CSE1500: Assignment WDT 04

General Information

As in the data segment of this course, you are required to work in teams of two (2) students each. It is **mandatory** to continue working with your partner from the data segment of the course, and it is **mandatory** to work with the same partner for all of the web assignments (WDT 04-06). We hope that this continuity will help make your collaboration and teamwork effective, helping you to achieve the learning objectives of the course.

There are three assignments remaining in CSE1500. As described in the main syllabus document,[1] each assignment will be graded with either a **`pass`** or **`fail`**. As a reminder, to pass the assignment component of the course, you need to have passed at least **50%** of the assignments. In case you have passed less than **50%** but more than **25%**, you will be offered a repair option.

Each assignment has a **deadline** as shown on Brightspace. Please make sure to submit it before the deadline. Submissions after the deadline will not be considered, and your grade for the assignment will be a **`fail`**. We will provide reference solutions shortly after the deadline has passed.

🧠 *Please remember that the assignments teach you several important practical skills not directly covered by the lectures. Assignments are a pivotal part of your learning activities which are designed to align well with your learning objectives. Thus, we strongly recommend completing all assignments, even if you are over the passing threshold.*

Introduction

This assignment requires you to use your front-end development skills in order to develop a web-based system or website using HTML, CSS, and JS for a **personal movie recommender platform**. Building off of your schema and database system in the DB segment of the course, you and your partner will now leverage and hone your web development skills to design the interface of your platform and bring it to life! Exciting, right? Do read along.

Learning Goals

This assignment will play an important role in helping you achieve the following learning objectives (LOs) of the CSE1500 course:

LO1 **Explain the basic architecture of the Internet and the web**

LO2 **Analyse the requirements for a web application given a description of an application idea**

LO3 **Create an application front-end with HTML and CSS based on a given static design**

LO4 **Create an interactive web application with client-side JavaScript**

[1]The syllabus document can be accessed from the course page on Brightspace. Read this thoroughly if you have not done so yet!

Technologies

You will be using the following technologies for the first assignment. We have linked useful resources for each topic at the end of this document for your own further reading. Please familiarize yourself with the OMDb API especially, as this will be an essential tool in your application.

- Telnet
- HTML5
- Cascading Style Sheets (CSS)
- JavaScript (Vanilla JS)
- OMDb API

Deliverables

Across all three assignments, you will develop a complete and interactive website. This assignment focuses on familiarizing yourself with basic architectural concepts of the web, and with frontend design. Thus, for this assignment, we will expect the following:

Task	Deliverable
1.1	Answers to 5 questions: text + images
1.2	Answers to 5 questions: text + images
2.1	3 wireframes as images on page
2.2/3	Website code in separate `.zip` file

Table 1: Deliverables Overview—Assignment `WDT 04`

All text and image/screenshot deliverables should be merged into a **single** PDF, and any code should be zipped into a **single** `.zip` archive before uploading to Brightspace. Your submission will consist of only those two files. If you lose your way within the assignment, use this deliverables overview (cf. Table 1) to get back on track.

1 HTTP

You should have attended Lecture 1 to do this task.

To get familiar with how the Web works on an architectural level, in the first part of this assignment, you will interact with a webserver at the lowest level possible: the Hyper Text Transfer Protocol. You will use a shell interface[2] and `telnet` to carry out these exercises. If you are using a Linux distribution or macOS, these tools might already be installed or can be installed through your package manager. If you are using Windows, consider installing the Windows Subsystem for Linux.

⭐ Here are a few practical tips to help you out:

- To store `telnet`'s output to file (in addition to printing it on the console), you can use the command `tee`, e.g., `telnet www.tudelft.nl 80 | tee out` will save all output to a file called `out`.
- To exit a telnet session, first press `CTRL` + `]`. This brings you to a `telnet>` prompt and you can enter `quit` to end the session.
- Be aware that `some-domain.com` is not the same as `www.some-domain.com`.

[2]You might find this MIT class about the shell useful.

- Be aware of the **backspace key** when working in the terminal: while on a normal command line a backspace deletes the last character typed, within the `telnet` environment this key may be forwarded to the server instead.
- A lot of webservers nowadays time out the connection quite fast. If that affects you, consider writing your request in a notepad and copying it into `telnet`. You can paste in most terminals with `CTRL` + `SHIFT` + V.

1.1 `telnet`

Study the OMDb API and obtain an API key. You can start your conversation with `www.omdbapi.com` by typing in `telnet www.omdbapi.com 80`. Your task is to request the details of the movie "Batman & Robin", by querying it *by title*.

a) Show the HTTP request(s) you made to get the information.

b) What did you do in order to request the movie by title? Did typing in `Batman & Robin` directly in the appropriate URL parameter work? Why do you think that is?

c) What encoding is used for the response data? Did the server tell you, or were you supposed to guess?

d) What does the `Cache-Control` header from the response tell you?

e) There are several references to **Cloudflare** in the response. What purpose does Cloudflare serve, how do they do that, and what HTTP features allow for some of these purposes? (Hint: refer to your answer for the previous question)

1.2 `telnet`

Your requests so far have been `GET` requests. To have you try out some other methods, we will make use of httpbin.org, a popular site designed to test HTTP messages. Below is an example of a `PUT` request. Observe how there is a newline between the headers section and the body. Try this request for yourself, then respond to the questions below.

```
cse1500@tudelft:~$ telnet httpbin.org 80

PUT /put HTTP/1.1
Host: httpbin.org
Content-Type: text/plain
Content-Length: 12

Hello World!
```

a) What does this endpoint do? Does it actually modify any data server-side (like a `PUT` is usually used for), or not? Can you think of a scenario where a tool like httpbin.org would be useful?

b) What happens if you try to replace `/put` in this exercise with another resource (e.g. `/myfile`)? Does the httpbin.org server allow the creation of a new resource?

c) The `Content-Length` is exactly the number of characters (12 - we count the whitespace as well!) of `Hello World!`. What happens if the `Content-Length` field is smaller or larger than the exact number of characters in the content?

d) Try out some other HTTP methods and tools that httpbin.org offers. What does `/deflate` do and why is something like that useful in the real world?

e) How about `/drip`? When do you know it's finished?

2 Website Prototype

As mentioned you will build a personal movie recommender. On this platform, users can choose to create lists of movies or interact with lists by following or commenting on them. Here is an exact list of the basic functionalities.[3] In general, you have considerable (artistic) freedom.

1. Users are able to sign-up or log-in to their profile on the platform. They should have a username, password, email, and gender.
2. Users are able to create lists that have a name and description and contain movies/series.
3. Users are able to add other users as "friends" and remove them. Lists of friends are displayed separately.
4. Users are able to query for movies/series on the website, upon which the option to add to a list is given.
5. Users are able to (un)follow lists and can access their followed lists.
6. Users are able to vote on a list (thumbs up or down), comment, and reply to comments.
7. Users are able to become "premium users" through a paid subscription offer[4]
8. Premium users are able to "favorite" any number of actors, which they can access on the application.

2.1 Wireframes

You should have attended Lecture 2 to do this task.

Before you start any coding for your web app, let us get down to basics. ***Wireframes*** are a skeletal framework for your digital project, outlining its structure and layout. They are useful because they make your ideas visual, and they ensure everyone is on the same page before the real work begins. Think of wireframes as a practical tool – they save time, help you focus on user experience, serve to easily communicate visual requirements before any programming happens, and move the guesswork and trial-and-error of trying out designs outside the programming phase.

In practice, again before getting to actually developing your website, it is also common to map out your expected user flows.[5] For this task, these user flows are **optional**, though recommended. For the actual wireframes, we suggest you use Figma and for the user flows we suggest Miro.

ℹ️ *Keep in mind these wireframes and designs are **living documents**. This means that they should not be set in stone. If requirements change over time, if a specific design element is deemed infeasible to implement in practice, if some aspect was overlooked during the initial design, or if any other deviation from the design is deemed necessary, make sure to update your designs and wireframes.*

a) Produce at least **three (3)** wireframes corresponding to three different webpages that you have identified from the list of expected functionalities.

b) ***Optional task***: Map out 5 main user flows for your platform.

[3] Do not hesitate to add a feature if you believe it benefits the website. This is optional and will not affect your grade.
[4] A dummy/mocked payment feature is enough (e.g., a button to become a paid user is sufficient).
[5] You can read more about user flows here!

2.2 Static Website Prototype (HTML + CSS)

You should have attended Lecture 2 to do this task.

Using your design from above as a starting point, in this task, you need to create a basic (hard-coded) version of your personal movie recommender platform. As a result of parsing through the list of expected functionalities, we expect you to identify the logic and flows between the three web pages that you choose to develop in order to realize the expected functionalities.

Note that for the first assignment we **DO NOT expect any working database/backend functionality**. This means hard-coded data and interactions are sufficient for pages/flows involving the database.[6] For example, when viewing your created lists, it is okay if hard-coded lists are displayed.

When creating your HTML pages, please take into account that interactions can extend upon your base interface. Likewise, when styling your website with CSS, keep user experience (UX) elements like color theory and responsive design in mind. Furthermore, your website should meet the following requirements:

1. You must develop the same three (3) designed pages from the wireframes task.
2. You must account for the basic functionality in the chosen pages (no need to make it interactive/working yet).
3. You must use modern HTML and CSS. Do not use HTML styling!
4. Your HTML and CSS should be in separate files such that the same CSS file can be used for all three pages.
5. You must use semantic/content sectioning tags where appropriate instead of generic tags like `<div>`.
6. You must use CSS flexbox at least **once** in an appropriate context.

2.3 JavaScript + Requests

You should have attended Lecture 1, 2, and 3 to do this task.[7]

⚠️ *Although in this course we will eventually touch on framework(s), for the sake of this assignment* ***you must*** *program in plain (vanilla) JavaScript. In principle, we* ***do not allow external libraries or frameworks*** *beyond those specified in assignments and lectures.*

For this task we expect a specific interactive web element: **a movie/series search system**. This will be achieved by interacting with the OMDb API. In order to use the OMDb API, you are going to need a free API key which you must use for all your requests. You should already have a key from Task 1.

You are **required** to implement a search page/component, which should consist of a search box with search button, and an area for displaying results based on **title**. If one of the pages you designed and developed above was a search page, implement this functionality there. If not, implement this functionality in a new page.

1. The JavaScript code must be in a separate file, not inside your HTML.
2. This cannot be a singular/hard-coded `GET` request[8], which you use to display one specific movie/series.
3. The value in the search box must be the key used for the search request.

[6] In the next assignment (`WDT05`) you will work with the database you designed in the data segment.
[7] Yes, the lecture on REST APIs is not needed.
[8] If you are struggling with this, please browse the change log of the OMDb API in more detail.

4. The search box and button must be elements of a `<form>` element.
5. The page must not refresh when pressing the search button.
6. The HTTP request(s) should be made using the `fetch()` API.
7. You have a choice between using `Promises` or `async/await`, provided what you're using works on the latest versions of Chrome or Firefox.
8. We expect that when one, for example, queries for "Batman", multiple results are visible.
9. The results area must show both the title of a searched movie, and also its poster.

Submission

1. Add all text and images to a **single** PDF document, making sure every exercise is clearly labeled and delimited.

2. Add all code to a **single** `.zip` file. Your archive should contain at least 3 `.html` files, 1 `.css` file and 1 `.js` file.

3. Upload these two and only these two files to Brightspace.

Resources

- MDN Docs (one of the most comprehensive documentations on Web technologies)
- Flexbox Froggy (a useful resource to practice CSS and responsive design)
- Google's web.dev (an amazing resource for anything web development related)
- W3Schools (a great resource for HTML/CSS)