P_i

1. Before executing an event P_i executes $C_i \leftarrow C_i + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s timestamp *ts (m)* equal to C_i after having executed the previous step.
3. Upon the receipt of a message *m*, process P_j adjusts its own local counter as $C_j \leftarrow \max\{C_j, ts(m)\}$, after which it then executes the first step and delivers the message to the application.

1.4. Basic Principles

We should observe a couple of basic principles.



1.4.1. Loosely Coupled Systems

Do not relay on synchronous communication. Tolerate a node failure.

1.4.2. Node Resilience

Make sure a node is up and running all the time. If not, use fallback servers.

1.4.3. Self Configuration

Try to configure nodes locally. Don't rely on global information.

1.4.4. Security

Observe security from the very beginning. It is not possible to "add" security afterwards.

1.5. Architectures

Some basic distributed architectures.

1.5.1. Middleware

Middleware

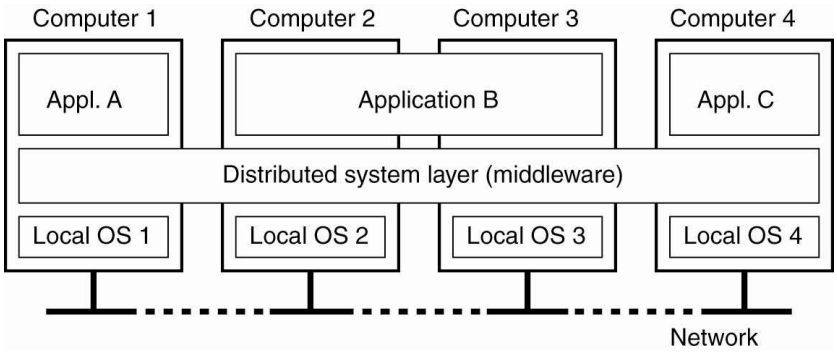


Figure 1-1. A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

Some Examples:

- Basic TCP/UDP
- ONC RPC
- DCE RPC
- DCOM
- Java RMI
- Remoting
- CORBA
- WCF
- XML Webservices
- REST Webservices
- JSON RPC
- gRPC
- Thrift

1.5.2. Client Server

One of the most important architecture is known as "client-server".

Centralized Architectures

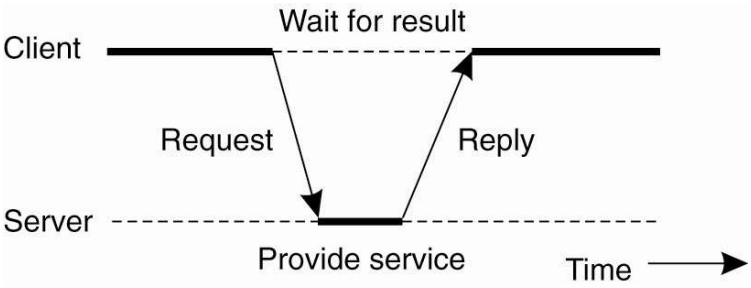


Figure 2-3. General interaction between a client and a server.

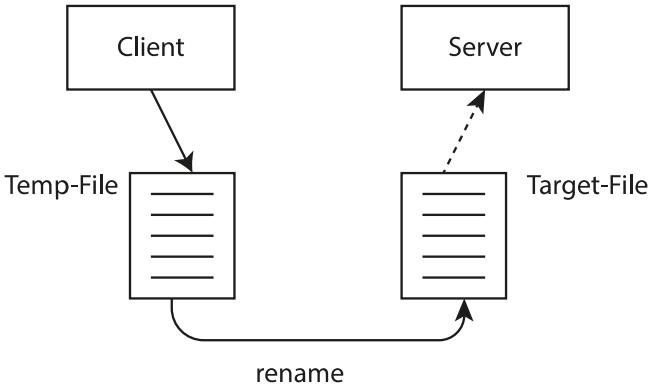
1.5.2.1. Example Send File

(Exercise)

1.5.2.2. Examples Send Message

(Exercise)

Client Server using files.



1.5.2.3. Call Function

(Exercise)

Call remote function.

1.5.3. Multi Tier Architectures

The basic client-server architecture can be extended to multiple layers.

1.5.3.1. Application Layering

This is a typical example.

Application Layering (1)

Recall previously mentioned layers of architectural style

- The user-interface level
- The processing level
- The data level

1.5.3.2. Example

Application Layering (2)

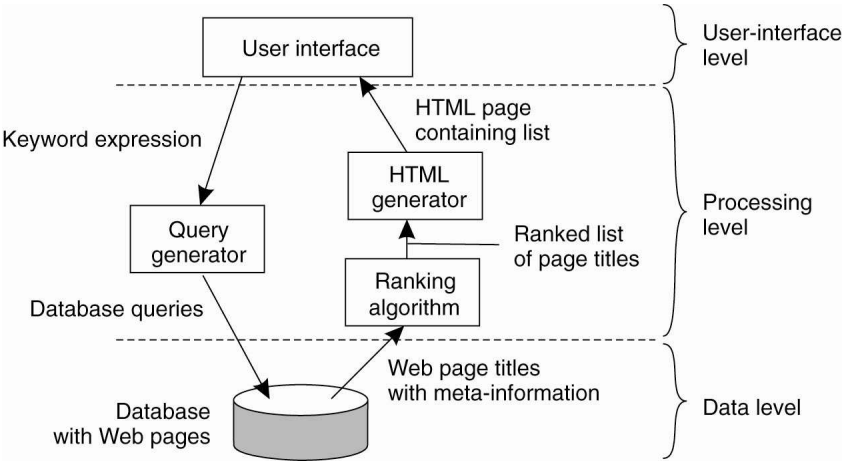


Figure 2-4. The simplified organization of an Internet search engine into three different layers.

2. Basic Tools

We start by learning (or refreshing) basic tools.

2.1. A First Example: Git

Let's start with git as a good example of a loosely coupled distributed system.

2.1.1. Basic usage

(demo)

The following code is executed from within powershell.

First, create a new directory

```
mkdir gittest
```

Change into that directory

```
cd .\gittest\
```

Initialize it

```
git init
```

Now we create a new file

```
notepad hello.txt
```

Inspect the new file

```
gc .\hello.txt
```

Check our git status, this shows us a new file.

```
git status
```

Now we add the file to the repo.

```
git add .\hello.txt
```

Again check the status. Now the file shows up in green.

```
git status
```

Let's create a first commit. For this we nees a commit message.

```
git commit -m "my first commit"
```

Now our status shows clean.

```
git status
```

The git log now shows one entry.

```
git log --oneline
```

Now we change our file.

```
notepad .\hello.txt
```

The status shows a modified file.

```
git status
```

The diff command can be used to inspect the difference.

```
git diff
```

We need to add the file again before a commit.

```
git add .\hello.txt
```

This is out second commit.

```
git commit -m "second commit"
```

Let's see the log.

```
git log --oneline
```

This is the extended version of the log.

```
git log
```

Let's modify the file again.

```
notepad .\hello.txt
```

And add it.

```
git add .\hello.txt
```

And create an other commit.

```
git commit -m "number 3"
```

The git log shows this new commit.

```
git log --oneline
```

Display the conntent of the file.

```
gc .\hello.txt
```

Now we switch our working copy to a previous version. Attention: When you redoo this step, you have to use the YOUR commit id from commit nr. 2.

```
git checkout dec65d8
```

Now the content shows the last version.

```
gc .\hello.txt
```

Switch back to the current master branch.

```
git checkout master
```

The file is up to date again.

```
gc .\hello.txt
```

Now we create a new branch.

```
git branch test
```

Attention: Creating a new branch does not imply to switch to it. The status command indicates we are still on the master branch.

```
git status
```

Using `checkout` we switch to the new branch.

```
git checkout test
```

Now status show up the correct branch.

```
git status
```

Let's make a modification here.

```
notepad .\hello.txt
```

And commit those changes. We use the combined option `-am` which automatically add any known files.

```
git commit -am "first modification of a team member"
```

The log shows the new commit.

```
git log --oneline
```

Check the modifiled file.

```
gc .\hello.txt
```

Switch back to master

```
git checkout master
```

Now we see the file before our modification in the branch.

```
gc .\hello.txt
```

Finally we merge the change into out master branch.

```
git merge test
```

Now the file contains the updated version.

```
gc .\hello.txt
```

We can now create a repository, which might be located on a fileshare. First we create a new directory. This can be anywhere.

```
cd ..
mkdir repo
cd  repo
```

We initialize a "bare" repo. The term "bare" means, that this repo does not contain any working copies.

```
git init --bare
```

Switch back to our working copy.

```
cd ..\gittest\
```

Now we add a remote, which is simply pointer to our nee central repo.

```
git remote add origin C:\temp\repo\
```

This command lists all known remotes.

```
git remote -v
```

Finallay we push our local repo to the remote.

```
git push origin master
```

A new team-member might now clone the repo.

```
mkdir newmember
cd .\newmember\
git clone C:\temp\repo\ .
```

This log command shows the complete history.

```
git log --oneline
```

2.1.2. Architecture

The base idea of "git" is, that it is merely a collection of local text files or compressed objects. A developer can work only locally and commit or reset its work.

An interesting feature of "git" is, that it replaces filename with hashes. In other words: files or object are addressed by content, not by name. This means that it never can happen, that two different file names point to the same content or, vice versa, it can never happen, that to different files have the same name. This is especially interesting, because in a distributed system, like "git", we do not need to synchronize naming using any central component. Git replaces "reference by name" with "reference by content".

2.1.3. IDs

As mentioned before, "git" uses hashes to identify objects. In effect, "git" calculates the id as:

```
id = sha1("type size\0content")
```

Where type is `blob|tree|commit`, size is the size of the object and content is the content itself.

2.1.4. Blobs, trees and commits

Effectively, "git" converts a hierarchical file structure into a tree of hashes, where blobs (binary large objects) are the files an the directories a the trees inner nodes. A commit points to a tree, which is the current directory, and a list of ancestor commits.

The following sequence is a little bit "git hardcore", but shows what happens step by step internally. In effect it creates a commited file test containing "hello world" out of a standard input stream. At last the file is checked out even though it has never been created directly.

```
echo hello world  | git hash-object --stdin -w
git update-index --add --cacheinfo 100644 3b18e512dba79e4c8300dd08aeb37f8e728b8dad test
git write-tree
git commit-tree -m init 5b873f747ccb268e4491f289eb37fc675ff5825b
echo 5efa9e425fd03480b1ca45a27c4dd8a6405e356a >.git/refs/heads/master
git reset --hard
```

2.1.5. Repos

You can work with a local repo only or join a remote repo.

2.1.5.1. Local

In its simplest form, a repository is a local subdirectory, called ".git".

2.1.5.2. Remote

A "remote" is a central place, where the objects from all developers are collected together. When you push to a remote, you store your objects in this central place. Notice again, due to the nature of git using hashes, there can never be a conflict in naming, Of course, we can have a conflict in content.

A "remote" can be file share, a WebDAV server or a SSH-connection.

(exercise)

2.1.5.3. GitHub, Gitlab or MS DevOps

These are famous git repository servers:

- Github

  Famous open source repository. Repositories are open by default. You can pay for private repositories. Github was acquired by Microsoft.
- Gitlab

  Competitor to github. Allows private repository for free. Includes good process tools and CI/CD capabilities. Server can be run locally as container.
- MS DevOps

  Supports TFS and Git repository types. Supports process templates (e.g.scrum) and CI/CD devops. DevOps allows github as source. Deploymenmt to MS-Azure. Free for up to 5 users. Now allows public repositories.
- Bitbucket

  Famous git reporitory. Allows free private repositories. Supports code snippets.

2.1.6. Important Commands

Let's check a couple of useful commands.

- init
- status
- add
- commit
- reset
- log
- remote
- push
- pull
- fetch
- branch
- merge
- blame
- stash

2.1.7. Exercises

depends on your previous knowledge.

2.1.8. Summary and Recap

A recap from chapter 2.2.

2.1.8.1. What is Source Control

The process of software development.

- The process.
- The tools, especially source control.

2.1.8.2. Basic Exercises

Let's do a couple of basic git examples.

1. Create a local directory and initialize a repository. See whats been created.
2. Create a couple text-files and check them in. Create at least 2 files.
3. After every step, check the git status.
4. List the commit history.
5. Use the command `git blame -n <filename>` to list the history of your changes.
6. Create a central repository on a file share or on another local directory. Initialize it with "`--bare`". See whats been created. You can use: `\\hs-ulm\fs\org\Institute\IFI\LV\dwsys`
7. Add your central repo as a remote `git remote add ...`.
8. Push your work to your remote.
9. Join with an other student to share your repo. Or simulate this by yourself.
10. Your team-member should clone your repo. You can do this by yourself as well.
11. Make a couple of changes to one of the files.
12. Commit the changes and push them.
13. Your team member (or yourself) can no pull the changes.
14. Create a new branch and check it out. Every team member should create a branch with a different name!
15. Now every team member should make a change on a different file.
16. Push the changes and pull the result.
17. Checkout the master branch.
18. Integrate your changes from your branch with the `merge` command.
19. Integrate your team members changes as well.
20. Again switch to your branch (checkout).
21. Every team member shall now change a different line in the same file.

22. Push, pull and merge again.
23. Now every team member shall change the same line in the same file.
24. Push, pull and merge again.
25. Now you need to resolve the conflict manually.
26. Check the result with `git blame`.

2.1.8.3. Advanced Exercises

For those of you, who know about the basics, here are a little more advanced exercises.

1. First, make sure your git config is up to date.

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
git config --global core.editor vi
```

2. Create a couple of files, a local repository and commit all files.
3. After each step, i suggest to check the git status and the git log as well.
4. Make a couple (3-4) of changes. After each change create a new commit. Don't forget to add your file after each change or use `commit -am`.
5. Try to amend your last commit. You might need to tag the last commit again.
6. Show the content of one of your files from different commits: `git show [commit]:[file]`. Instead of the commit hash, you can use the tag as well.
7. Use git `diff [commit][commit]`, to compare the difference between commits.
8. You can use `git blame -n` to get a better overview, whats been changed in which commit.
9. Create a new branch and check it out. Make couple of changes and commit your changes. Try to force a merge conflict.
10. Go back to master and merge your changes. Resolve the merge conflict.
11. You can visualize what happened with: `git log --all --decorate --oneline --graph`.
12. Create a team of 2-3 students. Create a share repo on a fileshare. You can use: `\\hs-ulm\fs\org\Institute\IFI\LV\dwsys` I recommend to name the directory with your login name. Initialize a bare repo: `git init --bare`.

    As an alternative, login to our internal test linux and create a repo there. In this case, use the ssh-protocol.
13. One of the team members should create a fresh new local repo, add a readme-file, commit it, add the shared directory as a remote and push the first commit.
14. The other team members should clone (`git clone`) the shared repo into a local copy.
15. Each team member should now create a local branch. Use your loginname as branch-name.
16. Each team member should add a completely new file. Commit the change and push the result to the shared repo.
17. It's now the job of the maintainer to merge those changes and push the result. Every team member gets the result back with `git pull`.
18. Now try the same, but modify the same file to force a merge conflict.
19. Check the result with `git blame`.
20. You can create branches from branches as well. Try that and merge back the results into the first branch and then to the master. Note that git does not dictate any merge strategy. Git is only the toolset to work with. It's completely up to the user to develop a good strategy. This should be done before you start a team project.
21. Go to you shared repo and rename the file `.git/hooks/post-update.sample` to `.git/hooks/post-update`. Now you should be able to clone the repo via http AFTER the next update. Use the url: `https://fs.hs-ulm.de/org/Institute/IFI/LV/dwsys`,
22. A local git repo can have multiple remotes. Add a ssh remote:

    `git remote add myssh ssh://loginname@i-linux-01.hs-ulm.de/existing/direcory`.

    Of course, the directory must have been initialized with the `--bare` option. Push your directory there and clone it again.
23. If you have an account on github, gitlab, bitbucket or dev.azure.com, you can use that repo for your push. I recommend to use one of them when you work in teams. Normally there is a basic account for free.

2.1.8.4. Internal Exercises

Now a couple of exercises to get in touch with the git internals.

1. Do this exercise on linux! Start with a completely fresh new local repo.You can safely delete any file within the `.git/hooks` directory. For this exercise you can irgnore the logs directory. But do not delete it!
2. Add a single file and commit the file.
3. Which files have been created in the objects subdirectory? Use the `tree` command to visualize the files. Display any of these objects using git `cat-file -p` or git `cat-file -t`.
4. Go to the directory of your files blob and try to decode the file manually using `pigz`.
5. Redirect the output of the previous step to a new file and calculate the sha1 hash out of it. Use `sha1sum`.
6. Open the file `.git/HEAD`. What is the content of the file? What does this mean?
7. Open the file pointed to by the HEAD. What do you find in this file?
8. Add an other new file and commit it.
9. Explain the newly created objects. You might want to `cat-file` the new objects.
10. What do you think has changed in `HEAD` and `refs/heads/master`? Check it out.
11. Create a tag for your first and your last commit. Check the contents of the files in `.git/refs/tags`.
12. Now checkout the fist commit using your tag or the commit-hash.
13. Now you are in "detached HEAD state". What does this mean? Inspect the content of the HEAD file. What is different?
14. Go back to your latest commit using `git checkout master`.
15. Create a new branch. Inspect whats changed and explain.
16. Checkout the new branch. What do you think has changed now. Verify your assumption.
17. Switch back to the master branch.
18. Create a bare repo in an other directory, add it as a remote and push your repo.
19. Clone this repo into a third directory. Change to this clone and make a small change to the file, e.g. add a line at the end. Commit and push your change.
20. Go back to your first directory and inspect the file tree.
21. Now bring in the remote change with `git fetch origin master`.
22. Again inspect the file tree and explain the difference. Why is it completely safe to bring in the remote objects without any check?
23. Inspect the files `.git/refs/remotes/origin/master` and `.git/refs/heads/master`. Why are theses files different?
24. Now bring in the remote changes: `git merge master origin/master`.
25. Again inspect the files from exercise 22.
26. At the end, inspect all changes with `git reflog`.

2.1.8.5. Online tutorials

These are good online references or tutorials:

- Git documentation: https://git-scm.com/doc
- Atlassian tutorial:https://www.atlassian.com/git/tutorials.
- Youtube tutorial for beginners: https://www.youtube.com/watch?v=SWYqp7iY_Tc

2.2. Visual Studio Code

What is VS-Code?

2.2.1. Basics

Some basic operations.

- Explorer
- Terminal
- Settings
- Search
- Source Control
- Debugger
- Extensions

2.2.2. Extensions

The necessary extensions based on you language are normally installed by default. These might be useful as well:

- Git-Lens
- Live-Server
- Remote Workspace

2.2.3. Exercises

Depends how far we proceed.

2.3. Netcommands

Some useful commands on the command line.

2.3.1. Motivation TCP/UDP Connection

Most higher level api's rely on transport-level tcp connection.

2.3.2. Linux

Linux comes with a couple of nice commands.

2.3.2.1. telnet

The easiest way to check a tcp connection is telnet. We will use telnet to connect a webserver and download the homepage. The HTTP-Protocol is quite easy. The command is:

```
GET / HTTP/1.0
```

Telnet is Invoked as: `telnet host port`.

Exercises:

- Download google homepage.
- Create a text or HTML page and download it.
- Use putty as telnet client.

2.3.2.2. nc

Basically cat over tcp or udp.

Help page: https://wiki.ubuntuusers.de/netcat/.

Exercises:

- Use nc in client and server mode and send a hello world string. On the client-side, use the option `-N`.
- Use nc in server mode and connect with telnet to it. Attention: On a german keyboard, the escape character `^]` is `Strg-5`.
- Transfer a file using nc.

2.3.2.3. wget

The command `wget` allows to download files and invoke web-requests.

Help page: https://wiki.ubuntuusers.de/wget/.

Exercises:

- Download some webpage to a file.

- Download your own webpage or textfile. To do this, create a file in `public_html` and set the access rights to `755`. The request URL is `http://i-linux-01.hs-ulm.de/~yourname/yourfile`.

A small HTML file looks like this:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <h1>Demo Page</h1>
    <p>Text Text Text Text.</p>
</body>
</html>
```

2.3.2.4. curl

More flexible than wget, comes with a supporting library.

Help page: https://wiki.ubuntuusers.de/cURL/.

Official webpage: https://curl.haxx.se/.

Exercises:

- Download another webpage as well.
- Download the current weather forecast: `http://wttr.in/`.
- Have a look at the following website: https://jsonplaceholder.typicode.com/. Try to download all posts.
- You might want to format the output with `jq`.
- Try to download a single post.
- Download a file from your home directory `https://fs-hs-ulm.de/users/yourname/...` Add the option `-u yourname`.
- Create a file on your home directory containing some text. You need the option `-X PUT` and `--data "some text"`.

2.3.3. Powershell

In powershell, we can use `Invoke-Webrequest` or use plain tcp programming from dotnet.

2.3.3.1. Invoke-Webrequest

Exercise: Download a web page with powershell.

2.3.3.2. Windows command wc

The command `wc` is the implementation for windows. It is not part of the windows distribution.

Exercises: Redo the exercises from linux "nc" on windows.

2.3.4. Exercises

Try to establish a communication between windows and linux.

2.4. SSH Remote Login

Secure Shell is the recommended way to connect to a remote computer.

2.4.1. Ports

SSH uses port 22 by default. The port can be changed.

2.4.2. Configuration

Important config files:

- ~/.ssh/config

  Example:

```
Host i-linux-01
    HostName i-linux-01.hs-ulm.de
    Port 12345
    User loginname
Host myhost.com
    IdentityFile ~/.ssh/my.key
Host another.com
    StrictHostKeyChecking no
```

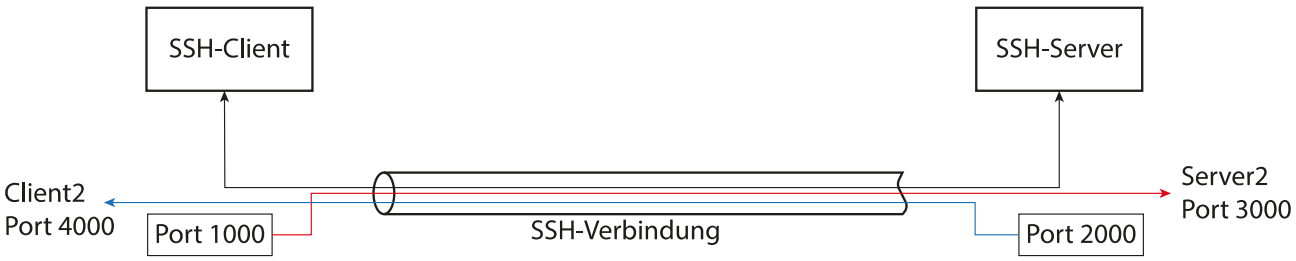
- ~/.ssh/id_dsa ~/.ssh/id_rsa
- ~/.ssh/authorized_keys[2]
- ~/.ssh/known_hosts

2.4.3. Public Private Key Login

(Exercise already done)

2.4.4. X-Forward and Tunnel

SSH allows to tunnel a tcp connection in parallel to the regular ssh data stream. This can be done in both directions.




```
SSH -L1000:Server2:3000 -R2000:Client2:4000 SSH-Server
```

From local to remote, use: `-L<localport>:<remoteaddr>:<remoteport>`.

From remote to local, use: `-R<remoteport>:<localaddr>:<localport>`.

Don't use the "<" brackets!

Exercises:

- Start a X-Server and start an X-Application, e.g. `xclock`, `xwindows`, `code`.
- Connect to your virtual machine. Establish a tcp tunnel. Send a message to your virtual machine. Use `wc` and `nc`.
- Try the other direction.

2.5. Remote and Distributed Filesystems

Remote file systems were the first attempt to create distributed systems. A user wants to access a remote file or share files with other users. There are a lot of protocols. We can distinguish between two basic models.

- Remote-Access: Change the file directly on the server using a remote method call.
- Upload / Download: Download a file, change it locally, and then upload it again.

Beyond the basic file operations (open, close, read, write, seek), a file system, locally or remotely, might support file locking as well.

2.5.1. TFTP

Trivial File Transfer Protocol.

- Transfer file without authentication
- Nowadays used for remote boot.

Protocol explanation: https://www.slideshare.net/PeterREgli/tftp-6027256

Commands: http://www.tutorialspoint.com/unix_commands/tftp.htm

(exercise)

2.5.2. FTP / FTPS

FTP was one of the first protocols for file transfer.

It supports active and passive mode: explanation at slackside

Command explanation: http://www.nsftools.com/tips/MSFTP.htm

(exercise)

FTPS is basically FTP over SSL oder TLS.

2.5.3. Local Mount

Before continuing, make sure you know how to mount local devices!

(exercise)

2.5.4. NFS

NFS is the most important protocol for unix systems. NFSv3 is not very secure but used in internal networks. NFSv4 adds more security bases on GSS-Api.

The basic setup procedure involes:

1. Install necessary packages: nfs-kernel-server, nfs-common
2. Create a data directory
3. Create an entry in `/etc/exports`
4. start the export: `exportfs -a`, you can restart the nfs-kernel-server as well.

5. Mount on the export on the client: `mount -t nfs server:exportdir localdir`.

Security problems:

- Security based on ip-address only (Version 3).
- User IDs are not mapped, but used without any change.
- Root-Squash enabled by default.

(exercise)

2.5.5. CIFS

Microsofts implementation of a remote file protocol uses the cifs protocol (former smb).

Create a share (requires admin privileges):

- Use explorer, right-click directory and enable share
- Powershell: `New-SmbShare`
- pre powershell: `net share <Name>=<Path> /grant:<whoom>,FULL`
- Linux: Edit file `/etc/samba/smb.conf`.

Here is an examples how to (cifs) share the directory data in samba (as root append to `/etc/samba/smb.conf`):

```
[data]
    path = /data
    writeable = yes
```

You need to restart the samba daemon after you made the changes. Make sure, that the directory `/data` exists!

```
systemctl restart smbd
```

Samba manages it's own password database. So set a cifs password for the opsys user:

```
smbpasswd -a opsys
```

Connect to a share:

1. Use explorer: Map network drive.
2. Use command: `net use letter: \\server\share /username:domain\user`.
3. Powershell: `New-PSDrive –Name "K" –PSProvider FileSystem –Root "\\server\share"`.
4. Linux: `mount -t cifs //server/share <directory> -o username=name@domain`.

Exercises:

1. Connect via cifs to our demo host. Use user demo.
2. Connect via cifs to our windows-server. User user demo (the password is different!)
3. Create your own share on your VM. Don't forget to setup an extra smb-password.
4. Connect to your vm from your windows-machine. (Mac-OS should work as well). You can ommit the domain part of the username.
5. Write and read files from the share.
6. Review the files from within linux.
7. You can try to connect to other users as well (as long a you know their username and password)

CIFS mount with options to connect to your homedrive.

```
mount -t cifs //rz-max1.hs-ulm.de/users/traub /mnt -o username=traub@hs-ulm.de,file_mode=0600,dir_mode=0700,uid=
```

or:

```
mount -t cifs //rz-max1.hs-ulm.de/users/traub /mnt -o username=traub@hs-ulm.de,file_mode=0600,dir_mode=0700
```

2.5.6. WebDAV

WebDAV uses HTTP for filetransfer. It is an example for an upload/download model.

Setup of a WebDAV server is a little bit more complicated.

1. Install a Webserver
2. Create certificates, as most clients require https.
3. Enable WebDAV support (see doku).

Mount the share:

- Windows: `net use drive: https://server.domain/share`.
- Linux (root): `mount -t davfs https://sever.domain/share mountpoint`.
- Linux (regular user): Enable S-Bit from `mount.davfs` and add an entry to `/etc/fstab`.

Exercises:

1. Connect to our testserver via webdav. Again use user: demo.
2. Upload and download files.
3. Connect your windows computer to your home-drive in our university. The path is: `https://fs.thu/de`.

If you have enough time, you can try to install apache and enable webdav:

```
apt -y install apache2
sudo a2enmod dav
sudo a2enmod dav_fs
```

Create a password file:

```
htpasswd -c /etc/apache2/demo
```

This section intentionally left blank. (acuc-en)

Restart apache2 (`systemctl restart httpd`) and try to connect:

2.5.7. SSH, sftp, sshfs

SSH can also be used for filetransfer: Use can use `scp` or the more complex `sftp`. In both cases, it might be a good idea to configure passwordless login on the target machine first. For example, if you want to connect to a remote machine and use a specific private-key-file, you can use this config file `~/.ssh/config`:

```
Host demo
    HostName 10.1.0.34
    User demo
    IdentityFile ~/.ssh/demo
```

This means, when connecting to host demo, use these settings. Of course you need to download and install the private key first:

```
scp demo@10.1.0.34:.ssh/id_rsa ~/.ssh/demo
```

The try to connect: `ssh demo`. You should be logged in without a password.

Exercise: use command `sftp demo` an explore the commands.Download an upload some files.

3. Languages

3.1. C#, a small Introduction

The objectives of this chapter is a quick Introduction to the programming language C#. We will use this language to create a basic client server application. We will learn which programming models are suitable for these kind of applications, especially when there are a lot of clients.

3.1.1. Prerequisites

You need to know how to program in Java. The basic structures are the same.

If you know C++, this is sufficient as well, but the differences are bigger.

3.1.2. Getting Started

In order to run a C# application, it needs to be compiled to MSIL-Code. You can use an IDE like Visual-Studio, an editor like VS-Code or a command-line tool like dotnet (from dotnet core). These are the steps for dotnet-core:

- `dotnet new console`
- edit the file "Program.cs" if needed.
- `dotnet run`

3.1.3. Selected Language Constructs

Here is a selected subset of the C# language constructs.

3.1.3.1. Hello World

Let's start with with the classical `hello world`.

- Basic setup
- Base types and @ $ Strings.
- var and dynamic types.

3.1.3.2. Classes and Interfaces

- Create a class with some fields.
- Use constructor or initializer to setup the fields.
- Use getter and setter.
- Implement `ToString`() and print it out.
- Inherit from a class.
- Implement an interface.

```
using System;

namespace csdemo
{
    interface print
    {
        void print();
    }

    class data
    {
        public string name;
    }

    class sub : data, print
    {
```

```
        public void print()
        {
            Console.WriteLine(this.name);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            data d = new data();
            d.name = "test";
            Console.WriteLine(d.name);

            data e = new data() { name = "hans" };
            Console.WriteLine(e.name);

            sub s = new sub() { name = "sub " };
            Console.WriteLine(s.name);
            s.print();
        }
    }
}
```

3.1.3.3. Properties

Setter and Getter are replaced by properties.

- Replace the getter and setter.
- Define an extended getter.
- Use default get and set.

```
using System;

namespace csdemo
{
    class data
    {
        string name;
        public String Vorname { get; set; }
        public data()
        {
            name = "test 2";
        }

        public string Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            data d = new data();
            d.Name = "New";
            Console.WriteLine(d.Name);
            d.Vorname = "Hans";
            Console.WriteLine(d.Vorname);
        }
    }
}
```

3.1.3.4. Indexer

An indexer is used to "simulate" an array access.

- Create a string based index.
- Simulate setter.

```
using System;

namespace csdemo
{
    class data
    {
        public String this[String v]
        {
            get
            {
                return "Objekt " + v;
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            data d = new data();
            Console.WriteLine(d["test"]);
        }
    }
}
```

3.1.3.5. Exceptions

Exceptions are quite similar to java.

1. Catch a div by zero exception.
2. Throw and catch your own exception.

3.1.3.6. Delegates

Delegates a pointer to methods.

- Create a delegate and assign two different static methods.
- Assign an instance method and call the delegate.

```
using System;

namespace csdemo
{

    class Program
    {
        delegate int func(int a, int b);

        static func f;

        static int add(int a, int b)
        {
            return a + b;
        }

        static int sub(int a, int b)
        {
            return a - b;
        }

        static void Main(string[] args)
        {
```

```
            f = sub;
            Console.WriteLine(f(1,2));
        }
    }
}
```

3.1.3.7. Enumeration and yield

Yield is used to freeze the enumeration process until the next element is requested.

- Fill an array with 100 integers an iterate over them using `foreach`.
- Implement this array "virtually" with basic enumeration. Create a new class which inherits from `IEnumerable`.
- Replace the implementation with `yield`.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;

class Data : IEnumerable
{
    string[] colors = { "red", "green", "blue" };
    public IEnumerator GetEnumerator()
    {
        for (int i = 0; i < colors.Length; i++)
        {
            yield return colors[i];
        }
    }

}

class Program
{
    static void Main(string[] args)
    {
        Data d = new Data();
        foreach (var item in d)
        {
            Console.WriteLine(item);
        }
    }
}
```

3.1.3.8. Extension Methods

Need some more methods on Strings? Use extensions methods.

- Add an extension method with capitalizes the first letter.
- Add an other extension method of your choice.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;

static class Ext
{
    public static String FirstUpper (this String v)
    {
        return v.Substring(0, 1).ToUpper() + v.Substring(1);
    }
}

class Program
{
    static void Main(string[] args)
```

```
    {
        Console.WriteLine("abc".FirstUpper());
    }
}
```

3.1.3.9. Attributes and Reflection

C# has a reflections api as well. This is basically the same in java.

- Get name of an object and list it's properties.
- Add more methods and try again.
- Create an attribute. Invoke methods from any class containing a given attribute.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

class Help : System.Attribute
{
    public string txt;
    public Help(string txt)
    {
        this.txt = txt;
    }
}

[Help("Hilfetext")]
class Data
{
}

class Program
{
    static void Main(string[] args)
    {
        Data d = new Data();
        Help h =
        (Help)d.GetType().GetCustomAttributes(typeof(Help),false)[0];
        Console.WriteLine(h.txt);
    }
}
```

3.1.3.10. Lambda Expressions

Lambda expressions are anonymous functions. They are often used with delegates.

- Create a delegate which checks an integer for validity.
- Create a check method which takes an integer and a check function as input. Make an example.
- Replace the function with a delegate.
- Check for a range of values.
- Lambdas can be used as body of a function.

```
using System;
using System.IO;
using System.Threading;

class Program
{
    delegate bool Test(int a);

    static bool LT10(int a)
    {
        return a < 10;
    }

    static void print(int v, Test f)
```

```
    {
        Console.WriteLine(v + " " + f(v));
    }

    static void Main(string[] args)
    {
        print(8, LT10);
        print(18, LT10);
        print(20, (x) => x < 30);
    }
}
```

3.1.4. Concurrent Programming

3.1.4.1. Threads

In java, the threads main function is always `run`. Because of delegates, we can choose any name.

- Create a counting thread which increments a value every second (see operating systems).
- Execute the thread multiple times.
- Pass a Thread-ID.
- When will the main process be terminated?

```
using System;
using System.IO;
using System.Threading;

class Program
{
    static void Work(Object arg)
    {
        for (int i = 0; i < 100; i++)
        {
            Console.WriteLine($"{i} / {arg}");
            Thread.Sleep(1000);
        }
    }

    static void Main(string[] args)
    {
        for (int i = 0; i < 10; ++i)
        {
            (new Thread(Work)).Start(i);
        }
    }
}
```

3.1.4.2. Asynchronous Programming

The main problem, when it comes to asynchronous programming, is that we need somehow to generate a finite state machine, which handles all state transfers. The compiler uses `async` and `await` to generate this behaviour for us.

The keyword `async` generates the state machine for a function. This allows us to jump out of this function.

The keyword `await` actually jumps out of the function until an event occurs.

Exercises:

- Create a worker function, which returns `Task` an awaits some timeout. The function should print some useful messages.
- Call the function.
- The Task return type can be extended by a type. In this case we can directly assign the result value to a variable.
- Call the function twice an see what happens. When are we back int main?
- Do the same, but await the function.
- Call the worker function using `Task.Run`.
- Wait until all Tasks run to completion.

```
using System;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    async static Task work()
    {
        Console.WriteLine("begin work");
        await Task.Delay(3000);
        Console.WriteLine("end work");
    }

    async static Task<String> read()
    {
        Console.WriteLine("begin work");
        await Task.Delay(3000);
        Console.WriteLine("end work");
        return "text";
    }

    static async void run1()
    {
        await work();
        await work();
    }

    static async void run2()
    {
        String r = await read();
        Console.WriteLine(r);
    }

    static void run3()
    {
        Task.Run(work);
        Task.Run(work);
        Console.WriteLine("start done");
    }

    static void run4()
    {
        Task.WaitAll(new Task[] { Task.Run(work), Task.Run(work) });
        Console.WriteLine("all tasks done");
    }

    static void Main(string[] args)
    {
        run1();
        Console.WriteLine("back in main");
        Thread.Sleep(8000);
    }
}
```

3.1.5. Library Examples

Now let's do some basic library calls.

3.1.5.1. File IO

To begin, create a small text file with some lines.

- Read all text and all lines.
- Open and close files.
- use `using`.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Reflection;
using System.IO;
using System.Globalization;

class Program
{
    static void Main(string[] args)
    {
        string fname = @"c:\temp\test.txt";
        using (StreamReader rd = File.OpenText(fname))
        {
            while (!rd.EndOfStream)
            {
                string line = rd.ReadLine();
                Console.WriteLine(line);
            }
        }
    }
}
```

3.1.6. More Language Constructs

Some more advanced language constructs.

3.1.6.1. Generics

We can use a type as parameter to a class. Same as java.

- Create a simple generic class with a property.
- Create a class with multiple types.

```
using System;
using System.IO;
using System.Threading;

class Data<T>
{
    public T data;
}

class Program
{
    static void Main(string[] args)
    {
        Data<int> x = new Data<int>();
        x.data = 123;

        Data<String> y = new Data<string>();
        y.data = "hello";
    }
}
```

3.1.6.2. Nullable

Nullable values allows you to prevent null reference exceptions

```
using System;

namespace NullableDemo
{
    class Item
    {
        public string Name { get; init; }
        public Item? Next { get; set; }
        public Item(String Name, Item? Next) => (this.Name, this.Next) =
        (Name, Next);
    }
    class Program
```

```
    {
        static void Main(string[] args)
        {
            Item l = new(Name:"Last", null);
            l = new(Name: "Second", l);
            l = new(Name: "First", l);

            Item? m = l;
            while (m != null) {
                Console.WriteLine(m.Name);
                Console.WriteLine(m.Next?.Name ?? "<EOL>");
                Console.WriteLine(m.Next?.Next?.Name ?? "<EOL>");
                Console.WriteLine();
                m = m.Next;
            }
        }
    }
}
```

3.1.6.3. Pattern Matching

The switch statement es extended to allow pattern matching

```
using System;

namespace ConsoleApp2
{
    class Program
    {
        record Person(string First, string Last);
        static void info(object o)
        {
            Console.WriteLine($"object = {o}");
            info1(o);
            Console.WriteLine(info2(o));
            Console.WriteLine();
        }
        static void info1(object o)
        {
            switch (o)
            {
                case 1: Console.WriteLine($"Integer One"); break;
                case int i when i > 10:
                Console.WriteLine($"Big Integer {i}"); break;
                case int i: Console.WriteLine($"Some other Integer {i}");
                break;
                case "hello": Console.WriteLine("Was String 'hello'");
                break;
                default: Console.WriteLine("I don't care"); break;
            }
        }

        static string info2(object o) => o switch
        {
            int i when i > 10 => $"Big Number {i}",
            int i => $"Normal Integer {i}",
            Person ("Gabi", _) => "A Person called Gabi",
            Person p when p.First == "Peter" => "This is Peter",
            string v => $"is a String {v.ToUpper()}",
            _ => "Nobody knows"
        };
        static void Main(string[] args)
        {
            info(1);
            info(2);
            info(20);
            info("hello");
            info(new Person(First:"Gabi", Last:"Hinze"));
            info(new Person("Peter", "Müller"));
            info(1.2);
        }
```

```
    }
}
```

3.1.6.4. LINQ

Language INtegrated Queries can be used to do sql like statements over enumerable objects.

```
using System;
using System.IO;
using System.Linq;
using System.Threading;

class Program
{
    static void Main(string[] args)
    {
        int[] data = { 10, 100, 40, 2, 45, 23, 66, 30 };
        var r = from i in data where i < 30 orderby i select i;
        foreach (var item in r)
        {
            Console.WriteLine(item);
        }

        var s = data.Where(x => x < 30).OrderBy(x=>x);
        foreach (var item in s)
        {
            Console.WriteLine(item);
        }
    }
}
```

3.1.6.4.1. LINQ to Arrays

Some basic exercises:

- Create an array of strings.
- Select a subset of these strings using linq.
- Sort the result set (ascending and descending).

3.1.6.4.2. LINQ to SQL

LINQ can be used to query databases as well.

3.1.6.4.3. LINQ to XML

LINQ can be used to select parts of an xml document.

3.2. Javascript and Typescript

Since our lecture is not primarily targeted to teach javascript, let's investigate a couple of fundamental language constructs.

3.2.1. Motivation

The main reason why we use javascript is that it is the primary language on web-browsers. Javascript is a dynamically typed language and is not related to Java, besides some common flow control statements. One reason, why javascript has been widely adopted in browsers is because it is well suited to implement asynchronous programming style through function literals.

3.2.2. Node

Because being so popular, javascript has been ported to the server side as well. This is where "node" comes into play.

3.2.2.1. Commandline

Call `node` to get a prompt oder `node file.js` to execute a javascript file.

3.2.2.1.1. npm

The command `npm` installs additional modules.

3.2.2.1.2. nvm

Sometime there is trouble with `nodejs` versions. You can use `nvm` to switch between versions.

3.2.2.2. VS Project

VS has a node template (demo).

3.2.2.3. VS Code

VS Code can handle javascript as well.

3.2.3. Javascript Language

Some very basic language contructs. For a more detailed introduction, see: https://www.w3schools.com/js/DEFAULT.asp.

3.2.3.1. Variables

You can declare variables with "var" and "let". The difference is the scope. While "var" declares a variable in the current scope (global object), "let" defines a variable within the current block.

```
var max = 5;

//for (let i = 0; i < max; ++i) {
for (var i = 0; i < max; ++i) {
    console.log("line " + i);
}

console.log(i);
```

3.2.3.2. Functions

Functions are declared using the "function" keyword.

```
function loop(max) {
    for (var i = 0; i < max; ++i) {
        console.log("line " + i);
    }
}

loop(5);
```

3.2.3.3. Function Literals

On of the big advantage of javascript is that functions are treaded as regular data values, which can be assigned to vaiables. These are called "function literals".

```
var f;

f = function () { console.log("test1"); };
f();

f = function (m) { console.log(m); };
```

```
f("abc");

f = () => console.log("test3");
f();

f = m => console.log(m);
f("xyz");
```

3.2.3.4. Arrays

Arrays are very straightforward. They can be extendes using push or split using `slice`

```
var a = [1, 2, 3, "abc", 3.4];
console.log(a);
console.log(a[0]);
a.push("last");
console.log(a);
console.log(a.slice(2, 5));
```

3.2.3.5. Objects

Objects are defined using curly braces. You can very easily add properties or methods. After that, you can convert them to a JSON string.

```
var o = {
    name: "Hans",
    alter: 12
};

console.log(o);

var s = JSON.stringify(o);
console.log(s);

var p = JSON.parse(s);
console.log(p.name);
```

Objects can also be defined by merely adding properties to an instance of the Object-class.

```
var o = new Object();

o.name = "My Object";
o.log = function () { console.log(this.name); };

o.log();
```

Functions are used as constructors for newly created objects.

```
function Person(name, age)
{
    this.name = name;
    this.age = age;
    this.print = function () { console.log(this); };
}

var p = new Person("Hans", 12);
p.print();
```

This is equivalent to:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.print = function () { console.log(this); };
}

var p = new Object();
Person.call(p, "Hans", 12);
p.print();
```

3.2.3.6. Prototypes

Example of prototype

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

var p1 = new Person("Hans", 12);
var p2 = new Person("Petra", 14);

Person.prototype.print = function () {
    console.log(`${this.name} at ${this.loc} `);
};

Person.prototype.loc = "Ulm";

p1.print();
p2.print();
```

Prototypes added to prototypes are accessible from all instances of an object based on the same constructor.

Inheritance can be implemented by direct call of a base contructor

```
function UlmItem() {
    this.print = function () { console.log(`${this.name} at ${this.loc}`);
    };
    this.loc = "Ulm";
}

function Person(name, age) {
    UlmItem.call(this);
    this.name = name;
    this.age = age;
}

var p1 = new Person("Hans", 12);
var p2 = new Person("Petra", 14);

p1.print();
p2.print();
```

or by assigning properties to prototypes.

```
function UlmItem() { }

function Person(name, age) {
    this.name = name;
    this.age = age;
}

UlmItem.prototype.loc = "Ulm";
UlmItem.prototype.print = function () {
    console.log(`${this.name} at ${this.loc} `);
};

Person.prototype = Object.create(UlmItem.prototype);

var p1 = new Person("Hans", 12);
var p2 = new Person("Petra", 14);

p1.print();
p2.print();
```

3.2.3.7. Modules

This is how you define a module:

```
function myModule() {
    this.hello = function () { console.log("hello module"); };
}
```

```
var copyright = "S. Traub";

module.exports = {
    inst: myModule,
    copyright: copyright
};
```

And use it in in your code:

```
const m = require("./module");

console.log(m.copyright);

var f = new m.inst();
f.hello();
```

3.2.3.8. Modern Language Constructs

The actual javascript version supports promises or even "async and await" statements.

3.2.3.8.1. Promises

A definition might be: "A promise is an object that may produce a single value some time in the future". Let's make a simple example:

```
function delay(milliseconds, msg) {
    return new Promise((resolve, reject) => {
        if (milliseconds < 1000) {
            reject("delay value too low");
        }
        else
            setTimeout(() => {
                resolve(msg);
            }, milliseconds);
    });
}

console.log("start app");

delay(2000, "hello after").then(console.log).catch(a =>
    console.log("Error = " + a));

delay(200, "hello after").then(console.log).catch(a =>
    console.log("Errro = " + a));

console.log("delay active");
```

Note: Promises are supported with Javascript (ECMA-Script) from version es2015 and above.If your runtime is older, you might want to use a transpiler like babbel: https://babeljs.io.

3.2.3.8.2. Keywords async and await

The latest extension to javascript is "async & await". It targets the same basic idea than promises, but async & await is better readable. Let's rewrite the previous example with async & await.

```
async function delay(val, msg) {
    await setTimeout(() => console.log(msg), val);
}

console.log("start app");
delay(2000, "hello after");
console.log("delay active");
```

3.2.3.8.3. Classes

Finally, Javascript got "classes". Use classes as in any programming language. Here is a brief example:

```
class UlmItem {
    constructor() {
```

```
        this.loc = "Ulm";
        this.print = function () { console.log(`${this.name} at
        ${this.loc}`); };
    }
}

class Person extends UlmItem {
    constructor(name, age) {
        super();
        this.name = name;
        this.age = age;
    }
}

var p = new Person("Hans", 12);
p.print();
```

3.2.3.9. Library Examples

Here is a simple TCP client and server code:

Server:

```
'use strict';

var net = require('net');
var HOST = "0.0.0.0";
var PORT = 10000;
var server = new net.Server();
server.listen(PORT, HOST);

server.on('connection', function (socket) {
    var remoteAddress = socket.remoteAddress + ':' + socket.remotePort;
    console.log('new client connected: %s', remoteAddress);
    socket.write('Hello, client.\r\n');

    socket.on("data", (data) => console.log(data.toString()));
});
console.log("server ready");
```

Client:

```
const net = require("net");

var HOST = "localhost";
var PORT = 10000;

var client = new net.Socket();

client.connect(PORT, HOST, function () {
    console.log('Client connected to: ' + HOST + ':' + PORT);
    client.write('Hello World!');
});

client.on('data', function (data) {
    console.log('received: ' + data);
    client.destroy();
});
```

3.2.4. Typescript

For a introduction, see: https://www.typescriptlang.org/.

For short: Typescript adds type information to java-script object. Is supports classes and other modern language constructs as well. Typescript is a superset to javascript. This means, you can embed native javascript to each typescript application.

3.2.4.1. Compiler

When you use "node" as your javascript engine, install the typescript compiler with:

```
npm install typescript
```

3.2.4.1.1. Commandline

The command can be invoked via:

`tsc`

You might need to add `node_modules/.bin` to your path.

3.2.4.1.2. Visual Studio

Visual Studio has a built in template for typescript.

Select `New/Project/Typescript/Node.js/Blank Node.js Console Application`.

3.2.4.1.3. VS Code

VS-Code has plugins for typescript. For an introduction, see: https://code.visualstudio.com/Docs/languages/typescript.

To debug typescript, refer to: https://code.visualstudio.com/docs/typescript/typescript-debugging.

To debug the current opened typescript file, you need the following files:

.vscode/launch.json:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "skipFiles": ["<node_internals>/**"],
            "console": "integratedTerminal",
            "preLaunchTask": "tsc-build",
            "program":
            "${workspaceFolder}\\out\\${fileBasenameNoExtension}.js",
            "outFiles": ["${workspaceFolder}/**/*.js"]
        }
    ]
}
```

.vscode/tasks.json

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "typescript",
            "tsconfig": "tsconfig.json",
            "problemMatcher": [
                "$tsc"
            ],
            "group": "build",
            "label": "tsc-build"
        }
    ]
}
```

3.2.4.2. Typescript Cheatsheet

For more information, see: https://rmolinamir.github.io/typescript-cheatsheet/

3.2.4.3. Basic Types

Typescript has only these limited base types:

- boolean
- number
- string
- unknown
- any
- void
- null
- Array
- Tuple
- Enum

Here is a link: https://www.typescriptlang.org/docs/handbook/basic-types.html.

3.2.4.4. Some small examples

A first example:

```
var s : string = "Hello Typescript"
console.log(s)
```

Here an example with basic types.

```
// Basic Types
let m : string = "message"
let n : number = 12.34
let b : boolean = true

// Arrays
let ar1 : string[] = ["hello", "test"]
let ar2 : Array<string> = ["abc","xyz"]
console.log(ar1[1])
console.log(ar2)

// Tuples
let tu : [string, number] = ["tuple", 10]
console.log(tu[0])

// Enums
enum Color {Red,Green,Blue}

let c: Color = Color.Red;
console.log(c)

// unknown
let u : unknown // not known yet
u = "test unkown"
let us : string = u as string // now, we know
console.log(us.length)

// any
let st : any // we never know
st = "any test"
// st = 5   // => runtime error
let ust : string = st
console.log(st.length)
```

A simple function witch adds two numbers.

```
function add(a: number, b: number = 10) : number
{
    return a+b;
}

console.log(add(1,2))
console.log(add(20))
```

You can use classes which are then compiled to native javascript.

```
class A {
    protected _text : string;
```

```
    constructor(text : string) {
        this._text = text;
    }

    public print() {
        console.log(this._text)
    }
}

class B extends A
{
    public print() {
        console.log("text: "+this._text)
    }
}

var my : A

my = new A("Test A")
my.print()

my = new B("Test B")
my.print()
```

Most important, you can define your own structured types.

```
interface Person {
    LastName: String,
    FirstName: String,
    Age?: number
}

function printPerson(p: Person)
{
    console.log(p.LastName+", "+p.FirstName+" "+p.Age)
}

// printPerson({FirstName:"Hans"})  // This is an error!
printPerson({FirstName:"Hans", LastName:"Mayer"})
printPerson({FirstName:"Hans", LastName:"Mayer", Age:12})
```

Here is a version, which uses `async` and `await`. The transpiled javascript relies on promises.

```
function Sleep(sec : number) {
    return new Promise(resolve => setTimeout(resolve, sec));
}

async function test()
{
    console.log("step 1")
    await Sleep(2000)
    console.log("step 2")
}

test()
console.log("back in main")
```

3.2.4.5. Install more Types

Most popular libraries now come with type support. To install JQuery types use:

```
npm install --save-dev @types/jquery
```

3.3. Python

Python became very popular over the last years. Even though this is a dynamically typed language, there is a lot of support, and there are a lot of frameworks available. So let`s have a very brief look at a selected subset of python language constructs. For a more comprehensive tutorial, see: https://www.w3schools.com/python/default.asp.

This concise tutorial shows especially aspects which are relevant to our class. We will concentrate on python version 3.

3.3.1. Getting Started

To get started, you must, of course, have Python installed. Download it from: https://www.python.org/downloads/.

Use any text-editor and create a python-app. Use the extension ".py". To start the app, use: `python myapp.py`. This should run on any operating system. If you use VS-Code, it will do the job for you. It has an extension to debug your code as well.

Sometimes you might need to add an addition library from the internet. There is a builtin package manager, which comes with python. To install the library flask, use: `pip install flask`, or `python -m pip install flask`.

3.3.2. Hello World

Let's begin with the classical hello world, which is quite easy.

```
print("hello world")
```

3.3.3. Variables and Data Types

Python is a dynamically typed language (unfortunately). Variables need not to be declared. You can assign any type to a variable. The data type is determined from the right-hand side of the assignment operator. See an example of "string", "number", "float" and boolean types. "None" means "unknown".

```
text = "hello world"
num = 1
fl = 3.1415926
ok = True
nix = None

print(text)
print(num)
print(fl)
print(ok)
print(nix)
```

3.3.4. Lists, Tuples, Sets and Dictionaries

Python comes with a significant number of list types, which are:

- Lists
- Tuples (immutable Lists)
- Sets
- Dictionaries

Lists are a collection of object of any type, which are accessed by index. You can easily add and remove items.

```
list = ["a", "b", 12]
print(list[1])
list.append("end")
list.remove("b")
print(list)
```

Tuples are, more or less, immutable lists. They are often used as return types from function.

```
t = (1,2,3)
print(t)
print(t[1])
```

Sets are sets of objects. A set cannot contain the same object twice. Set elements do not have an index, but they can be enumerated.

```
s = {"a", "b", "c"}
s.add("d")

print(len(s))
print(s)
print("d" in s)
```

A dictionary is a collection of objects, where objects can be accessed using a key. The key can be any object

```
d = {"name":"hans", "age":12, "phones": [1,2,3]}

print(d)
print(d["age"])
```

3.3.5. Control Constructs

Python uses the typical control constructs like any other language. The main difference is that a block of code is recognized by indentation and not by curly braces.

```
a = 10
list = ["a", "b", "c"]
d = {"name":"hans", "age":12, "phones": [1,2,3]}

if a < 10:
    print("yes")
else:
    print("no")

for i in range(10):
    print(i)

for e in list:
    print(e)

for k,v in d.items():
    print(k + " => " + str(v))

i = 1
while i<10:
    i = i+1
    if i == 2:
        continue
    print(i)
    if i == 6:
        break
```

3.3.6. Functions

Functions are defined with the "def" keyword. Parameters can be passed using sequence or key-value pairs. Parameters can be optional. A function, like any other type, can be assigned to a variable.

```
def hello(msg="test"):
    print(msg)

def add(a, b):
    return a+b

hello("hello world")
f = hello
f("called from f")
print(add(1,2))
print(add(a=2, b=3))
```

3.3.7. Classes

Python supports object-oriented programming. A class is defined using the keyword "class". The special variable "self" refers to the object instance. The function "__init__" is used to initialize the object.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def adult(self):
        return self.age >= 18
```

```
    def __repr__(self):
        return "Name:"+self.name+", age:"+str(self.age)

class Hans(Person):
    def __init__(self):
        super().__init__("Hans", 14)

p = Person("Hans", 12)
print(p)
print(p.name)
print(p.adult())

h = Hans()
print(h)
```

3.3.8. Exception Handling

Exception handling is very straightforward. The syntax from this example required python3.

```
a = 1
b = 0

try:
    c = a/b
    print(c)
except Exception as ex:
    print("caught exception "+str(ex))

print("done")
```

3.3.9. Modules

Python code can be split up into modules. A module is imported by another piece of code using "import" or "from name import name". The latter version only imports one function or variable.

```
import ImpMod
from ImpMod import hello

print(ImpMod.demo())
hello()
```

```
def demo():
    return "from module"

def hello():
    print("module print hello")
```

3.3.10. File IO

File IO is highly related to to the C-Library. In this example, the keyword "with" denotes a context manager. This means the file is automatically closed when the file-variable runs out of scope.

```
fname = "c:\\temp\\hello.txt"

with open(fname, "w") as f:
    f.write("text to file")

with open(fname) as r:
    s = r.readline()
    print(s)
```

3.3.11. String Formatting

To format a string, use curly braces within a format string. The following example should be self-explanatory.

```
a = 1
b = "test"

print("a = {}, b = {}".format(a,b))
```

```
print("a = {0}, b = {1}".format(a,b))

print("a = {va}, b = {vb}".format(va=a,vb=b))

print(f"a = {a}, b = {b}")
```

3.3.12. JSON and Regex

Python can very easily handle JSON-string. To do this, import the "JSON" module.

```
import json

js = '{"name": "Hans", "age": 12}'

d = json.loads(js)
print(d["name"])
print(d["age"])

d["age"] = 22
print(json.dumps(d))
```

Another useful module is "regex", which handled regular expression. The syntax is slightly different from other languages, but the basic ideas apply as well.

```
import re

s = "Height: 12.3, Length: 22.2"
l = re.findall("\d+\.\d+", s)
print(l)

m = re.match("Height: (?P<Height>\d+\.\d+), Length: (?P<Length>\d+\.\d+)",
s)
if m != None:
    print(m.group("Height"))
    print(m.group("Length"))
```

3.3.13. Threading

To handle multitasking, python uses thread or processes. This is an example of how to create threads. The locking is necessary in order to prevent multiple threads from intermixing the print-output.

```
import threading
import time

lock=threading.Lock()

def run(tn):
    for i in range(10):
        lock.acquire()
        print(f"Thread {tn} Loop {i}")
        lock.release()
        time.sleep(1)

for t in range(5):
    t = threading.Thread(target=run, args=(t,))
    t.start()
```

3.3.14. Async and Await

The latest version of python supports "async" and "await". The basic idea is the same as with other languages. Here is an example. Check if you can predict the execution order.

```
import asyncio
import time

async def after(delay, what):
    print(f"after {delay} print {what}")
    await asyncio.sleep(delay)
```

```
    print(what)

async def main1():
    await after(1, 'hello')
    await after(2, 'world')

async def main2():
    await asyncio.gather(
        after(1, 'hello'),
        after(2, 'world')
    )

asyncio.run(main1())
print("==============")
asyncio.run(main2())
```

3.4. Other Languages

Some examples of other interesting languages are:

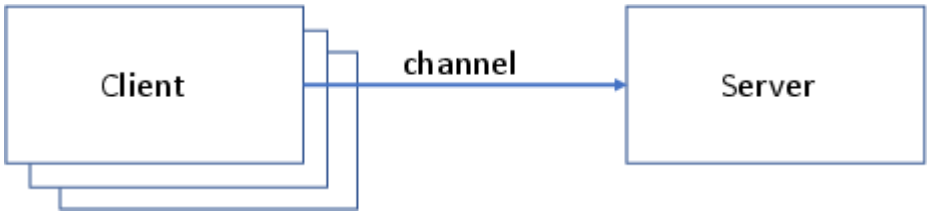
- Go
- Rust
- Swift

4. Distributed Programming

In this chapter, we learn how to implement a basic client server application.

4.1. BasicSystem Model

Our goal is to create a tcp based client server communication. We want to serve multiple clients, thus we use the following system model:



We want to give answers to these questions:

1. How can we connect from client to server?
2. How to serve multiple clients?
3. How does the message stream look like?
4. How to prevent eavesdropping?

When you develop a distributed application, there are the following major steps:

1. Test your application locally with two processes.
2. Move the server (or client) to a different physical machine (a vm will do it).
3. Try to server multiple clients.

4.2. Client Server Connection

Let's first discuss how to connect from a client to a server.

4.2.1. Sockets

The most commonly used api for communication the the socket interface. It is available in nearly every operating system and programming language.

The Socket Interface (1)

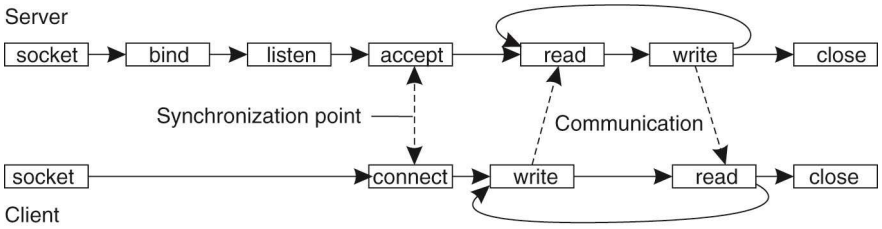


Figure 4-15. Connection-oriented communication pattern using sockets.

4.2.1.1. TCP Client and Server

Examples C# TCP Server.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;

namespace SocketDemo
{
    class Program
    {
        static int Port = 10000;
        static IPAddress Addr = IPAddress.Any;

        static void SocketServer()
        {
            Socket s = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream,
                ProtocolType.Tcp);
            s.Bind(new IPEndPoint(Addr, Port));
            s.Listen(5);
            Socket cl = s.Accept();
            NetworkStream ns = new NetworkStream(cl);
            StreamReader rd = new StreamReader(ns);

            String line = rd.ReadLine();
            Console.WriteLine(line);

            rd.Close();
            ns.Close();
            s.Close();
        }

        static void TcpServer()
        {
            TcpListener s = new TcpListener(new IPEndPoint(Addr, Port));
            s.Start();
```

```
            TcpClient cl = s.AcceptTcpClient();
            using (StreamReader rd = new StreamReader(cl.GetStream()))
            {
                String line = rd.ReadLine();
                Console.WriteLine(line);
            }
            s.Stop();
        }

        static void TcpClient()
        {
            TcpClient cl = new TcpClient("localhost", Port);
            using (StreamWriter wr = new StreamWriter(cl.GetStream()))
            {
                wr.WriteLine("hello from tcp client");
                wr.Flush();
            }
            cl.Close();
        }

        static void SocketClient()
        {
            Socket cl = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream,
                ProtocolType.Tcp);

            cl.Connect("localhost", Port);

            NetworkStream ns = new NetworkStream(cl);
            StreamWriter wr = new StreamWriter(ns);

            wr.WriteLine("hello from client");
            wr.Flush();

            wr.Close();
            ns.Close();
            cl.Close();
        }

        static void Main(string[] args)
        {
            Console.Write("cmd: ");
            string cmd = Console.ReadLine();
            switch (cmd)
            {
                case "ss":
                    SocketServer();
                    break;
                case "ts":
                    TcpServer();
                    break;
                case "sc":
                    SocketClient();
                    break;
                case "tc":
                    TcpClient();
                    break;
            }
        }
    }
}
```

4.2.1.2. UDP Client and Server

(Demo) UDP Client and Server.

4.2.1.3. Unicast, Broadcast and Multicast

When sending messages, we have three options.

- Unicast sends a message from a single node to another single node.
- Multicast sends a message from a single node to a group of nodes.
- Broadcast sends a message from a single node to all other nodes within the same broadcast domain.

4.2.2. Message-Queueing

In principle we can differentiate between transient and persistent communication. In a transient communication the sender and the receiver need to be alive at the same time. A socket communication is a typical example of a transient communication. Whereas in persistent communication the client and the server does not have to be up and running at the same time, they can run at different times. In order to implement persistent communication, we need some kind of persistent layer which stores the messages. This is called message queueing. You can roughly compare messaging as e-mail for applications.

Message-Queuing Model (1)

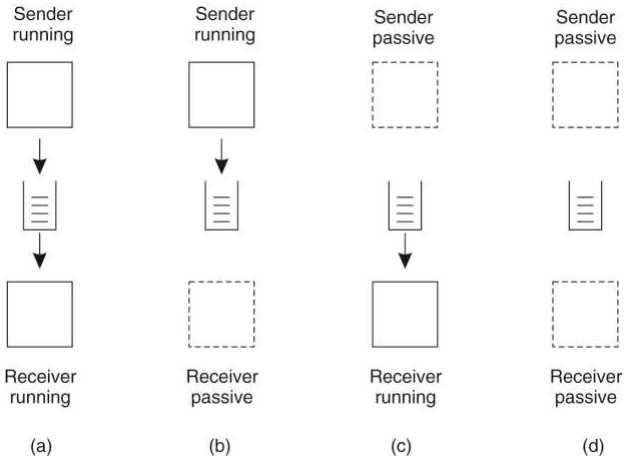


Figure 4-17. Four combinations for loosely-coupled communications using queues.

Examples are:

- Apache ActiveMQ: https://activemq.apache.org/
- IBM MQ
- ZeroMQ: https://zeromq.org/
- RabbitMQ: https://www.rabbitmq.com/
- Microsoft Message Queueing.
- Cloud based message queueing.

4.3. Multiple Client Support

These are possible sever types for multiple clients.

4.3.1. Programming Models

There are three programming models.

Multithreaded Servers (2)

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls

Figure 3-4. Three ways to construct a server.

Let's say, we define a servers main method as:

```
        static void Main(string[] args)
        {
            Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            IPEndPoint ep = new IPEndPoint(0, 9000);
            s.Bind(ep);
            s.Listen(10);
            System.Console.WriteLine("waiting ...");

            (new Thread(()=>server4(s))).Start();

            Console.ReadLine();
            s.Close();
        }
```

And a method, which handles clients requests as follows:

```
        static void HandleClient(Socket cl)
        {
            using (NetworkStream ns = new NetworkStream(cl))
            using (StreamReader rd = new StreamReader(ns))
            using (StreamWriter wr = new StreamWriter(ns))
            {
                String cmd = rd.ReadLine();
                int sec = Int32.Parse(cmd);
                Thread.Sleep(sec);  // Siumlation blocked system call
                wr.WriteLine("ok");
                wr.Flush();
            }
            cl.Close();
        }
```

Then we can implement the different system models:

4.3.1.1. Single Threaded

We use only a single thread which serves all clients. One after the other. Obviously this is not very efficient.

```
        static void server1(Socket s)
        {
            try
            {
```

```
            while (true) {
                Socket cl = s.Accept();
                HandleClient(cl);
            }
        }
        catch (System.Exception ex)
        {
            System.Console.WriteLine(ex.Message);
        }
    }
```

4.3.1.2. Concurrent, Multi Threaded

We can use concurrent programming. Each client gets a new thread. The operating system schedules the thread activities.

```
    static void server2(Socket s)
    {
        try
        {
          while (true) {
              Socket cl = s.Accept();
              (new Thread(()=>HandleClient(cl))).Start();
          }
        }
        catch (System.Exception ex)
        {
            System.Console.WriteLine(ex.Message);
        }
    }
```

4.3.1.3. Thread Pools

Normally we limit the number of threads using a "dispatcher worker" model.

Multithreaded Servers (1)

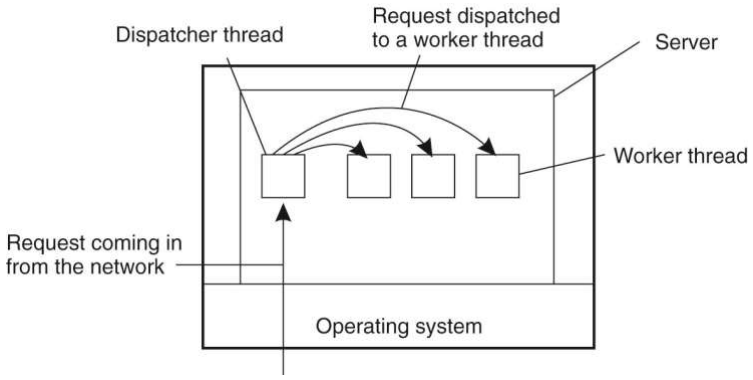


Figure 3-3. A multithreaded server organized in a dispatcher/worker model.


```
    static void server3(Socket s)
    {
        ThreadPool.SetMaxThreads(5, 5);
        try
        {
          while (true) {
              Socket cl = s.Accept();
```

```
                ThreadPool.QueueUserWorkItem((c)=>HandleClient(cl));
            }
        }
        catch (System.Exception ex)
        {
            System.Console.WriteLine(ex.Message);
        }
    }
```

4.3.1.4. Asynchronous, State Machine

If our operating system or runtime environment only supports a single thread, we can suspend our work and switch to a new client. To do this we must save the state of execution. This is known as a finite state machine. In the past this was very annoying, but modern compilers allows us to do this with less work.

```
    static void server4(Socket s)
    {
        try
        {
          while (true) {
            Socket cl = s.Accept();
            HandleClientAsync(cl);
          }
        }
        catch (System.Exception ex)
        {
            System.Console.WriteLine(ex.Message);
        }
    }
```

For this example, we need a special asynchronous implementation of the client handling:

```
    static async void HandleClientAsync(Socket cl)
    {
        using (NetworkStream ns = new NetworkStream(cl))
        using (StreamReader rd = new StreamReader(ns))
        using (StreamWriter wr = new StreamWriter(ns))
        {
            String cmd = await rd.ReadLineAsync();
            int sec = Int32.Parse(cmd);
            await Task.Delay(sec); // Siumlation blockierender Systemaufruf
            await wr.WriteLineAsync("ok");
            await wr.FlushAsync();
        }
        cl.Close();
    }
```

4.3.1.4.1. Summary

In theory all these approaches are equivalent, when we do not take performance into account. With concurrent programming, the operating system saves state (context switch) and decides what comes next. With asynchronous programming, the programmer decides what happens as a result of the next input. The latter is more lightweight but solves the same problems.

5. Security and Cryptography

Keep your message secret!

History of cryptography: https://en.wikipedia.org/wiki/History_of_cryptography.

5.1. Basics of Cryptography

The base idea is, of course, straightforward. Encrypt a plain text message into an encrypted message, which cannot be decodes by anybody, except the intended recipient. We can distinguish between two methods of cryptography.

5.1.1. Some history

Beginning with the ancient Romans, there were many attempts at encryption algorithms.

- Caeser cipher
- Enigma (world war 2)
- ...

For an overview, see https://en.wikipedia.org/wiki/History_of_cryptography or read the famous book. "The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet".

5.1.2. Symetric

If both parties (usually called alice and bob) agree on the same secret, we can use a couple of encryption algorithms to encrypt our message.

We make the following basic assumptions:

1. Separate the algorithm from the key.
2. We can use a secure channel to transport the key.

Because the key is always shorter than the message there is, at least in theory, a potential break. The only algorithm, which can be proven to be non breakable, are "one time pads". Here we use a key with the same size as the message, and never use it again. If the key is generated from a **true random source**, this code can never be broken.

Some known algorithm include:

- DES (old because only 56 bit key length)
- 3DES
- ...
- AES

Exercises:

- Create a small text file hello.txt and encrypt the file with AES. Use the command `openssl aes256 -in hello.txt -out hello.txt.enc`.
- Verify, that the file is encrypted.
- Decrypt the file. Use `openssl aes265 -d -in hello.txt.enc`.
- Encrypt a new file and send it to your neighbour. Let him decrypt the file. Use the network drive scratch (Q:) for file exchange.
- Encode an other file and send it via e-mail to a colleague. Let him decrypt the file.

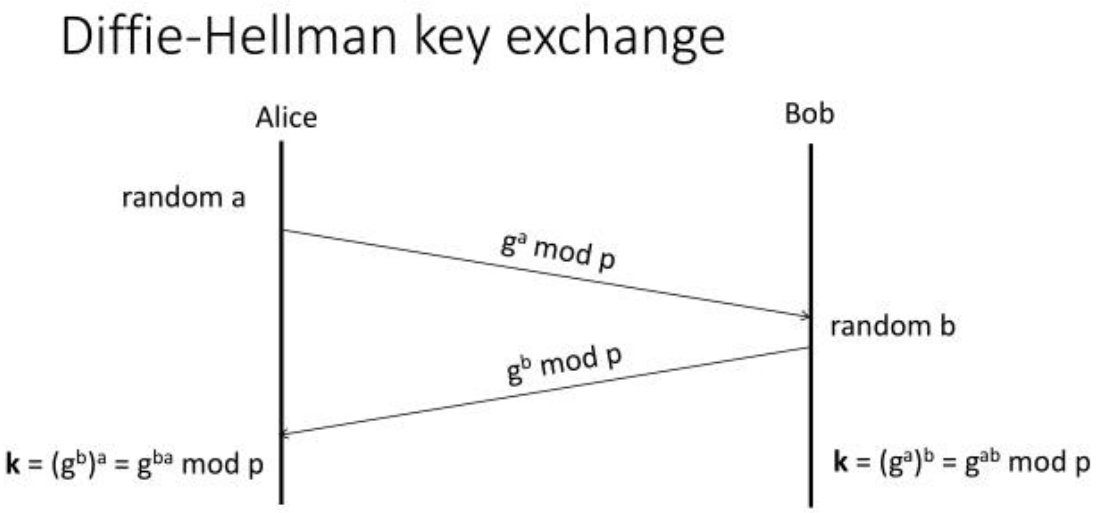
What was the biggest problem in these exercises?

5.1.3. Asymetric

We do not want to distribute keys. Instead we use a pair of keys, known as public and private key.

5.1.3.1. Diffie Hellmann Key Exchange

https://en.wikipedia.org/wiki/Diffie–Hellman_key_exchange.



*Both Alice and Bob have the same key **k**, without sending it on the network*

5.1.3.2. RSA-Algorithm

A famous example is the RSA-Algorithm.

1. Each party (A and B) generates a keypair: pub/priv.
2. The private key must be kept secret.
3. To encrypt a message, we use C=RSA(Message, Pub(B))
4. To decrypt a message, we use Message=RSA(C, Priv(B))

https://en.wikipedia.org/wiki/RSA_(cryptosystem).

It is interesting, that the algorithm works pairwise with the key. The reversed key usage is called "digital signature". We can use:

1. To sign a message, use Signature=RSA(Message, Priv(A))
2. To verify the signature, we check if Message=RSA(Signature, Pub(A))

Exercises:

- Create a new rsa keypair: `openssl genrsa -out my.key 2048`. View your file. It's normally highly recommended to encrypt your private key. If you want to do this, use the additional option `-aes256`.
- extract your public key from the file and place it in a new file called `yourloginname.pem`. Of course, replace "`yourloginname`". Use: `openssl rsa -in my.key -out yourloginname.pem -outform PEM -pubout`.
- Create a new file and encrypt it with your public key. `openssl rsautl -encrypt -inkey yourloginname.pem -pubin -in hello.txt -out hello.txt.enc`.
- Decrypt the file with your private key. `openssl rsautl -decrypt -inkey my.key -in hello.txt.enc`.
- Now exchange a encrypted message with your neighbour. Place your and your neighbours public key on drive Q. Of course **not** the private key! Encrypt the message with the correct recipients public key.

Advanced exercises: Unfortunately you cannot encrypt big files with RSA. To overcome this problem, you can do the following:

- Create a new temporary key file. `openssl rand -hex 64 -out key.bin`.
- Encrypt your large file with this key: `openssl aes256 -in largefile -out largefile.enc -pass file:key.bin`.
- Encrypt the temporary key with the recipients public key.
- Pass both files to the recipient.
- Decrypt the key.
- Decrypt the file.

What was the main problem with these exercises?

5.2. Hashes

As we saw in the chapter about git, hash function are use to compress a big binary into a limited sequence of character. Properties of good hash functions are:

1. It is (nearly) impossible to calculate the original document out of the hash value.
2. When a single bit changes on the input, the output is completely different.
3. The output numbers are evenly distributed among the numeric range.

Examples of hash functions:

- MD5
- SHA1
- SHA256

see also: https://en.wikipedia.org/wiki/Hash_function.

With openssl, you can create hash values from files. Example: `openssl dgst -sha256`, reads stdin and writes the hash to stdout.

Exercise, create a digital signature:

- Create a test file with some text.
- Create a hash from your file. Use `file.hash` as output.
- Sign the hash with your **private** key. `openssl rsautl -sign -inkey my.key -in file.hash -out file.sig`.
- Verify the signature with your public key. `openssl rsautl -inkey yourloginname.pem -pubin -in file.sig`. You should now see correct hash value.
- Create a signed file and send it to your neighbour. Place your public key, the original file and the encrypted hash on drive Q:
- What are the steps on the other end to verify the signature? Verify your neighbours signature.

We might encounter the same questions (problems) as in the previous exercise.

5.3. Certificates

Why use certificates? We want to make sure, that we use the correct recipients public key, which is not altered. The solution is to use a trusted third party, who signs a public key. Thus a certificate consists of:

- The users public key.
- Usage information.
- The digital signature of a trust center.

5.3.1. Workflow and Trusted Certificate Authority

Is is important to remember the flow of actions, when you use a certificate.

- A user create a public private key pair. Remember, the private key never leaves the users computer. Even better, if the private key never leaves a special hardware crypto device.

- The user create a request to sign the public key. This is called certificate signature request (CSR). This is a special file containing the users public key along with some additional information about the user. For use as an ssl certificate, the most important property is called "subjects common name". This property must exactly match the target servers domain name.
- The CSR is sent to a trusted third party, a "trusted certificate authority" (TCA). The TCA verifies the identity of the user (check passport etc) and then signs the csr with its own private key. The result certificate is returned to the user.
- To verify the validity of a certificate you must know the public key of the TCA. In windows, tese public keys are installed in the folder "Trusted Root Certification Authorities". Have a look at this folder. Start the utility "`certmgr.msc`".
- If the certificate was issued by an intermediate certification authority, you must install all certificates until you end up with on of the root certificates. This is called a "**certificate chain**".

5.3.2. File Formats

For certificates we find a couple of file formats:

- PEM, CRT, base64 coded certificate.
- KEY, base64 coded public private key pair.
- DES, binary code files.
- PFX, binary file, containing certificates (plural) and a private key. This is used for key chains.

5.3.3. Open SSL

Here are the important openssl commands for working with certificates.

- Create a public private keypair: `openssl genrsa -out server.key 2048`.
- To visualize your key, use: `openssl rsa -in server.key -noout -text`.
- Create a csr: `openssl req -new -key server.key -out server.csr`.
- To visualize your csr, use: `openssl req -in server.csr -noout -text`.
- If you use a config fle, use: `openssl req -new -key server.key -out server.csr -config server.cfg`.
- Send the CSR to your TCA and wait for it to be signed.
- The job of the TCA is more complicated, you normally don't do this: `openssl x509 -req -in server.csr -CA CA.crt -CAkey CA.key -CAcreateserial -out server.crt -days 365 -sha256 -extensions v3_req -extfile server.cfg`.
- Optional: Verify the validity of the certificate. Open it (click it). You might want to install it. With openssl you can use: `openssl x509 -in server.crt -noout -text`.
- Combine the certificate and your private key into one pfx file: `openssl pkcs12 -export -out server.pfx -in server.crt -inkey server.key`. This file is used for `SslStream` in the next sub chapter.
- You can generate a self signed certificate with: `openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365`. To trust this certificate, it must be installed in the "Trusted Root Certification Authorities" folder.

An exampled config file, might look like:

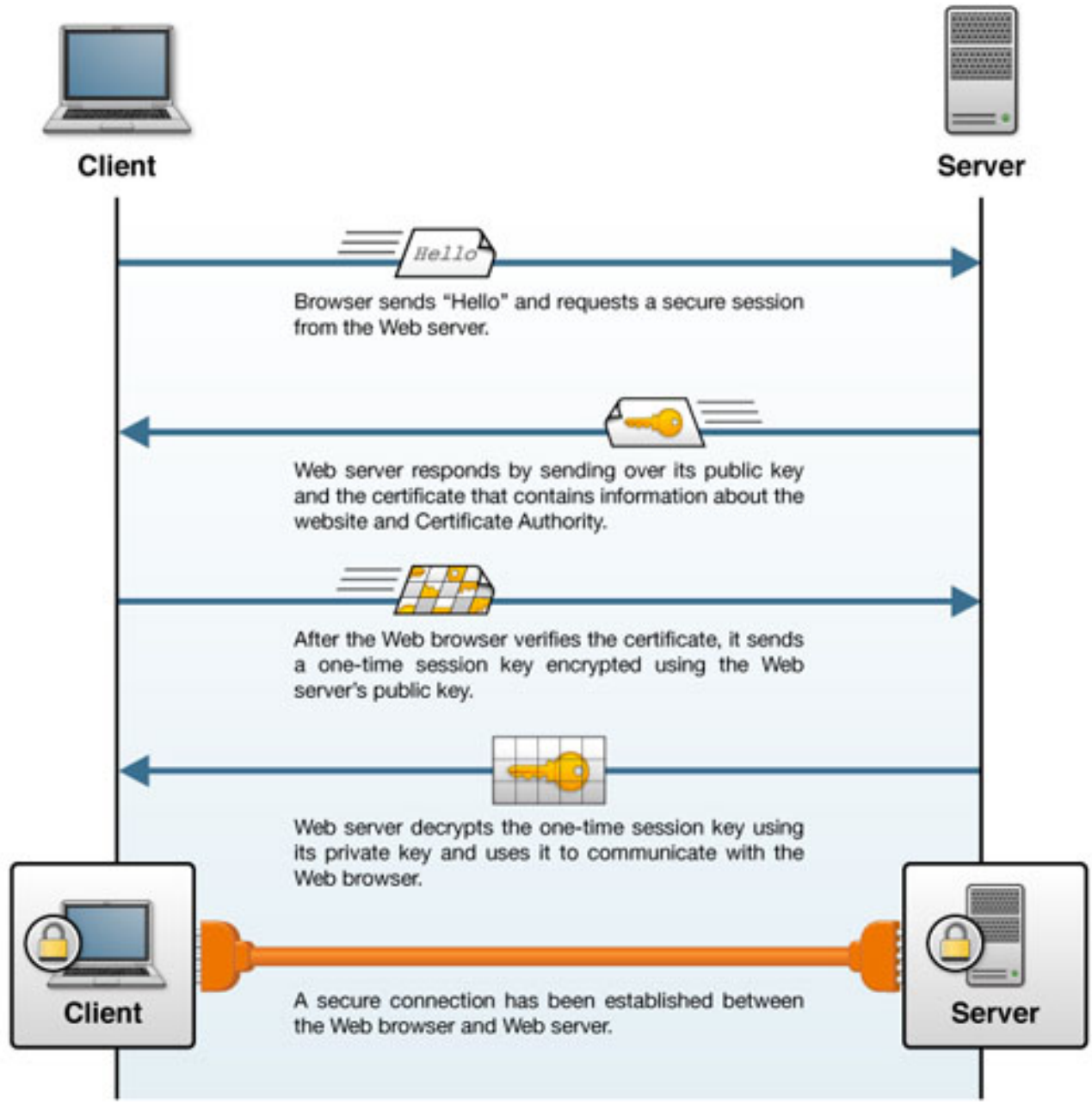
```
[req]
prompt = no
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
C = DE
ST = BW
L = Ulm
O = Hochschule Ulm
OU = Informatik
CN = server.informatik.hs-ulm.de
emailAddress = person@some.valid.email.de
```

```
[v3_req]
basicConstraints = critical, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
subjectAltName = @alt_names

[alt_names]
DNS.1 = server.informatik.hs-ulm.de
DNS.2 = serever
DNS.3 = localhost
DNS.4 = IP:127.0.0.1
DNS.5 = IP:[::1]
```

This file is important when you want to use different target names for your server. Put them in the alt_names section. You can put ip addresses as well. The latest security constraint requires to add an alt_names section even though you use only a single name.

5.4. SSL / TLS

The basic SSL / TLS workflow is (source: http://bartoz.no-ip.org/archives/3174):



5.4.1. Socket Example

Before creating a secured ssl connection, the first step you do is to create a certificate. For things to work, the client must either trust the certificate or disable verification. If these preconditions are met, you can use the class `SslStream`.

```
// SSL Client

using System;
using System.IO;
using System.Net.Security;
using System.Net.Sockets;
using System.Security.Cryptography.X509Certificates;

namespace Client
{
```

```
    class Program
    {
        static void Main(string[] args)
        {
            string host = "localhost";
            TcpClient cl = new TcpClient(host, 10001);
            Console.WriteLine("connected to server");

            SslStream ssl = new SslStream(cl.GetStream());
            ssl.AuthenticateAsClient(host);

            Console.WriteLine("authenticated as client");
            StreamReader rd = new StreamReader(ssl);
            StreamWriter wr = new StreamWriter(ssl);
            wr.WriteLine("hello from client");
            wr.Flush();
            string r = rd.ReadLine();
            Console.WriteLine(r);
            rd.Close(); wr.Close(); ssl.Close(); cl.Close();
        }
    }
}
```

```
// SSL Server

using System;
using System.IO;
using System.Net;
using System.Net.Security;
using System.Net.Sockets;
using System.Security.Cryptography.X509Certificates;

namespace Server
{
    class Program
    {
        static void Main(string[] args)
        {
            X509Certificate crt = new
            X509Certificate(@"your certificate path goes here", "yourpassword");
            TcpListener l = new TcpListener(new IPEndPoint(IPAddress.Any,
            10001));
            Console.WriteLine("server listening");
            l.Start();
            while (true)
            {
                TcpClient c = l.AcceptTcpClient();
                Console.WriteLine("connected");

                SslStream ssl = new SslStream(c.GetStream());
                ssl.AuthenticateAsServer(crt);
                Console.WriteLine("authenticated");

                StreamReader rd = new StreamReader(ssl);
                StreamWriter wr = new StreamWriter(ssl);
                String data = rd.ReadLine();
                Console.WriteLine(data);
                wr.WriteLine("message: " + data);
                wr.Flush();
                wr.Close(); rd.Close(); ssl.Close(); c.Close();
            }
            l.Stop();
        }
    }
}
```

5.4.2. HTTPS Example

(Demo)

5.5. Exercises

Before you begin, apply for a certificate.

5.5.1. Apply for Certificate

To apply for a certificate, do:

- Generate a new public private keypair
- Generate a CSR.
- Copy the csr file to `p:\traub\lect\usr\yourname`, where `lect = dwsys|vsys`.
- Retrieve the crt file from the same directory.
- Generate a pfx-file.

5.5.2. C# Encrypted Channel

Use the basic client server example. Add an additional `SslStream` variable, which covers the base stream.

5.5.3. HTTPS based Webserver

Extend the webserver demo and allow ssl connections.

5.5.4. Encrypted and Signed E-Mail

Apply for a certificate. Make sure to use your e-mail address as common name. After importing your certificate, send a signed e-mail to a colleague. You should now be able to reply with an encrypted mail.

If you want to encrypt the mail on first attempt, you need to (Outlook):

- Create a new contact.
- Import the corresponding certificate to the contact.
- Create an e-mail and make sure to select the contact from your list. Do not enter the e-mail address manually!

6. File-formats

For content encoding, there are a couple of options.

- Plain ASCII, properties are encoded as key value pairs. (see HTTP).
- SGML (HTML)
- XML (SOAP etc)
- JSON

These are all meta-languages, which means it defines that there are tags but not which tags. It defines the basic syntax as well.

6.1. XML

XML has now been around for a while. Originally invented in 1997 is now one of the most important meta-languages. Being a metalanguage means, XML defines that there are tags but not which tags.

6.1.1. Predesessor SGML

XML's predecessor is SGML. Defined as ISO standard in ISO 8879:1986, it defines how to use tags inside an ASCII document. Allowed tags are defined in a corresponding DTD (Document Type Definition). SGML is less restrictive than XML.

The most famous examples of a SGML file is HTML. This examples is a valid HTML/SGML Document, but not a valid XML document.

```
<!DOCTYPE html>
<html>
<BODY>
    Demo<p ID=10>Document.
</BODY>
</html>
```

6.1.2. XML Basics

XML is more restrictive. XML is case sensitive, while SGML is not. Attributes must be enclosed in quotation marks. An XML document must be well formed and may be valid.

Well formed means:

- There is a single root element. The processing instruction and whites paces are not counted.
- Every opening tag must be closed.
- Tags may not overlap.

XML must be well-formed in order to allow subsequent scripts to process the file. This is especially important because XML is normally not exposed to the end-user. An exception is XHTML.

XML can be used to describe nested data structures. A simple XML file look like:

```
<?xml version="1.0" encoding="utf-8"?>
<data>
  <person>
    <name>Petra</name>
    <age>24</age>
  </person>
  <person>
    <name>Hans</name>
    <age>12</age>
  </person>
</data>
```

This file describes an XML file containing a dataset with two persons. Here `<?xml...>` is called "processing instruction". Version is always 1.0. As you can see, XML allows you to specify the ASCII encoding. It is essential, that when you declare encoding "utf-8", you must use the same encoding when you save the file.

A good starting point for learning XML is: https://www.w3schools.com/xml/default.asp.

The basic building blocks are:

- Elements
- Attributes

There is no rule whether to attributes or elements. The main difference is, that attributes cannot defines nested structures. They are only strings.

Exercise: Create an XML file describing:

- Your address and phone numbers.
- A structured document.
- A remote function call.

6.1.3. Namespaces

When you combine two documents, there might be a name conflict on tag names. To overcome this problem, XML introduces namespaces. Each tag is associated with a namespace. All tags beyond a namespace declaration belong to this namespace. Namespaces are nothing else but unique names. These strings identify a namespace. We usually use the URL from our company to make sure nobody else uses the same namespace identifier.

The previous examples with namespace "`urn:test`" is written as:

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:test">
  <person>
    <name>Petra</name>
    <age>24</age>
  </person>
  <person>
    <name>Hans</name>
    <age>12</age>
  </person>
</data>
```

Try to combine a document with addresses from the previous exercise.

- Use direct `xmlns` declaration.
- Use `xmlns` prefixes.

6.1.4. Schema

To make an XML document valid, it must be compared against a schema specification. Most modern editors support this validation. A schema is an XML document with tag of namespace `http://www.w3.org/2001/XMLSchema`. Unfortunately, a schema declaration is a little bit complex. The XML-schema to the above example (without the namespace) might look like:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="data">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="person">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element name="age" type="xs:unsignedByte" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

As soon as you link the schema to your XML file you can use intellisense as well.

To link an XML document to a scheme, use:

```
<data xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="data.xsd">
...
```

Of course, change the filename accordingly. This is, unfortunately, a little bit complicated. Since the attribute "noNamespaceSchemaLocation" is not part of your data, it requires to define a namespace before its use. The namespace, only for the attribute, is "http://www.w3.org/2001/XMLSchema-instance".

To link a schema with a namespace, use:

```
<data xmlns="http://my-namespace"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://my-namespace schema-definition.xsd">
...
```

Even though XML-Schema is a little bit complex, it has a couple of advantages.

- XML-Schema is itself an xml-document, unlike the older DTD for example.
- You can add additional check, like restrictions.

Here is a more advanced example for the above document:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="data">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="person">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:pattern value="[A-Z][a-z]*"></xs:pattern>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="age">
                <xs:simpleType>
                  <xs:restriction base="xs:integer">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="120"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Exercises:

1. Write or reuse a simple XML file.
2. Let Visual Studio guess the schema.
3. Modify the schema a little bit. Limit the number of allowed persons.
4. Link your XML file to the schema. Check for intellisense support.

If Visual Studio does not complain about errors, you have a valid XML document.

6.1.5. Inclusions (XInclude)

One XML file may include an other XML file. This is expressed with `XInclude`. Example:

```
<xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
   href="foo.xml"/>
```

Attention: Declaring an `XInclude` does not actually do the include. Depending on your editor or language support you might need to do this by yourself.

6.1.6. Known XML Schema's

There is a huge number of XML schema's. Some examples:

- XHTML
- SVG
- Docbook
- XAML
- ...

6.1.7. Programming

One big advantage of XML is the wide support nearly every operating system and programming language. Let's use C# for XML-processing.

6.1.7.1. C#

Our goal is to send data-structures from client to server. In order to do this, we must convert an object into a serialized text string.

To serialize a data structure, use:

```
Person p = new Person() ....
XmlSerializer xs = new XmlSerializer(typeof(Person));
using (StreamWriter o = File.CreateText(xmlfile))
{ xs.Serialize(o, p); }
```

To deserialize it, use:

```
XmlSerializer xs = new XmlSerializer(typeof(Person));
using (StreamReader o = File.OpenText(xmlfile))
{ p = xs.Deserialize(o) as Person; }
```

Exercise:

- Create a data structure which will be sent from client to server. Create a class by your choice.
- Serialize an instance of your object and save it to a file. Look at the file.
- Instead of saving the XML code to a file, use TcpClient and send it to your server.
- Read the XML from your file, deserialize it and print it out.
- Receive the XML from your client instead of reading it from a file.

Additional exercises:

1. Open on of your XML files. Use one without a namespace.
2. Recursively traverse your file and print it to the console.
3. Add a new element with attributes and save the file.
4. Add a new element with a different namespace. Save and inspect the file.

6.1.7.2. Powershell

Reading an XML file in PowerShell is quite easy. It is a simple as:

```
$d = [xml](Get-Content MyXmlFile.xml)
```

After that, you can directly access the properties of the XML. Examples:

```
$d.data.person[0].name
```

6.1.8. XPath and XSLT

One big advantage of XML is, that it can be easily transformed into a target format. In principle, this can be done with any programming language. Because it is so common, there exists a special transformation processor: XSLT. It is mainly constructed using a pattern matching algorithm. A pattern is expressed by an XPath expression. When a match is found, a new output is generated.

An XPath is a "path" inside an XML document. In our example, the first person of our XML-file can be selected as:

```
/data/person[1]/name
```

Selects the data-nodes, then the first person-node and finally the name-node. Note: the indices are 1-based, not 0-based.

For a reference and tutorial, see at: https://www.w3schools.com/xml/xpath_intro.asp.

To checkout XPath expressions, you can use the PowerShell commandlet `Select-Xml`.

The following command applies the XPath expression to a file:

```
Select-Xml -Path data.xml -XPath "/data/person[1]/name"
```

If there is a namespace within your XML document, you want to write:

```
Select-Xml -Path datas.xml -XPath "/x:data/x:person[1]/x:name" -Namespace @{x="urn:test"}
```

Exercises:

- Create an example XML file with some data.
- Select part of the document an print the result.

An XSLT file is basically a collection of template matching expression. When one of the expression applies, the containing document is copied to the output.

This is a simple XSLT file, which extracts the first person looks like the following snippet.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl">

    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="/data">
      <out>
        <info>
          <xsl:value-of select="person[1]/name"/>
        </info>
      </out>
    </xsl:template>
</xsl:stylesheet>
```

Unfortunately, Visual Studio Community Editions has no support for XSLT, thus we use the command-line:

```
msxsl data.xml tranform.xslt -o result.xml
```

Or with powershell:

```
$tr=new-object System.Xml.Xsl.XslCompiledTransform
$tr.load("tr1.xslt")
$tr.Transform("data.xml","out.xml")
```

Here is a short list of important XSLT statements:

- `value-of select="node-expr"`: Select a node from input.
- `apply-templates select="xpath"`: Goon with patteren mataching unsing the XPath expression.
- `for-each select="xpath"`: Iterate over a node-set.
- `attribute`: Create a new attribute on output tree.
- `if`: Test if a condition is met.
- `call-template`: Go direct to an other template without any new pattern search.

For a reference and tutorial, see: https://www.w3schools.com/xml/xsl_intro.asp, and https://www.w3schools.com/xml/xsl_elementref.asp.

Exercises:

- Create a simple XML file.
- Use the identity transformation (generated file) from Visual Studio to generate an output file.
- Create an output-file, containing all persons from our examples but with different tags.
- Create an xhtml output.

6.2. JSON

The "JavaScript Object Notation" is much easier compared to XML. It gained popularity because of the widely used language javascript. As the language javascript, it lacks type-support as well.

6.2.1. Basics

The file format is as simple as the serialized version of a javascript object. Examples (with nodejs):

```
var o = new Object()
o.name="Hans"
o.age=12
JSON.stringify(o)
```

The output is:

```
{"name":"Hans","age":12}
```

Or the reverse operation (deserialize):

```
var x = JSON.parse('{"name":"Hans","age":12}')
console.log(x.name)
```

6.2.2. Objects

As we saw, objects are simply described by key-value pairs enclose in curly braces. Of course, objects can contains objects:

```
{ "sister": { "name": "Petra", "age":12 }, "brother": { "name":"Peter", "age":14 }}
```

6.2.3. Arrays

Arrays are denoted by square brackets. This is an example:

```
[1,2,3,4,"end"]
```

Let's rewrite our person a JSON object:

```
{
    "data" : [
        {"person" : {
            "name" : "Petra",
            "age": 21
        }},
        {"person" : {
            "name" : "Hans",
            "age" : 23
        }}
    ]
}
```

Array elements need not to have the same type.

As you see, JSON is very easy, but lacks true type-support. Currently there is an attempt to attach a schema to JSON as well. We can write a JSON schema for our person as (schema.json):

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "myschema",
    "title": "My Data",
    "type": "object",
    "properties": {
```

```
        "data" : {
            "type" : "array",
            "items": {
                "type" : "object",
                "properties": {
                    "person" : {
                        "type" : "object",
                        "properties": {
                            "name" : {
                                "type" : "string"
                            },
                            "age" : {
                                "type" : "integer"
                            }
                        },
                        "required": ["name", "age"]
                    }
                }
            }
        }
    }
}
```

We refer to this schema within our file with the attribute:

```
"$schema": "./schema.json"
```

We might also want to further restrict the possible attribute values as before. We can so so by using these snippets.

```
...

"person" : {
    "type" : "object",
    "properties": {
        "name" : {
            "type" : "string",
            "pattern": "[A-Z][a-z]*"
        },
        "age" : {
            "type" : "integer",
            "minimum": 0,
            "maximum": 120
        }
    },
    "required": ["name", "age"]
}

...
```

For further reference, see: http://json-schema.org/.

6.2.4. Programming

Of course, JSON is best handled with javascript. Since we did not introduce javascript so far, let's do some programming in C#.

6.2.4.1. C#

C# has no builtin support for JSON, but there is a nuget package called: "Newtonsoft.Json". Right-click your project, choose "manage nuget packages" and select this package. In Dotnetcore use: `dotnet add package Newtonsoft.Json`.

First, create a C# class, which represents your object. After you created an instance, you can serialize it:

```
String json = JsonConvert.SerializeObject(f);
```

After that, try to deserialize it from string or file:

```
JObject r = JsonConvert.DeserializeObject(json) as JObject;
Console.WriteLine(r["sister"]["name"]);
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json;

public class Person
{
    public string name;
    public int age;
}

class Program
{
    static string jsonfile = @"c:\temp\test.json";
    static void Main(string[] args)
    {
        Person p = new Person() { name = "Hans", age = 12 };
        String s = JsonConvert.SerializeObject(p);
        Console.WriteLine(s);

        File.WriteAllText(jsonfile, s);

        String t = File.ReadAllText(jsonfile);
        Person q = JsonConvert.DeserializeObject(
            s, typeof(Person)) as Person;
        Console.WriteLine(q.name);
    }
}
```

You can do the same in a more dynamic way:

```
dynamic p = JsonConvert.DeserializeObject(File.ReadAllText("test.json"));
System.Console.WriteLine(p.data[0].person.name);
```

6.3. Client Server Exercise

Create a client server application using sockets or the TcpClient class. Do the following steps:

- Send a person record from the client to the server.
- Print the person on the server.
- Add a "printout" option to your message. For example when you send "full" print the full record, when you send "name" print the name of the person only.

6.4. HTML and XHTML

HTML is used for coding a content-page in the world wide web.

6.4.1. SGML and XML

HTML is based on SGML as meta language. The HTML schema is defined by a Document Type Description (DTD). At last, this was true until HTML 4.01. See: https://www.w3.org/TR/html401/sgml/dtd.html.As with HTML5 there is no official DTD.

XHTML is basically the same as HTML, only that is is based on XML as meta language. For XHTML there is a DTD (https://www.w3.org/TR/xhtml1/dtds.html#a_dtd_XHTML-1.0-Strict).

If you design a new HTML-page is is a good idea to use XHTML instead of HTML.

1. Every browser understands SGML and XML.
2. Is is much easier to transform and manage an XML document, because of the widespread use of XML parsers.
3. The syntax is cleaner.
4. It is easy to embed documents from other namespaces, like SVG.

6.4.2. Basic Setup end Encoding

A basic HTML page looks like:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<p>Hello HTML World.</p>
</body>
</html>
```

Make sure to define the correct encoding. Save your file with the encoding defined in the header.

6.4.3. Exemplary Tags

Here are a couple of exemplary HTML tags:

- h1: Define a heading.
- p: Define a paragraph.
- img: Load an image.
- div: Create a block.
- ...

Here is an example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Demo</title>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <h1>HTML Demo</h1>
    <p>This is a paragraph.</p>
    <img src="logo.gif">

    <div>
    <a href="www.hs-ulm.de">Hochschule Ulm</a>
    </div>
</body>
</html>
```

Since this is not a class for HTML programming, please refer to https://www.w3schools.com/html/default.asp for further reference:

6.4.4. CSS

In order to separate content and layout, it is very common to define all style settings in a separate CSS file. Here is a small example.

```
body {
    font-family: sans-serif;
}

h1 {
    font-size: 16pt;
}

a {
    text-decoration: none;
    color: red;
}
```

For further reference see: https://www.w3schools.com/css/default.asp.

6.5. Exercises

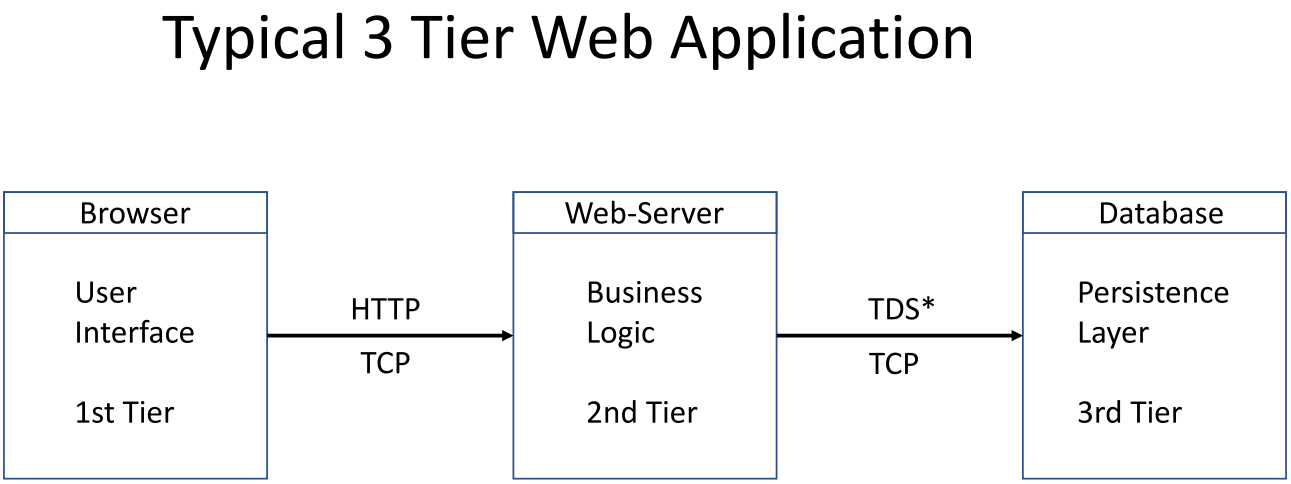
Create a simple HTML document and link a stylesheet. Change some style settings.

7. World Wide Web

The world wide web is obviously one of the most important applications on the internet.

7.1. System Model

Let's first extend our basic client server model to a three tier application.



* Tabular Data Stream (SQL-Server), may also be REST, MySQL, CIFS, NFS ...

7.2. Basics

The world wide web consists of the following building blocks.

- HTML(based on SGML) for page encoding.
- URI for referencing a resource.
- HTTP for accessing a resource.

7.2.1. HTML and CSS

I will not create my own tutorials within this script because there are a lot of very good online tutorials available. One of them can be found at: https://www.w3schools.com/.

To learn HTML use: https://www.w3schools.com/html/default.asp.

I recommend to use these examples:

- Basic: https://www.w3schools.com/html/html_basic.asp
- Elements: https://www.w3schools.com/html/html_elements.asp
- Attributes: https://www.w3schools.com/html/html_attributes.asp
- Heading: https://www.w3schools.com/html/html_headings.asp
- Paragraphs: https://www.w3schools.com/html/html_paragraphs.asp
- Links: https://www.w3schools.com/html/html_links.asp
- Lists: https://www.w3schools.com/html/html_lists.asp
- Tables: https://www.w3schools.com/html/html_tables.asp
- Images: https://www.w3schools.com/html/html_images.asp (different image formats)
- Styles: https://www.w3schools.com/html/html_styles.asp
- CSS: https://www.w3schools.com/html/html_css.asp

To format documents, we use CSS.

- Selectors: https://www.w3schools.com/css/css_selectors.asp
- Colors: https://www.w3schools.com/css/css_colors.asp (use correct semantics)

- Height and Width: https://www.w3schools.com/css/css_dimension.asp
- Borders: https://www.w3schools.com/css/css_border.asp
- Box Model: https://www.w3schools.com/css/css_boxmodel.asp
- Fonts: https://www.w3schools.com/css/css_font.asp
- Tables: https://www.w3schools.com/css/css_table.asp
- Position: https://www.w3schools.com/css/css_positioning.asp
- Display: https://www.w3schools.com/css/css_display_visibility.asp

Modern web application target different types of output devices from small mobile phones to big wide screen desktop computers. It can be very hard and cumbersome to implement the correct CSS styles for those different types of devices. The bootstrap framework can help us a lot. It is included in a lot of project templates as well by default.. Try the following examples:

- Getting Started: https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp
- Grind: https://www.w3schools.com/bootstrap4/bootstrap_grid_basic.asp
- Buttons: https://www.w3schools.com/bootstrap4/bootstrap_buttons.asp
- Tables: https://www.w3schools.com/bootstrap4/bootstrap_tables.asp
- Navigation: https://www.w3schools.com/bootstrap4/bootstrap_navbar.asp

7.2.2. Uniform Resource Identifier, URI

A resource is identified by URI. Basically there are two types of a URI.

- URL, a uniform resource locator. It points to a DNS "Location".
- URN, a uniform resource name. This is a globally unique name, which needs to be resolved into a URL for further reference.

A complete URL looks like:

```
protocol://user:password@servername:port/resource/name?querystring
```

7.2.3. HTTP

History:

- HTTP 0.9, very basic
- HTTP 1.0, adds support for metadata
- HTTP 1.1, adds host information and persistent tcp as default

The basic http protocol is quite easy. After connection establishment, the client sends an http request as follows:

```
GET /resource/name HTTP/1.x
Accept: */*
User-Agent: mybrowsertype

body content
```

The request consists of:

1. The request line with the http verb. In this example "GET".
2. Multiple lines of key-value pairs for additional metadata.
3. An empty line.
4. Possible content in any encoding, dependent of the HTTP verb. For example, a form content may be sent that way.

After a successful request the server responds with:

```
HTTP/1.1 200 OK
Content-Type: text/html

Content ...
```

The response is similar to the request. It consists of::

1. The status line, 200 means ok.
2. Multiple lines of key-value pairs for metadata. The most important is "content-type", which defines the datatype of the following resource.
3. An empty line.
4. The content of the resource being requested. This could be binary data as well.

7.2.3.1. Verbs

HTTP defines a number of so called "verbs" or "operations". Here is the list:

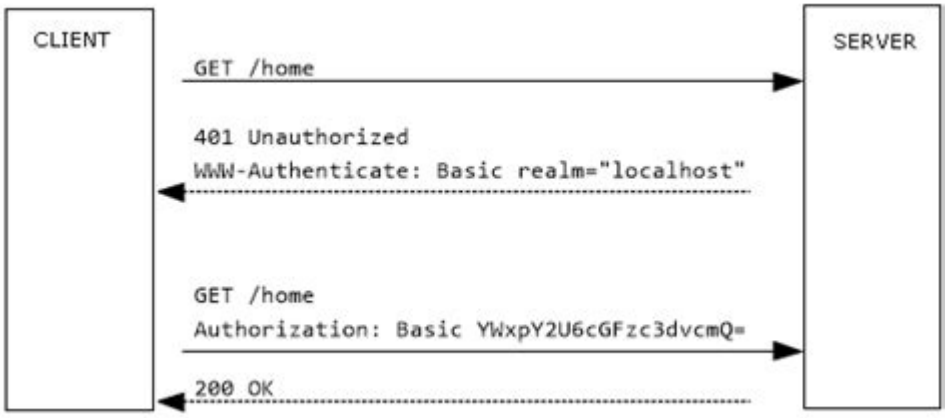
- GET, load a resource.
- HEAD, load only metadata of a resource.
- POST, reply to a form. Send the forms content. (In REST APIs this is used for creating a resource)
- PUT, store a new resource. (In REST APIs this is used for update)
- DELETE, delete a resource.
- OPTIONS, list options about a resource.
- CONNECT, connect to a tunnel.
- TRACE, check an access path to the target resource.
- PATCH, modify partial content of a resource.

7.2.3.2. HTTP Authentication

Of course, a user might need to be authenticated at the server. To do this, we have a couple of options:

- Form based authentication. This is NOT a HTTP authentication! This is just a form that is sent to the server, with which it can do everything, including authentication on the server side.
- HTTP Basic Authentication. Password is sent base64 codes, which is clear text!
- HTTP Digest / NTLM / Kerberos Authentication.
- Client Certificates

The typical message flow with basic authentication looks like:



Exercise: Run our test webserver, enter a username and password. Decode the response. Use the command `openssl base64 -d`. Pipe the response string to standard input.

Since basic authentication sends credentials as clear text, encrypt your traffic!

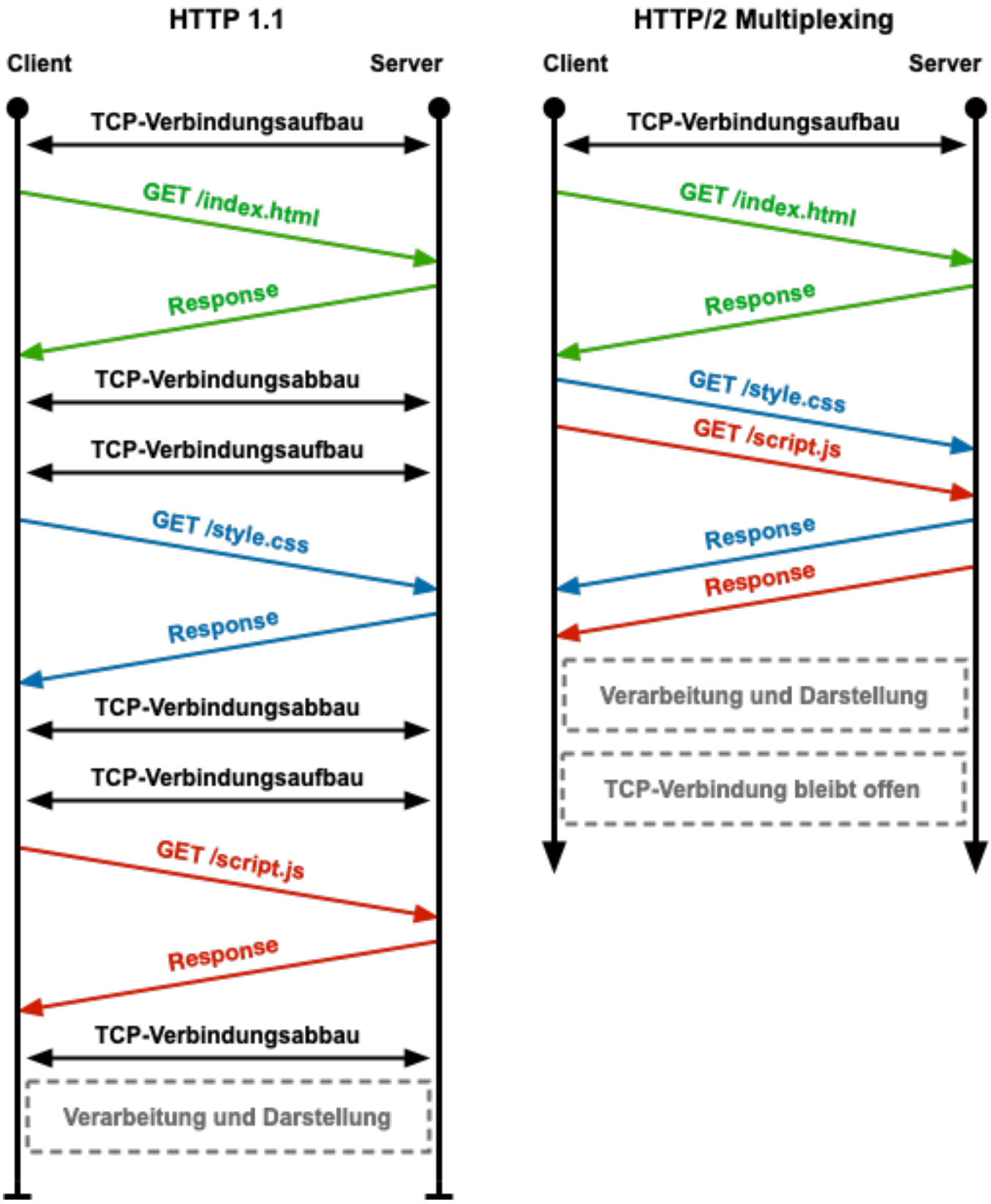
```
// ======================================================
//  Server Implementation Demo
// ======================================================

using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

```
using System.Threading.Tasks;

class Program
{
    static int Port = 10000;
    static IPAddress Addr = IPAddress.Any;

    static string Response()
    { return "HTTP/1.0 200 OK\r\nContent-Type: text/plain\r\n\r\ntest " +
    DateTime.Now; }

    static string ForceAuth()
    { return
    "HTTP/1.0 401 Unauthorized\r\nWWW-Authenticate: Basic realm=\"Tesserver\"\r\n\r\n"; }

    private static void Serve(TcpClient cl)
    {

        Console.WriteLine($"client: {cl.Client.RemoteEndPoint.ToString()}");
        using (StreamWriter wr = new StreamWriter(cl.GetStream()))
        using (StreamReader rd = new StreamReader(cl.GetStream()))
        {
            List<String> cmd = new List<string>();
            do {
                cmd.Add(rd.ReadLine());
            } while (cmd[cmd.Count-1]!="");

            String auth = cmd.Find(x => x.StartsWith("Authorization"));
            if (auth == null) {
                wr.WriteLine(ForceAuth());
            } else {
                wr.WriteLine(Response()+"\r\n"+auth);
            }
            wr.Flush();
        }
        cl.Close();
    }

    static void Main(string[] args)
    {
        TcpListener server = new TcpListener(Addr, Port);
        server.Start();
        while (true)
        {
            TcpClient cl = server.AcceptTcpClient();
            Serve(cl);
        }
    }
}
```

7.2.4. HTTP/2

While keeping the semantics of the protocol version 1.1 the protocol version #2 is an improvement because it adds the capability to multiplex multiple TCP streams within a single TCP connection which improves the response time when you access multiple resources from a website. To improve the performance a little bit further it uses a binary compression of the header meta information and the content as well.

Another advantage is that you don't have to modify your application at all, you simply must switch on HTTP/2 support at your server. Most of the modern browsers already support this protocol version.



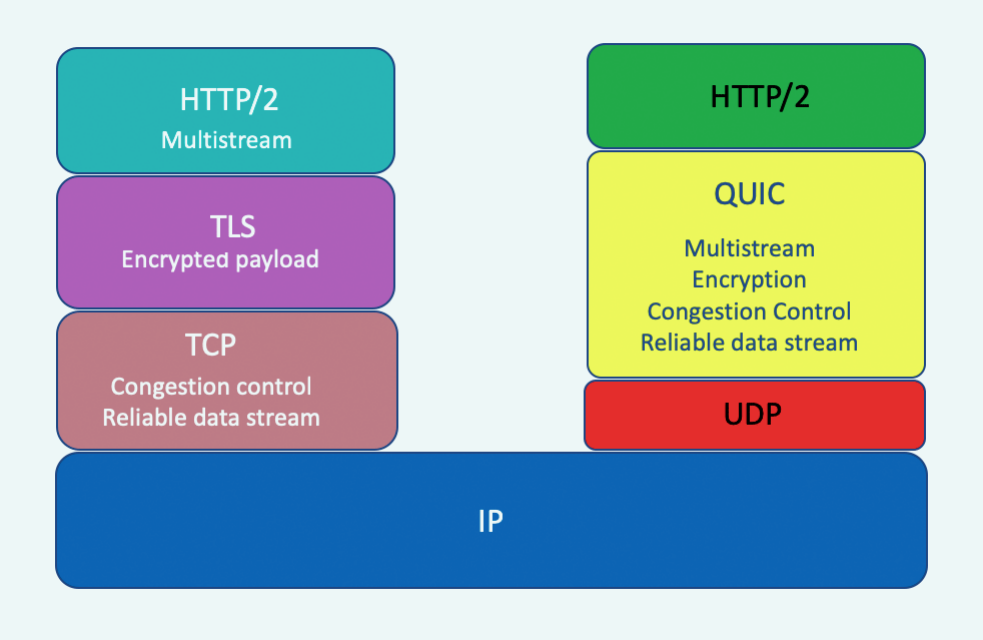
Source: elektronik-kompendium.de : https://www.elektronik-kompendium.de/sites/net/bilder/18073111.png

7.2.5. HTTP/3

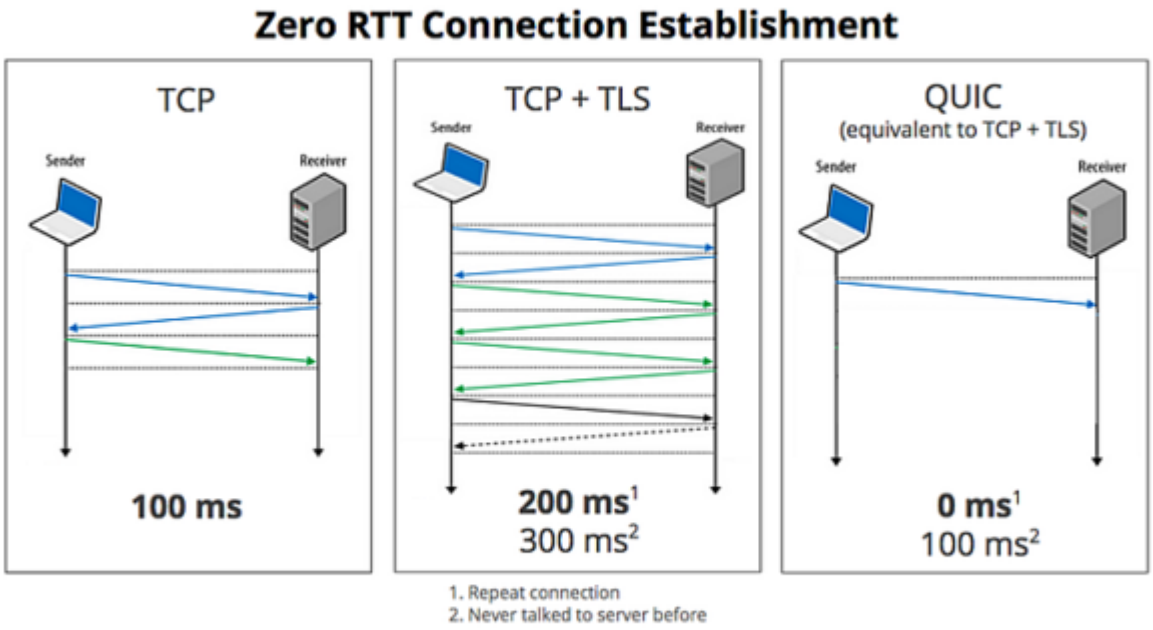
HTTP version 3 is yet another extension to the existing protocol. In this case the existing TCP IP protocol stack is replaced by a new protocol called "QUIC" based on UDP as transport media and a completely new implementation of a connection oriented transport level protocol.

There are couple of advantages of using "QUIC" as a new transport protocol. One of them is that the connection establishment takes fewer packets than the typical 3 way handshake from TCP. Another advantage is that "QUIC" implements data encryption by default , which means it does not require an extra TLS layer.

When you use HTTP version 2 or version 3, the semantics of the application layer protocol HTTP is preserved. This means you don't have to change your application logic at all.



Source: https://circleid.com/posts/20190304_a_quick_look_at_quic



Source: zdnet.com

7.3. Applications

7.3.1. The Browser

7.3.2. The Webserver

7.3.2.1. Implement You Own Server

An an exercise, let's create a small webserver.

7.3.2.1.1. Simple Web-Server

The HTTP protocol is very easy. See the following message flow:

The client sends:

```
GET / HTTP/1.0

```

Please note the extra empty line at the end! The end of line is encoded as `cr+lf` (`"\r\n"`).

The server should respond as follows:

```
HTTP/1.0 200 OK
Content-Type: text/plain

Textfile from server ...
```

Again: watch out for the empty line!

If you want to send HTML content, use `Content-Type: text/html`.

```
// ============================================================
//  Web-Server Implementation Demo
// ============================================================

using System;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static int Port = 10000;
    static IPAddress Addr = IPAddress.Any;

    // This generates the web response
    static string Response()
    {
        return
        "HTTP/1.0 200 OK\r\nContent-Type: text/plain\r\n\r\nUhrzeit: "
                + DateTime.Now;
    }

    // This reads a file an generates the correct response
    static byte[] Response(string file)
    {
        byte[] content;
        byte[] header;
        String fname = @"..\..\..\"+file.Substring(1);
        String ctype = "text/plain";

        // Set the correct content type, currently we only support htm and
        gif
        switch (file.Substring(file.LastIndexOf(".")+1))
        {
            case "htm": ctype = "text/html"; break;
            case "gif": ctype = "image/gif"; break;
        }

        try // to read the file
        {
            content = File.ReadAllBytes(fname);
        } catch(Exception ex)
        {
            content = Encoding.Default.GetBytes($"{ex.Message}: {fname}");
        }

        // set the header
        header =
        Encoding.Default.GetBytes($"HTTP/1.0 200 OK\r\nContent-Type: {ctype}\r\n\r\n");
        return header.Concat(content).ToArray();
    }

    // Serve a single client
    private static void Serve(TcpClient cl)
    {

        Console.WriteLine($"client: {cl.Client.RemoteEndPoint.ToString()}");
        using (StreamWriter wr = new StreamWriter(cl.GetStream()))
        using (StreamReader rd = new StreamReader(cl.GetStream()))
        {
            String cmd = rd.ReadLine();
            Console.WriteLine(cmd);

            string url = cmd.Split(' ')[1];
            if (url == "/") // The homepage is our local clock
            {
```

```
                Thread.Sleep(2000);
                wr.WriteLine(Response());
                wr.Flush();
            }
            else // Handle regular files
            {
                byte[] data = Response(url);
                cl.GetStream().Write(data, 0, data.Length);
            }
        }
        cl.Close();
    }

    // single threaded implementation
    static void web1()
    {
        TcpListener server = new TcpListener(Addr, Port);
        server.Start();

        while (true)
        {
            TcpClient cl = server.AcceptTcpClient();
            Serve(cl);
        }
    }

    // multi threaded implementation
    static void web2()
    {
        TcpListener server = new TcpListener(Addr, Port);
        server.Start();

        while (true)
        {
            TcpClient cl = server.AcceptTcpClient();
            (new Thread(() => Serve(cl))).Start();
        }
    }

    // thread pool implementation
    static void web3()
    {
        TcpListener server = new TcpListener(Addr, Port);
        server.Start();
        ThreadPool.SetMaxThreads(5, 5);
        while (true)
        {
            TcpClient cl = server.AcceptTcpClient();
            ThreadPool.QueueUserWorkItem((c) => Serve((TcpClient)c), cl);
        }
    }

    // asynchronous Serve method
    private async static void ServeAsync(TcpClient cl)
    {

        Console.WriteLine($"client: {cl.Client.RemoteEndPoint.ToString()}");
        using (StreamWriter wr = new StreamWriter(cl.GetStream()))
        using (StreamReader rd = new StreamReader(cl.GetStream()))
        {
            String cmd = await rd.ReadLineAsync();
            await Task.Delay(2000);
            await wr.WriteLineAsync(Response());
            await wr.FlushAsync();
        }
        cl.Close();
    }

    // asynchronous implementation
    static void web4()
    {
        TcpListener server = new TcpListener(Addr, Port);
        server.Start();
```

```
        while (true)
        {
            TcpClient cl = server.AcceptTcpClient();
            ServeAsync(cl);
        }
    }



    static void Main(string[] args)
    {
        web1();
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Net.Sockets;
using System.Threading;

class Program
{
    static void Test(Object o)
    {
        TcpClient cl = new TcpClient("localhost", 10000);
        using (StreamWriter wr = new StreamWriter(cl.GetStream()))
        using (StreamReader rd = new StreamReader(cl.GetStream()))
        {
            wr.WriteLine("GET / HTTP/1.0\r\n");
            wr.Flush();
            string r = rd.ReadToEnd();
            Console.WriteLine(r);
        }
    }

    static void Main(string[] args)
    {
        Stopwatch w = new Stopwatch();

        w.Start();
        int N = 10;
        List<Thread> tl = new List<Thread>();

        for (int i = 0; i < N; i++)
        {
            Thread t = new Thread(Test);
            tl.Add(t);
            t.Start(i);
        }

        for (int i = 0; i < N; i++)
        {
            tl[i].Join();
        }
        w.Stop();

        Console.WriteLine("elapsed time: " + w.ElapsedMilliseconds +
        " msec");
    }
}
```

7.3.2.1.2. Single Threaded Server

Start with a simple single threaded implementation.

- Create a method `Respond` which generates the output string (see previous section). Respond with the current date and time. Add a delay of 2 seconds at the end of this method in order to simulate a long running server action.

- Create a method `Serve` which handles an incoming connection. Use `TcpClient` as parameter. This method creates a `StreamReader` and `StreamWriter` based on the TcpClient argument. Then, read a line and send the response. Do not forget to `flush` the output.
- In the main method, create a `TcpListener`, which listens on any ip address and port 10000. Start the listener and accept incoming connection. After accepting a connection, call your `Serve` method. Don't create any threads yet.
- Add a couple of debug messages, so that you know whats going on.
- Check if your app is running with a browser. What is the url?

7.3.2.1.3. Test the server

After successfully connection to your server using a browser, create a simple client.

You can implement this client in C# or Powershell. Invoke your test.

In PowerShell this can be done with:

```
(Invoke-WebRequest http://localhost:10000).Content
```

Now let`s see what happens if we call the server with multiple threads. Extend your application and call the server 10 times.

In powershell, you can use:

```
(1..10 | % { start-job { Invoke-WebRequest http://localhost:10000 } } | wait-job | receive-job).Content
```

This is a single line!

Notice the overall response time.

7.3.2.1.4. Concurrent Server

Now change your server implementation.

- Create a new thread for each incoming client. You can use an explicit thread method or use a lambda expression.
- Check again the overall response time.

7.3.2.1.5. Thread Pools

If you expect a large number of client, you can use the typical client dispatcher model. Instead of creating a new thread for each client you can use thread pools. Use the method `ThreadPool.QueueUserWorkItem` to queue a new incoming client to the pool. The argument is a delegate to your worker method. You can limit the number of working threads with `ThreadPool.SetMaxThreads(5, 5);`. This limits the worker threads to 5. Again check the overall response time.

7.3.2.1.6. Asynchronous Sever

Now go back to the first implementation. Don's use any threads.

- Make your `Serve` method an asynchronous method. Make sure to use asynchronous calls inside your method as well. Otherwise you will not see any improvements.
- Again check the overall response time.

7.3.2.1.7. Extend your Sever

Now try to extend your server implementation. Allow it to serve files instead of a single page. Check what is different when a browser refers to a file instead of the homepage. Parse the request and serve an html-file. A simple HTML page lokks like:

```
<html>
<body>
    <h1>Demo</h1>
    <p>Testpage.</p>
```

```
</body>
</html>
```

Try to add a picture to your page:

```
<html>
<body>
    <h1>Demo</h1>
    <p>Testpage.</p>
    <img src="logo.gif">
</body>
</html>
```

7.3.3. The Persistence Layer

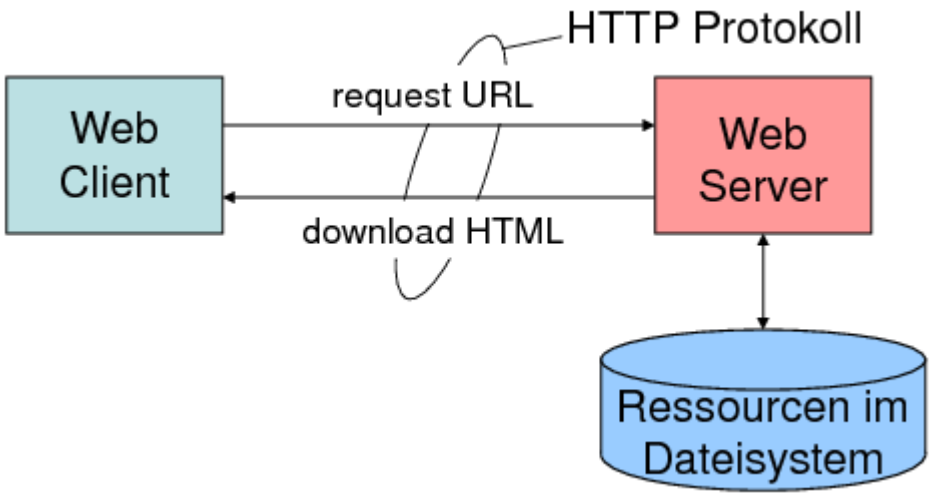
7.4. Web-Applications

Initially, the web was defined to serve static pages. Over the years, there were lots of frameworks being developed. In principle, we can distinguish between client-side-code and server-side-code.

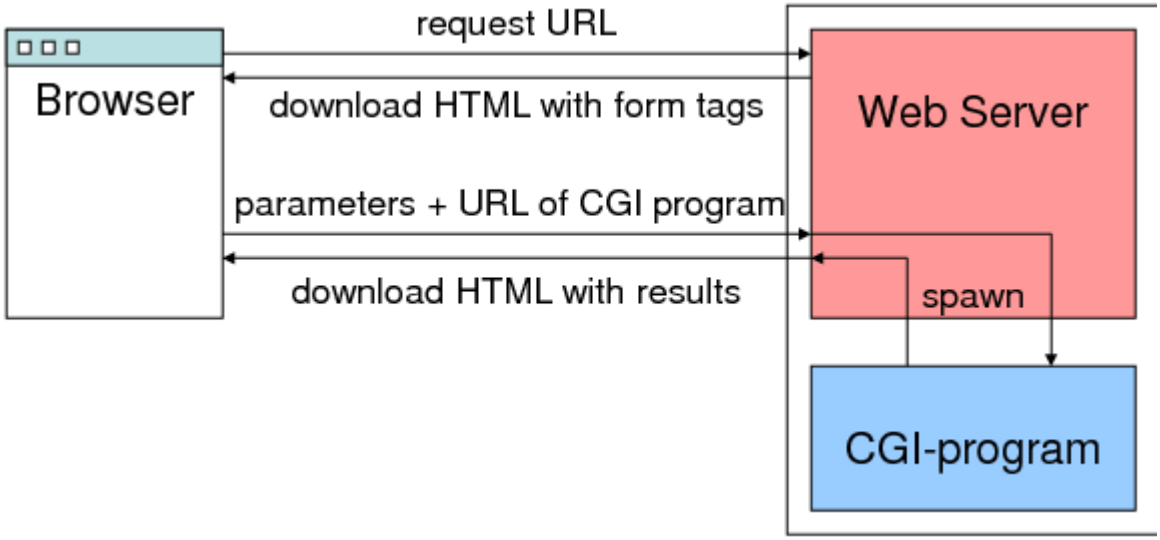
7.4.1. Server-Side-Code (CGI)

The basic idea is as follows: Instead of reading a static file, the web server invokes an external program or script. The standard output of the external process is redirected to the TCP-channel. Now the process can create any desired HTML content. Additional information is passed to the process by means of environment variables. For examples, everything after the "?" is put into the environment variable "QUERYSTRING".

Read static files:



Invoke CGI script:



All other web frameworks are merely modifications of the basic scheme.

In principle, we can distinguish three generations of web frameworks. In all of these approached, the browser sees only pure html.

1. Generation 1 runs any executable and redirects the stdout to the browser. You have to create anything in code. If something has to be changed, the complete code must be recompiled. Examples: Any executable, java-servlets.
2. Generation 2 intermixes HTML code and app code. The code is executed in place when the page is requested. The might result in unreadable code. Examples: JSP, PHP.
3. Generation 3 tries to separate the code from the layout. Only placeholders are put in the HTML template. After the code completes the placeholders are rendered into HTML. Examples: JSF, ASP.Net, Template-Frameworks.

There are a lot of frameworks, which supports CGI programming. Examples are:

- ASP.Net, Razor-Pages, Blazor-Pages (C#)
- Laravel (PHP)
- Django (Python)
- Spring (Java)
- Ruby on Rails (Ruby)
- Express (Javascript)
- Meteor (Javascript)

7.4.2. Stateless HTTP and Sessions

For good reasons the HTTP protocol is designed as a stateless protocol. This means when a HTTP transaction completes the server forgets everything about the current client transaction. The advantage of using this kind of protocol is that it scales very well.

The problem is, that with a stateless protocol it's not that easy to implement stateful client applications which require, for example access a client state to implement a shopping cart. Bottom of the line is, because you can't store any client information on the server side you store any client information on the client browser. To do this we have basically three options:

- Cookies
- Hidden Fields
- URL Rewriting

7.4.2.1. ASP.Net core

To illustrate this let's implement a very simple counter application which increments a counter whenever you click on a button. the following examples are implemented using asp.net core.

First the basic setup.

7.4.2.1.1. Basic Setup

Startup.cs

```
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddSession();
            services.AddRazorPages();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            { app.UseDeveloperExceptionPage(); }

            app.UseRouting();
            app.UseSession();

            app.UseEndpoints(endpoints =>
            { endpoints.MapRazorPages(); });
        }
    }
```

Index.cshtml

```
@page
@model coreweb.Pages.IndexModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<html xmlns="http://www.w3.org/1999/xhtml">
<body>
    <form asp-page-handler="button" method="post">
        <button>click me</button>
    </form>
    <div>@Model.Msg</div>
</body>
</html>
```

7.4.2.1.2. Events

This is how we can react on events.

Index.cshtml.cs

```
using Microsoft.AspNetCore.Http;

namespace coreweb.Pages
{
    public class IndexModel : PageModel
    {
        public string Msg { get; set; }

        public void OnPost()
        {
            Msg = "click";
        }

    }
}
```

7.4.2.1.3. Wrong Implementation 1

This first implementation does not work because the server creates a new object for each incoming HTTP request which means the counter is going to be initialized with zero for each request.

```
using Microsoft.AspNetCore.Http;
```

```
namespace coreweb.Pages
{
    public class IndexModel : PageModel
    {
        public string Msg { get; set; }

        int cnt = 0;

        public void OnPost()
        {
            Msg = "#click = " + (++cnt);
        }

    }
}
```

7.4.2.1.4. Wrong Implementation 2

we could now come to the idea to replace the counter with a static class variable. This seems to work in the first place but the problem is that the counter is shared among all sessions which means it is shared among all browsers.

```
using Microsoft.AspNetCore.Http;

namespace coreweb.Pages
{
    public class IndexModel : PageModel
    {
        public string Msg { get; set; }

        static int cnt = 0;

        public void OnPost()
        {
            Msg = "#click = " + (++cnt);
        }

    }
}
```

7.4.2.1.5. The Cookie Approach

A first working example uses cookies to store the counter value.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Http;

namespace coreweb.Pages
{
    public class IndexModel : PageModel
    {
        public string Msg { get; set; }

        int cnt = 0;

        public void OnPost()
        {
            string c = HttpContext.Request.Cookies["cnt"];
            if (c != null) cnt = Int32.Parse(c);
            Msg = "#click = " + (++cnt);
            HttpContext.Response.Cookies.Append("cnt", cnt.ToString());
        }

    }
}
```

7.4.2.1.6. Use Hidden Fields

Another working example uses hidden fields to store the counter value temporarily in the browser. In order to do this we have to modify the HTML file a little bit.

```
<input type="hidden" name="cnt" value="@Model.cnt" />
```

And enable the binding operation.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Http;

namespace coreweb.Pages
{
    public class IndexModel : PageModel
    {
        public string Msg { get; set; }
        [BindProperty] public int cnt { get; set; }

        public void OnPost()
        {
            Msg = "#click = " + (++cnt);
        }

    }
}
```

7.4.2.1.7. The Correct Session Implementation, Soft State

The former two applications are working in principle. One of the main problems is that because the information is stored in the browser, this can be easily modified by an attacker. Never ever trust an input which comes from a browser, because it can be easily modified. A typical solution is to implement what's known as a **soft state**. A soft state is a state information on the server side, where the server sores to state information for the client for a limited amount of time and the synchronization between the browser and their soft state is implemented using cookies. Here is an example:

```
using Microsoft.AspNetCore.Http;

namespace coreweb.Pages
{
    public class IndexModel : PageModel
    {
        public string Msg { get; set; }

        int? cnt = 0;

        public void OnPost()
        {
            cnt = HttpContext.Session.GetInt32("cnt");
            if (cnt == null) cnt = 0;
            Msg = "#click = " + (++cnt);
            HttpContext.Session.SetInt32("cnt", cnt.Value);
        }

    }
}
```

7.4.2.2. More on DotNet-Core

Let's have a brief look at the dotnet core web framework.

7.4.2.2.1. Hello World

A first example, containing only a simple "hello world" response. The page is delivered by the internal webserver "kerstrel".

7.4.2.2.2. Serve Static Files

Before we can load static files, we must enable the service with `app.UseStaticFiles();` and create the directory `wwwroot`.

7.4.2.2.3. Add JQuery and Bootstrap

These are two very common libraries. Use the following command to add them.

```
dotnet add package bootstrap --package-directory wwwroot
```

After that, add `bootstrap.css`, `jquery.js` and `bootstrap.bundle.js` to your project.

7.4.2.2.4. Add Razor Pages

In order to create dynamic pages, we must enable RazorPages.

In `CofigureServices`, add:

```
services.AddRazorPages();
```

And add the following endpoint mapping:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapRazorPages();
});
```

7.4.2.2.5. Add C# to CSHTML

Next we need to create a directory named `Pages`.

In this directory create a file with the extension ".cshtml". We use the name "Index.cshtml", because this is the default page. In this file, we can mixup HTML and C#. C# is recognized by the "@" character. The first statement is:

```
@page
```

7.4.2.2.6. Add a Model Class

The lines of code within a page should be kept very small. Thus we create a separate model class, which inherits from `PageModel`. Within the `cshtml` file we must add: `@model NameOfOurClass`. The public properties of this class can be accessed via: `@Model.prop`.

7.4.2.2.7. About OnGet and OnPost

Inside a C# code block, we now create a section marked as:

```
@functions{
}
```

Here we can place our `OnGet` and `OnPost` functions, which are invoked when we receive a `GET` or `POST` request. Use the debugger to verify.

7.4.2.2.8. Pass QueryString Arguments

Arguments from `OnGet` are taken directly from QueryString. We add an argument to `OnGet` and invoke the page using a different QueryString.

```
public void OnGet(string text) ...
```

This method can be called via `/?text=hello`.

7.4.2.2.9. Define a Form

To define a form, use the regular HTML syntax. Additionally, add the tag-helper `asp-for="myprop"` to an input. Then declare a property within the model class an add the attribute `[BindProperty]`.

7.4.2.2.10. Named Handler

Forms can add the `asp-page-handler` attribute. If this is done, we can add an extra parameter `handler` to the `OnGet` or `OnPost` methods or add the name of the handler to `OnGet` or `OnPost`. This is very useful for event handlers.

```
<form asp-page-handler="Edit" method="post">
        <button class="btn btn-info" name="Btn" value="1">Edit 1</button>
        <button class="btn btn-info" name="Btn" value="2">Edit 2</button>
</form>
```

Now add a extra method:

```
public ActionResult OnPostEdit(string Btn)
{ ... }
```

7.4.2.2.11. Routing Directives

By default a URL path maps to a filename of a cshtml file. We can change this with an argument to the page directive.

```
@page "/test"
```

The page can now be accessed using this path. The path can be absolute or relative. We can also pas arguments in the path like:

```
@page "{count}"
```

Which is then the retrieved by:

```
@RouteData.Values["count"]
```

7.4.2.2.12. Define a Common Layout

Because every page needs to have a common layout, create a file "Layout.cshtml". Copy all static HTML code to this file. In the Index.cshtml file, add a reference to the layout:

```
@{ Layout="Layout" }
```

The place in the layout-file, where we want to put our page is marked as:

```
@RenderBody()
```

7.4.2.2.13. Partial View

With partial view, we can create reusable HTML code. Simply place a HTML fragment in a cshtml file. In the main file, refer to it with:

```
@await Html.PartialAsync("Label")
```

We can pass an option argument as well, which is referred as `@Model`. If we enable tag-helpers:

```
@addTagHelper *,Microsoft.AspNetCore.Mvc.TagHelpers
```

We can also refer to the partial with:

```
partial name="Label" model="@i" />
```

7.4.2.2.14. MVC App

(Demo)

7.4.2.2.15. Web API / REST

A typical Web API is nowadays implemented using the REST architectural style. See https://en.wikipedia.org/wiki/Representational_state_transfer.

The reason why everybody moves to this REST architectural style is that it uses the basic HTTP methods to get and set the data and the content is represented as a JSON file which can be very easily passed using JavaScript. Thus we don't need an extra XML library on the client side. The only problem is, that a rest API does not support for exchange of meta information. The latter was resolved by using the OP API ()swagger) specification.

(Demo)

7.4.2.2.16. XML (SOAP) Webservice

Even though the XML or SOAP based web services is considered to be a little bit outdated let's have just a brief look on hog this works.

(Demo create the service)

```
    public class s1 : System.Web.Services.WebService
    {

        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }

        [WebMethod]
        public Person getPerson()
        {
            return new Person() { FirstName = "John", LastName = "Doe" };
        }
    }

    public class Person
    {
        public String LastName { get; set; }
        public String FirstName { get; set; }
    }
}
```

Consume it via powershell:

```
$w=New-WebServiceProxy -Uri http://localhost:33756/s1.asmx?WSDL
$w.getPerson()
```

7.4.3. Persistence Layer, Database Integration

The persistence layer is typically built using databases. A good starting point is to use the Entity-Framework.

7.4.4. Clientcode

The browser can natively execute javascript only, unless any helper plugin is installed, for example for java or flash. This browser code has no direct access to any server side code! Modern browser also support webassembly: see https://webassembly.org/. In principle, we can compile any code run on webassembly. Thus we are not limited anymore to javascript.

Here is a (not complete) list of client-side-frameworks:

- JQuery
- React
- Angular
- Vue
- Blazor (C#)

7.4.4.1. HTML-Dom

After being parsed by the browser, the Javascript code can change the complete HTML tree (or HTML DOM, DOM = Document Object Model).

Simple example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <style type="text/css">
        body { font-family: sans-serif;}
    </style>
    <script type="text/javascript">
        function myclick() {
            let e = document.getElementById("msg")
            e.innerHTML = "OK"
        }
    </script>
</head>
<body>
    <h1 id="msg" onclick="myclick()">Click me</h1>
</body>
</html>
```

7.4.4.2. JQuery

Unfortunately, browsers are very incompatible when it comes to the Javascript API. To overcome this obstacle, the project JQuery was created which tried to hide all the browser specialities. It is a very powerful framework for browser side code.

Here are a lot good examples:

- Getting Started: https://www.w3schools.com/jquery/jquery_get_started.asp
- Basic Syntax: https://www.w3schools.com/jquery/jquery_syntax.asp
- Selectors: https://www.w3schools.com/jquery/jquery_selectors.asp
- Events: https://www.w3schools.com/jquery/jquery_events.asp
- DOM Manipulation: https://www.w3schools.com/jquery/jquery_dom_get.asp
- GET and POST: https://www.w3schools.com/jquery/jquery_ajax_get_post.asp

7.4.4.3. Security

The typical security restrictions for Javascript are:

- The code cannot access any local information like cookies or browser history.
- It is only allowed to contact the same server from which the first resource was loaded from (Same Origin Policy).

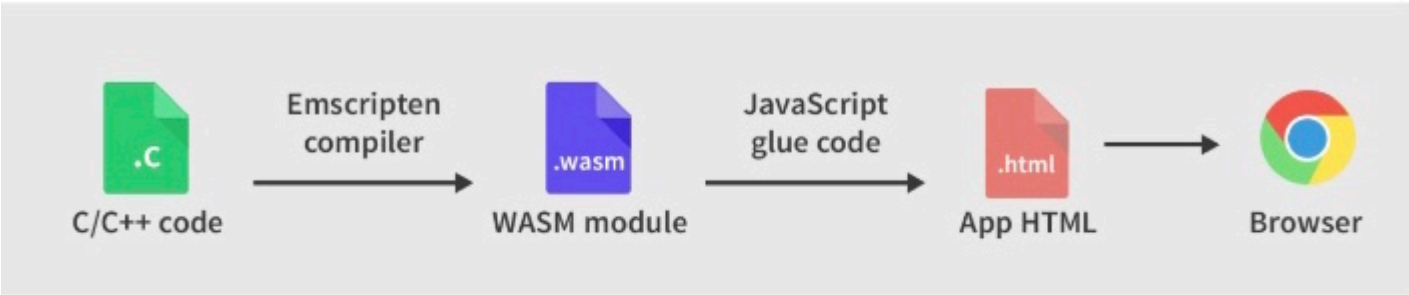
7.4.4.3.1. Same Origin Policy and CORS

To weaken things a little bit, we can use CORS:

See: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing.

7.4.4.4. Webassembly

One of the first attempts to run downloaded code inside the browser was implemented by means of java applets. In Java applets the browser downloaded the Java binary class file into a browser sandbox and ran it localy. This type of implementation is nowadays replaced by so called "web asembly" which is also an implementation of virtual machine, like Java, as is implements as a stack machine as well. The advantages is, that you can run any programming languages inside a browser as long as it can be compiled into webassembly binary.



Source: https://dev.to/real_sahilgarg/why-rust-is-good-for-web-assembly-and-path-to-learning-it-2njf

A good source of references is:

- Main web page: https://webassembly.org/
- C or C++ Compiler: https://emscripten.org/
- Webassembly binary toolkit: https://github.com/WebAssembly/wabt

7.4.4.4.1. C or C++

Let's assume, we want to call a C-function from within an html page. Here is the C-Code:

```
#include <stdio.h>
#include <emscripten/emscripten.h>

EMSCRIPTEN_KEEPALIVE int add(int a, int b) {
    return a+b;
}
```

Compile it ti WASM:

```
emcc func.c -o func.js -s WASM=1
```

We only need the wasm file. This can be loaded int html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <style>
        body { font-family: sans-serif; }
    </style>
    <script type="text/javascript">
        async function myclick()
        {
            const response = await fetch('func.wasm');
            const buffer = await response.arrayBuffer();
            const module = await WebAssembly.compile(buffer);
            const instance = new WebAssembly.Instance(module);
```

```
            const result = instance.exports.add(1,2);
            let r = document.getElementById("msg")
            r.innerHTML = "result = "+result
        }
    </script>
</head>
<body>
    <h1 id="msg" onclick="myclick()">click me</h1>
</body>
</html>
```

7.4.4.4.2. Blazor Page

An example, where we use Webassembly is Blazor-Pages. We can use C# in the browser.

(Demo).

7.4.5. Push vs. Pull

The HTTP protocol is pull based by default. If we want to push information from the server to the client, there are these option:

- Keep the http connection open for one resource and stream the changes.
- Create separate socket connection.
- Use web sockets
- Use special libraries like SignalR

7.4.6. Backend Services

Most modern web applications are interacting with services on the backend. Let's make a few examples.

7.4.6.1. Simple Text Service

This is a simple HTML page and webservice. The webservice returns the server time, which is added to to the HTML page.

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="default.aspx.cs" Inherits="srv._default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Service Demo</title>
    <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
    <script type="text/javascript">
        $(function () {
            $("#click").click(function () {
                $.get("srv.aspx", function (data) {
                    $("#time").html(data);
                });
            });
        });
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <div id="click">
            Click for server time: <span id="time">...</span>
        </div>
    </form>
</body>
</html>
```

```
using System;

namespace srv
```

```
{
    public partial class srv : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Response.ContentType = "text/plain";
            Response.Write(DateTime.Now.ToString());
            Response.End();
        }
    }
}
```

7.4.6.2. XML Webservices

The basic idea of a webservice is: Send (POST) the request as xml structure and read the response as xml as well. The xml structure is called SOAP (Simple Object Access Protocol).

The metainformation for enpoint mapping is defined in WSDL (WebService Description Language).

(create and consume a web-service)

7.4.6.3. Web API and REST

The REST (Representational State Transfer) approach uses the following ideas:

- Refer to the object, on which an operation should be applied, using regular URL syntax. Note: a URL can refer to any hierarchical object.
- Use the HTTP Verbs POST,GET,PUT,DELETE for the typical CRUD (Create Read Update Delete) operations.
- Encode the object in JSON or XML.

The metainformation can be defined by OpenAPI (formally known as swagger), which is an additional project: See: https://swagger.io/.

(Create a REST service using MVC Framework)

7.4.6.4. GraphQL

Problems with simpl REST APIs:

- Over-fetching
- Under-fetching

8. Cloud Computing

The term "cloud" computing is based on the fact, that the internet is visualized as a cloud in nearly every diagram. Thus "cloud computing" simply means that we can access it-resources over the internet. Beyond that simple definition, one fact is interesting. Cloud computing allows you to allocate resource "on demand". You do not have to spend money on resources when you don't need them, what is the case in on-premise installations.

8.1. Definitions

Some important definitions:

We can define a service model: (Everything as a service)

- IaaS: Infrastructure as a service.
- PaaS: Platform as a service.
- SaaS: Software as a service:

Depending on where the cloud resources are physically located, we can define:

- Public cloud
- Private cloud
- Hybrid cloud

See: https://en.wikipedia.org/wiki/Cloud_computing.

8.2. Provider

An example of major cloud providers are

1. Amazon
2. Azure
3. Google
4. VMWare
5. IBM
6. DigitalOcean
7. Telekom
8. Strato
9. Scanplus
10. ...

8.3. Storage

Remote storage was one of the first "cloud resources" which became available. It was used even before the cloud itself was invented. In its simplest form, it is a remote WebDAV connection. (See the beginning of our script).

8.3.1. Files

This is simply a remote file server using one of the widely adopted remote protocols: WebDAV and CIFS.

8.3.2. File sync

The major providers offer a file synchronization service as well. Examples are:

- Onedrive (microsoft)
- Google Drive (google)
- Dropbox (amazon for storage)
- ICloud (Apple)
- BW Sync and Share (Germany, Baden-Württemberg)

It's highly recommended to encrypt sensitive data on the client side, not on the server! The is necessary if we use personal data. This is necessary, if we use personal data in accordance with the Data Protection Act.

Example of a client side software for encryption: https://www.boxcryptor.com.

Remote storage comes in different flavours as well.

8.3.3. Blobs

A BLOB (Binary Large OBject) is typically the main unit of allocation for cloud providers. It's a binary storage of any kind. It can be used for files or disk images. Usually, redundancy assignments (locally and globally distributed) are defined in units of blobs.

(demo)

8.3.4. Tables

More structured storage are tables. This can be roughly compared with a database with one table and a different number of columns per row. Each row has a primary key for access and a key for partitioning (distributing the table among different physical locations).

(demo)

8.3.5. Queues

Queues can be used to decouple multi-tier application, thus creating loosely coupled distributed applications. It's intermediate storage for application communication.

(demo)

8.3.6. SQL Databases

You can allocate traditional SQL Databases in the cloud. Usually, every database has a remote access protocol. Thus it is well suited to be run in the cloud.

(demo)

8.3.7. NoSQL Databases

NoSQL databases are designed for nonstructured data or with replication in mind. Thus they provide a more loosely consistency model, compared to a SQL-Database. In short, this means, that a programmer cannot expect that every replica in updated at the same time, he might have to wait. (See book Tanenbaum for an explanation of consistency models).

8.4. Compute

The next most important remote resource is a "compute resource".

8.4.1. VM

A virtual machine (VM) is virtual hardware running on top of the cloud providers hypervisor. They can be remotely administered. It is up to the user to fully manage the instance.

8.4.2. Container

A computer resource can be an instance of a container as well. See class "operating systems" for definitions of containers. The most famous examples are docker container:

See: https://www.docker.com.

8.5. Network

Since all compute-resources must somehow be connected to the outside world, each provider allows you to define the network interconnectivity. This is known as "Software Defined Network".

8.6. Webapp

Since nearly every cloud application is kind of a web application, this is by far the most important class of a platform as a service. You can create a local web application and deploy it to a running instance of a web server. You do not have to manage the web server by yourself, as it would be the case if you use a VM only.

8.7. Serverless Functions

To make things even more easy, "serverless functions" allows you to define your business function only. You do not have to create a full-fledged web application. These functions are triggered by an external event like a queue message, a timer or a REST API call.

8.8. Exercises

Let us finally create a distributed webbased application and deploy it.

Depending on your previous experience, create one of the following examples application. The examples are with increases difficulty.

8.8.1. Create a local web app

Create a local application:

1. Using asp.net, create a calculator app, which allows you to add, subtract, multiply or divide numbers.
2. Deploy your app to our private cloud. Use the following connection to get more info about the publishing settings: `http://<yourloginname>.web.vsys.st.hs-ulm.de/`. Of course replace "`<yourloginname>`" with your real loginname. Now you can load the web-apps from other students as well.
3. Deploy your app to the public azure cloud. You need to import the publishing profile, which can be found in the same directory where you uploaded your public key. The public URL will be something like: `https://hsu-dwsys-yourloginname.azurewebsites.net/`.
4. Create an other aspx file, which does the calculation but returns plain ASCII. Use Querystring to pass your arguments. Call your service from PowerShell or curl.
5. Create a plain HTML file, which consumes your service. The HTML file has to be loaded from the same server for security reasons.

If you have enough time, you can implement your calculator service as regular webservice or REST webservice.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="default.aspx.cs"
Inherits="calculator._default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Calculator</title>
    <style type="text/css">
        body { font-family:sans-serif }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
            <br />
            <br />
            <asp:Button ID="ButtonPlus" runat="server" Text="+"
            OnClick="ButtonPlus_Click" />
```

```
            <asp:Button ID="ButtonMinus" runat="server" Text="-"
            OnClick="ButtonMinus_Click" />
            <asp:Button ID="ButtonMult" runat="server" Text="*"
            OnClick="ButtonMult_Click" />
            <asp:Button ID="ButtonDiv" runat="server" Text="/"
            OnClick="ButtonDiv_Click" />
            <br />
            <asp:Label ID="Ergebnis" runat="server"
                Text="---" Font-Size="24pt"></asp:Label>
        </div>
    </form>
</body>
</html>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace calculator
{
    public partial class _default : System.Web.UI.Page
    {
        int a;
        int b;
        string err;

        private void getArgs()
        {
            try
            {
                a = Int32.Parse(TextBox1.Text);
                b = Int32.Parse(TextBox2.Text);
            } catch (Exception ex)
            {
                err = ex.Message;
            }
        }

        protected void Page_Load(object sender, EventArgs e)
        {
            this.PreRender += _default_PreRender;
        }

        private void _default_PreRender(object sender, EventArgs e)
        {
            if (!String.IsNullOrEmpty(err)) Ergebnis.Text = err;
        }

        protected void ButtonPlus_Click(object sender, EventArgs e)
        {
            getArgs();
            Ergebnis.Text = (a + b).ToString();
        }

        protected void ButtonMinus_Click(object sender, EventArgs e)
        {
            getArgs();
            Ergebnis.Text = (a - b).ToString();
        }

        protected void ButtonMult_Click(object sender, EventArgs e)
        {
            getArgs();
            Ergebnis.Text = (a * b).ToString();
        }

        protected void ButtonDiv_Click(object sender, EventArgs e)
        {
            getArgs();
            Ergebnis.Text = (a / b).ToString();
```

```
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace calculator
{
    public partial class srv : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string ergebnis;
            Response.ContentType = "text/plain";
            try
            {
                int a = Int32.Parse(Request.QueryString["a"]);
                int b = Int32.Parse(Request.QueryString["b"]);
                int r;
                switch (Request.QueryString["op"])
                {
                    case "add": r = a + b; break;
                    case "sub": r = a - b; break;
                    case "mul": r = a * b; break;
                    case "div": r = a + b; break;
                    default: throw new Exception("illegal op");
                }
                ergebnis = r.ToString();
            } catch (Exception ex)
            {
                ergebnis = ex.Message;
            }
            Response.Write(ergebnis);
            Response.End();
        }
    }
}
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Service Test</title>
    <style type="text/css">
        body {
            font-family: sans-serif
        }
    </style>
    <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
    <script type="text/javascript">
        function calc(op) {
            var a = $("#a").val();
            var b = $("#b").val();
            var url = "srv.aspx?a=" + a + "&b=" + b + "&op=" + op;
            $.get(url, function (data) { $("#res").html(data) });
        }
    </script>
</head>
<body>
    <input id="a" type="text" />
    <input id="b" type="text" />
    <br />
    <br />
    <input id="add" type="button" value="+" onclick="calc('add')" />
    <input id="sub" type="button" value="-" onclick="calc('sub')" />
    <input id="mul" type="button" value="*" onclick="calc('mul')" />
    <input id="div" type="button" value="/" onclick="calc('div')" />
    <br />
    Ergebnis: <span id="res">----</span>
```

```
</body>
</html>
```