TDP013 – Web Programming and Interactivity

Lecture 3: JS in the browser, AJAX, CORS, Ethics assignment

Huanyu Li

Human-Centered Systems, Department of Computer and Information Science

LINKÖPING UNIVERSITY

Recap from lectures 1 and 2

- JavaScript
- Callback function
- Node.js
- Server framework written in JavaScript
- Support for almost everything in ES6 (if you work with defining your code as a module)
- MongoDB
- HTML, CSS, JavaScript Cookies

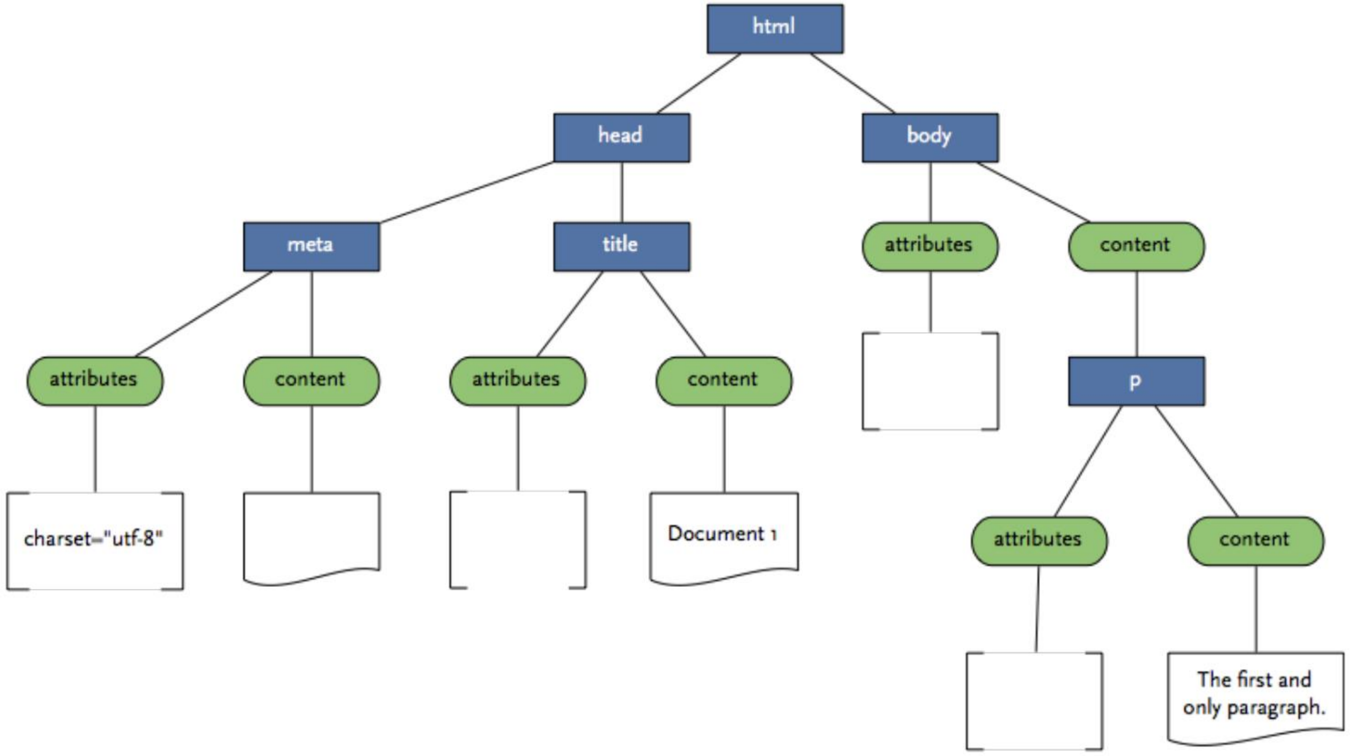
JavaScript in the browser

LINKÖPING UNIVERSITY

JavaScript in the browser

- Browsers have a JavaScript engine that executes JavaScript code
  - e.g. V8 in Chrome, SpiderMonkey in Firefox
- Web APIs are typically used with JavaScript
  - A list of Web APIs: https://developer.mozilla.org/en-US/docs/Web/API
  - HTML DOM API
    - access to and control of HTML elements via DOM
    - https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API

DOM


html
head
body
meta
title
attributes
content
attributes
content
attributes
content
attributes
content
p
attributes
content
charset="utf-8"
Document 1
The first and only paragraph.

Node in DOM

- Each element in an HTML document is a node in the DOM tree (including
- <!-- comments -->)
- There are 12 different types of nodes
- Element, TextNode and AttributeNode are the three types that are generally interesting for web design

Navigating the DOM

- To make changes to the DOM tree with JavaScript, you need to be able to get specific elements, e.g.:
- document.getElementById('param') returns the element with the specified ID
- document.getElementsByTagName('param') returns a list of elements with a specific tag
- document.querySelector(<css selector>) returns the first element based on a CSS selector
- document.querySelectorAll(<css selector>) retrieves a list of elements based on a CSS selector.

Operations on nodes

- element.childNodes returns a list of all nodes directly below element in the DOM tree.
- element.parentNode returns the node directly above element in the DOM tree.
- element.nextSibling returns the node directly to the right and at the same level as the element in the DOM tree.
- element.previousSibling returns the node directly to the left and at the same level as the element in the DOM tree.

Operations on nodes

- document.createElement(’param’) creates a new element based on a tag expressed as a string
- document.createTextNode(’param’) creates a new TextNode from a string.
- element.appendChild(child) places the specified element child last in the list of nodes directly below element
- element.removeChild(child) removes an element from the list of nodes directly below the specified element. The node must be in the list of the element’s children.

Callback and asynchronous calls

LINKÖPING UNIVERSITY

Event-loop

- Node.js only uses on one thread and all requests are executed in this thread
- If Node.js waits for each line of the code to execute before continuing, it means that everyone who made the calls to the server need to wait

```
// a function that needs longer running time
let data = ProcessNeedsLongerRunning()
```

If we have such a function call above, the response can be very slow

- Node.js uses Promises to handle asynchronous operations

Asynchronous calls

- Run a function without pausing
- can utilize callbacks or Promises
- asynchronous functions are marked with async, and return promises

```
async function doSomething(){
        // e.g., time consuming processing
        return "Hello World"
}
```

- To wait on an async function use await
  - wait for a resolved promise, inside an async function
  - can be used to make asynchronous calls behave serially

```
async function main() {
        let a = await doSomething();
        console.log(a);
}
```

Asynchronous calls - Promise

- Object representing a “promise”
- acts as a placeholder for a result to be available at some point
- 3 states
  - pending: initial state
  - fulfilled: the operation succeeded
  - rejected: the operation failed
- created using “new Promise()” constructor
  - the constructor takes an argument, i.e., an executor function with 2 arguments
    - resolve: a function to call if the operation succeeds
    - reject: a function to call if the operation fails

Asynchronous calls - Promise

- .then(...)
  - this block handles successful resolutions
- .catch(...)
  - this block handles rejections happened in the promise or any of the .then blocks
- multiple .then(...) can be defined for the same Promise

Asynchronous calls - Promise

```
function loadData(){
        return [
                {'title': 'Gone in 60 seconds', 'year': 2000},
                {'title': 'Pulp Fiction', 'year': 1994}
        ]
}
let p = new Promise((resolve, reject) => {
        let data = loadData()
        if(data !== null){
                resolve(data)
        } else {
                reject('Failed to load data')
        }
})
p.then((x) => {
        // 'then' is called if we succeed
        console.log('Data loaded successfully:')
        console.log(JSON.stringify(x, null, 2))
}).catch((msg) => {
        // 'catch' is called if we fail
        console.log(`Something went wrong: ${msg}`)
})
```

What are callbacks?

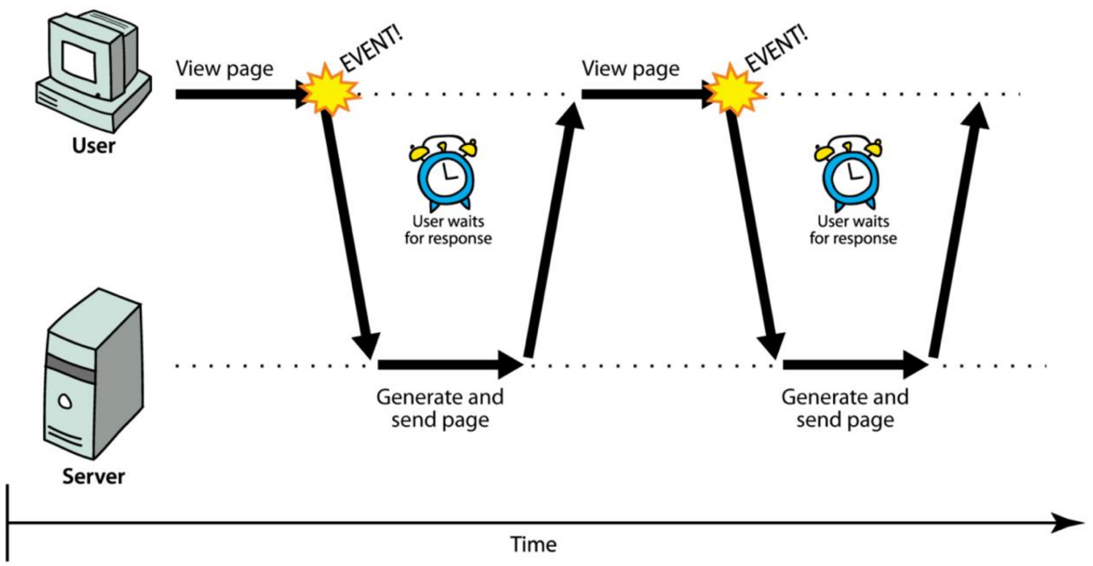
- functions as arguments to functions
- hands over the responsibility for capturing data and events to the called function
- in JavaScript and third-party libraries
- “If I give you my passport, could you pick up the package I ordered, leave it outside my door and then call me?”

HTTP calls
Retrieving and Sending data on the web

LINKÖPING UNIVERSITY

Synchronous calls on the web

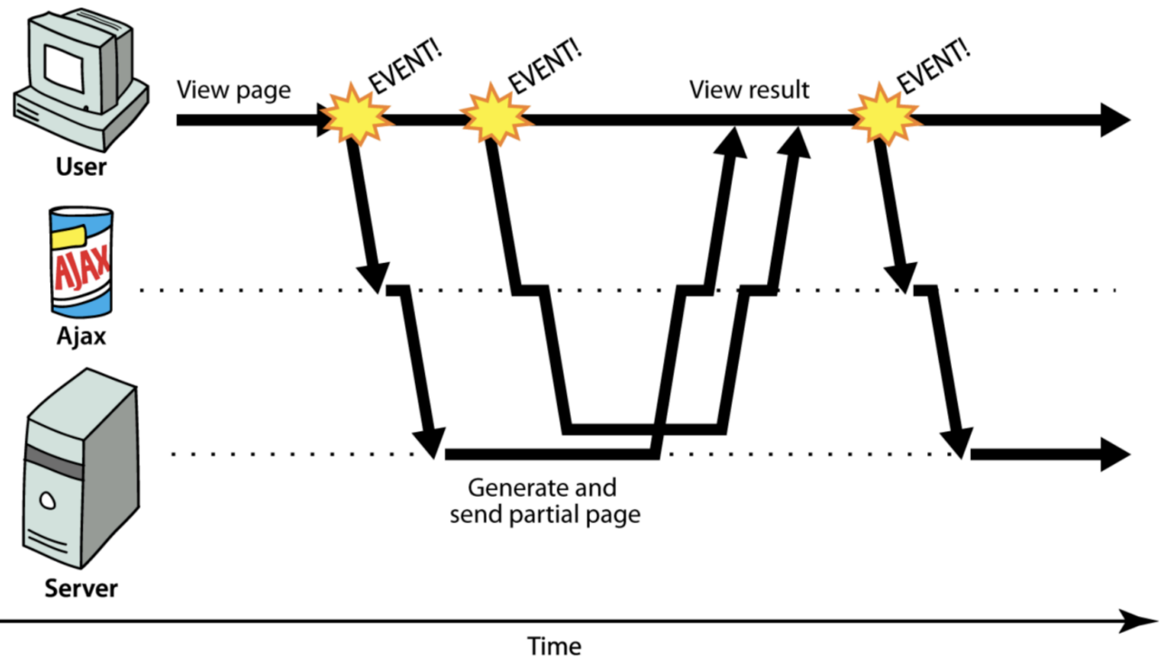
- The user must wait for a response and cannot do anything in the meantime.
- The entire page is refreshed.



LINKÖPING UNIVERSITY

Asynchronous calls on the web

- The user can do other things while waiting for a response from the server.
- Only the affected parts of the page are changed.



https://www.rose-hulman.edu/Class/csse/csse290-WebProgramming/201330/Slides/lecture21-ajax.shtml

AJAX

- Asynchronous JavaScript and XML
- Enables asynchronous calls on the web via JavaScript
- Is done a little differently depending on which browser is used, but the differences are today very small
- What comes back from the server is (usually) JSON, XML, etc.

AJAX – Send request

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = () => {
        if (xhttp.readyState == 4 && xhttp.status == 200) {
                let data = JSON.parse(xhttp.responseText);
                console.log(data);
        }
};
xhttp.open('GET', 'https://gorest.co.in/public/v1/users', true);
xhttp.send();
```

0 UNSENT

1 OPENED

2 HEADERS_RECEIVED

3 LOADING

4 DONE

"true" makes the call asynchronous

HTTP methods

HTTP – methods

- Request-Response model between client and server
- Most common methods
  - GET – Asks the server to return a specific resource
  - HEAD – Asks the server to send information about a specified resource (without sending the content itself)
  - POST – Sends information to the server that changes information on the server OR sends information that is inappropriate to include as part of the URL
  - PUT – Adds or updates a resource
  - DELETE – Deletes the specified resource
  - OPTIONS – Asks the server to return a list of HTTP commands that the server supports

HTTP – methods

- Request-Response model between client and server
- Most common methods
  - GET – Asks the server to return a specific resource
  - HEAD – Asks the server to send information about a specified resource (without sending the content itself)
  - POST – Sends information to the server that changes information on the server OR sends information that is inappropriate to include as part of the URL
  - PUT – Adds or updates a resource
  - DELETE – Deletes the specified resource
  - OPTIONS – Asks the server to return a list of HTTP commands that the server supports

AJAX – Send data with GET

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = () => {
        if (this.readyState == 4 && this.status == 200) {
                let data = JSON.parse(this.responseText);
                console.log(data);
        }
};
xhttp.open('GET', 'https://gorest.co.in/public/v1/users', true);
xhttp.send();
```

0 UNSENT

1 OPENED

2 HEADERS_RECEIVED

3 LOADING

4 DONE

"true" makes the call asynchronous

AJAX – Send data with POST

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
                let data = JSON.parse(this.responseText);
                console.log(data);
        }
};
xhttp.open('POST', 'https://gorest.co.in/public/v1/users', true);
xhttp.setRequestHeader('Content-type', 'application/json');,      xhttp.setRequestHeader('Authorization', 'Bearer
<access token>');
xhttp.send('{"name": "John Doe", "gender": "male",
                        "email": "john.doe@noone.com", "status": "active"}');
```

Code Example

fetch(...)

AJAX with promises

LINKÖPING UNIVERSITY

Why promises or callback-hell

```
fetchResource(
    url,
    function (result) {
        // Do something with the result
        fetchResource(
            newUrl,
            function (result) {
                // Do something with the new result
                fetchResource(
                anotherUrl,
                function (result) {
                        // Do something with the new result
                },
                failureCallback
            );
            },
            failureCallback
        );
    },
    failureCallback
);
```

```
fetchResource(url)
.then(handleResult, failureCallback)
.then(handleNewResult, failureCallback)
.then(handleAnotherResult, failureCallback);
```

JavaScript Fetch API

- The API allows web browser to make HTTP requests to web server
  - no need to use XMLHttpRequest
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

AJAX – Send request

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = () => {
        if (xhttp.readyState == 4 && xhttp.status == 200) {
                let data = JSON.parse(xhttp.responseText);
                console.log(data);
        }
};
xhttp.open('GET', 'https://gorest.co.in/public/v1/users', true);
xhttp.send();
```

0 UNSENT

1 OPENED

2 HEADERS_RECEIVED

3 LOADING

4 DONE

"true" makes the call asynchronous

fetch: AJAX with promises

```
const url = 'https://gorest.co.in/public/v1/users'
fetch(url, { 'method': 'GET' })
        .then(resp => resp.json()) // parse and move on to the next 'then'
        .then(data => {
                console.log(data);
});
```

```
const url = 'https://gorest.co.in/public/v1/users'
let resp = await fetch(url, { 'method': 'GET' })
if (resp.ok) { // HTTP status 200-299
        let data = await resp.json();
        console.log(data);
}
```

CORS

Cross-Origin Resource Sharing

LINKÖPING UNIVERSITY

Same-origin policy

- A security model for web browsers
- Restrictions about how a document /scripts from one origin can access data from another origin

- Same origin
  - same protocol, host and port, “scheme/host/port”

- Link an image from an external website
- Fetch data from your API

CORS

- Restrictions due to security reasons
  - “Cross-site scripting”
  - Risk of injections
  - Can bypass authentication
- AJAX requires that all calls are made to exactly the same domain that the client is running on!
  - If your page is on the domain http://example.com, you can only call services on http://example.com/...
- CORS is used to explicitly grant permissions to the server for certain domains

CORS

- Browsers typically use the "same-origin policy"
- Before the GET/POST call, an OPTIONS call is sent to the server
- If the correct headers are returned, the browser allows you to perform GET/POST
- A relatively "neat" way of doing it that minimizes too much code changes in existing systems
    - configure CORS on the server side

CORS: Response headers

- On the server side, you add what and which domains should be allowed based on what is written as a response in headers
- Must be added to all outgoing "responses" that you want to make available
- NOTE: Here we choose to set '*' which allows all domains to call the server. In a production environment, you usually specify domains that should be allowed to send calls.

```
let headers = {};
headers['Access-Control-Allow-Origin'] = '*';
headers['Access-Control-Allow-Methods'] = 'POST, GET, OPTIONS';
res.writeHead(200, headers);
res.end();
```

CORS: Response headers

- How can you speed up and simplify the process with headers?

```
if(req.method == 'OPTIONS'){
        let headers = {};
        headers['Access-Control-Allow-Origin'] = '*';
        headers['Access-Control-Allow-Methods'] = 'POST, GET, OPTIONS';
        res.writeHead(200, headers);
        res.end();
} else {
// vid POST, GET, etc.
}
```

CORS: Response headers

- In Express.js we can do this easily with .use(…)
- .use(…) is called every time the app receives a request, regardless of which route is used
- There are libs that make working with CORS even easier
  - https://www.npmjs.com/package/cors
  - https://www.npmjs.com/package/helmet
  - …but it's a good idea to check that you don't open things up too much!

```
app.use((req, res, next) => {
        res.header('Access-Control-Allow-Origin', '*');
        res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');
        next();
});
```

CORS: cors middleware

- Simple usage by enabling all cross-origin requests

```
import express from 'express';
import cors from 'cors';

let server = express();

server.use(cors())

server.get('/cors', (req, res) => {
res.status(200).send('cors');
});
```

LINKÖPING UNIVERSITY

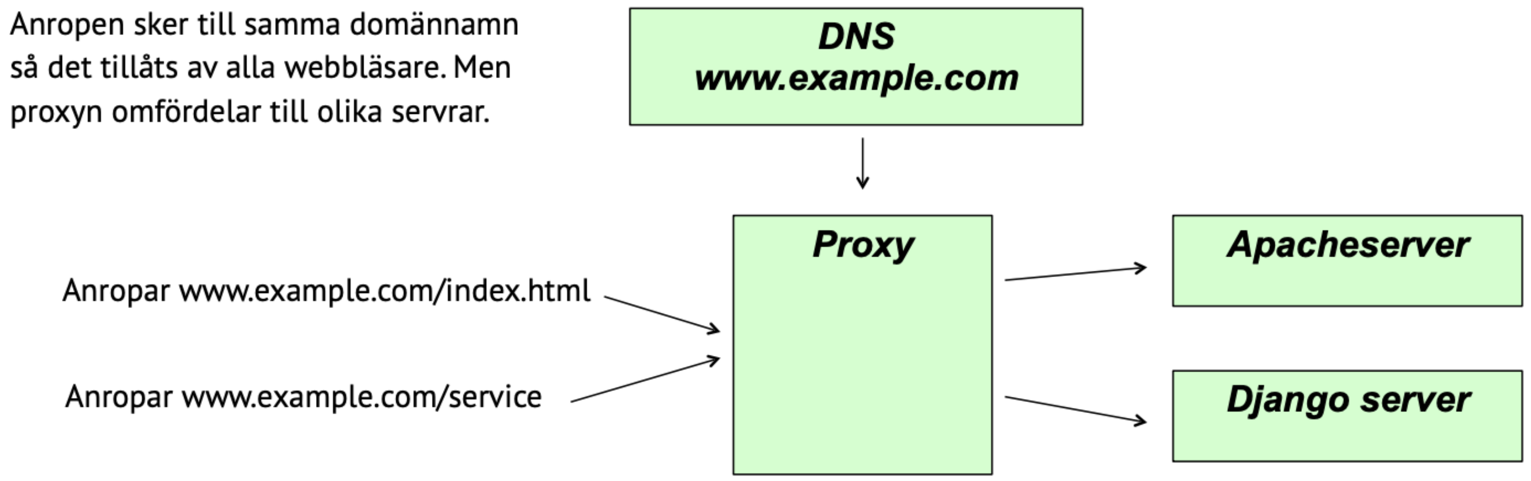
CORS: cors middleware

- With cors configuration

```
const corsOptions = {
  origin: 'http://example.com',
  optionsSuccessStatus: 200,
  methods: ['GET', 'POST', 'PUT', 'PATCH']
};

server.use(cors(corsOptions))

server.get('/cors', cors(corsOptions), (req, res) => {
  res.status(200).send('cors');
});

// cors not enabled for this route
server.get('/no-cors', (req, res) => {
  res.status(200).send('no cors');
});
```

Can you do without CORS?

- Yes, but it gets more complicated
- You can use a “proxy” that handles all calls to the domain
- A proxy can also have other benefits such as “caching” while also working with all browsers
- Out-of-scope in this course!


Anropen sker till samma domännamn så det tillåts av alla webbläsare. Men proxyn omfördelar till olika servrar.
DNS www.example.com
Proxy
Apacheserver
Django server
Anropar www.example.com/index.html
Anropar www.example.com/service

How do we check that CORS works?

- Call your server from an external domain
    - Easy if your server is online
- Option 1:
    - Create a file that makes a call to localhost
    - Then open the file directly in the browser!
    - NOTE: Does not always work for all browsers
- Option 2:
    - Make a call with OPTIONS and check the content in the headers

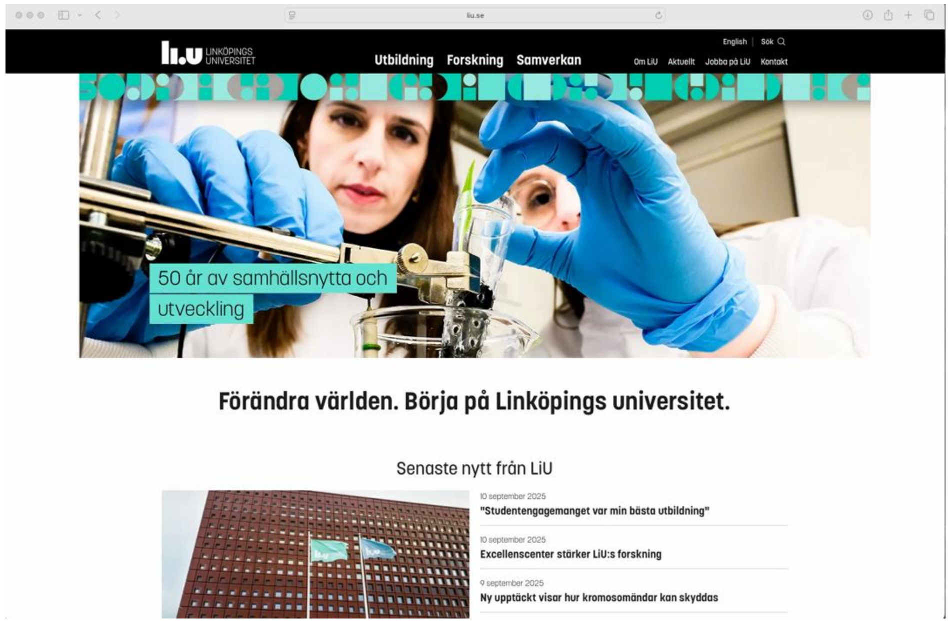
Code Example

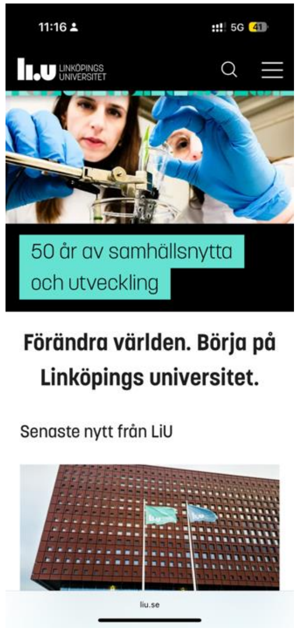
Responsive web design

LINKÖPING
UNIVERSITY

Responsive web design

- Make web pages render well and look good on all devices
  - desktops, laptops, mobiles, tablets, watches, etc.


liu.se
LINKÖPINGS UNIVERSITET
English | Sök
Utbildning Forskning Samverkan
Om LiU Aktuellt Jobba på LiU Kontakt
50 år av samhällsnytta och utveckling
Förändra världen. Börja på Linköpings universitet.
Senaste nytt från LiU
10 september 2025
"Studentengagemanget var min bästa utbildning"
10 september 2025
Excellenscenter stärker LiU:s forskning
9 september 2025
Ny upptäckt visar hur kromosomändar kan skyddas


11:16
5G
LINKÖPINGS UNIVERSITET
50 år av samhällsnytta och utveckling
Förändra världen. Börja på Linköpings universitet.
Senaste nytt från LiU
liu.se

How to design responsive web sites

- Media queries, introduced in CSS3
  - it allows us to apply CSS styles based on some conditions such as screen size, device orientation, resolution.

```
@media CONDITION {
/* … */
}
```

```
@media (max-width: 600px) {
 body {
  font-size: 14px;
  background: lightblue;
 }
}
```

More conditions: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries/Using_media_queries

Toolkit

- Bootstrap
- Foundation
- Zimit Framework
- Pure.css

Bootstrap

- CSS framework for mobile-first front-end web development
- Grid system and components
- Button example:
    - https://jsfiddle.net/lihuanyuasas/scemh0ny/

Ethics Assignment

LINKÖPING UNIVERSITY

Task Overview

- Reflect on corporate ethics policies
- Learning objective
  - "Kunna redogöra och analysera etiska aspekter relaterade till ämnesområdet"
- What you need to do:
  - choose a company with a formal ethical code or policy (e.g., "code of conduct")
    - software or hardware development with a global connection.
  - Send your choice to Huanyu, and register on webreg by September 26, 23:59
    - Assign group for seminars on October 1st
    - *https://www.ida.liu.se/webreg3/TDP013-2025-1/UPG1*
  - Apply Gibbs's Reflective Cycle (adapted version) to analyze and reflect
  - Write a reflection about 1 page (ca. 500 words)

Ethics assignment

- Include or link to the code of ethics or policy
- Upload the documents to git
- The seminar assignment is done individually

- Deadline for report: Wednesday, October 1st
- Seminar: Wednesday, October 1st 1:15 PM–5:00 PM
  - Charlie Simonson: 1:15pm to 2pm, 2:15pm to 3:00pm
  - Anders Fröberg: 1:15pm to 2pm, 2:15pm to 3:00pm
  - Huanyu Li: 3:15pm to 4pm, 4:15pm to 5:00pm

LINKÖPING
UNIVERSITY