Project Safe Web - Specification

Project Files: Figma project / REST API
Project Mentor: Marjan Ralevski

Description

This UI project is about a web application that focuses on educating how to have a safe web experience. Please initially review the Figma project in detail in order to familiarize yourself with the expected design and pages. Moreover, consider running the Figma project by using the play button in the top-right corner.

About the used images in the project, please download them from the Figma project itself (you need to log in to Figma first and then export the resources) or use any other images that look fine in the overall UI. For the pages where video is displayed (i.e., the home page video, with a title below it "Информирај се и заштити се"), you can initially link the page to any larger resolution picture and skip the video playback. For the advanced requirements, you can integrate a proper video playback via HTML and eventually JavaScript (more about this further down the text).

All the text content which is displayed in the application can be mocked by using any random text generator, preferably a Lorem Ipsum one (i.e., this one). All the data that seems to be fetched from some backend, such as the comments posted in the "Лоши навики при 'live streaming'" страната should be mocked (i.e., hard-coded in the code itself). The icons displayed in the application can be downloaded from this Figma page (take a look at the "Design system" section of the page).

You are not limited to using any particular technology for implementing the UI application, but as mentioned in the *Project Deliverables* section, you would need to provide written documentation in a file (within the GitLab repository itself perhaps) which would describe how to start the project locally.

For logging in (authenticating) the following username - password pairs (credentials) can be used:

- User123 - Pass123
- User456 - Pass456
- User789 - Pass789

Moreover, in the logging in page the Google/Apple/TikTok authentication action should not be linked to any specific action (no functionality should be implemented for them). The "Запомни ме" checkbox should not introduce any additional application behavior.

Furthermore, the “Добредојде!” overlay should be displayed after the user is successfully authenticated (when using one of the credential pairs from above).
In order to enable the login functionality, you need to set up a simple REST API locally written in Python with the Flask library. For this, you need to have Python installed locally (go to the “Files” section in the page and you will most probably need the “Windows installer (64-bit)” if you are running locally on Windows). The version which was used for developing the application 3.12.3). After installing Python, you should download the REST API locally and then run it by using the following CLI command in the directory where the extracted REST API directory is located:

python '.\REST API\authenticator.py'

When you execute this command, if you get the following error:

```
Traceback (most recent call last):
  File "C:\Users\ralev\Downloads\REST API\REST API\authenticator.py", line 1, in <module>
    from flask import Flask, request, jsonify
ModuleNotFoundError: No module named 'flask'
```

or an error like “No module named ‘flask_cors’”, then you should install the flask library and/or the flask_cors by running the following CLI commands:

pip install flask
pip install flask_cors

and then retry the **python** command from above.

Afterwards, you can use the REST API for authenticating, as explained in requirement #6 (please be aware that in actual frontend-backend communication the credentials are most often heavily encrypted). For confirming that the authenticator REST API is up-and-running, you should see the following output in the terminal:

```
 * Serving Flask app 'authenticator'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 126-270-323
```

A tutorial about how to send HTTP POST requests with JavaScript - tutorial link. In case you want to test the HTTP POST endpoint from the REST API without JavaScript, then you can use the cURL CLI tool - tutorial link.

The application should not initially support internationalization, meaning that the content should remain originally in Macedonian, as displayed in the Figma design.

And lastly, all the content in the Figma design that is not explicitly mentioned in this specification can be hardcoded or added without any additional functionality (e.g., the magnifying glass icon in the top-right corner).

Requirements

1. Implement the UI application identically to how it is displayed on Figma. For this, use the images provided in the Figma project or use any other images that look fine in the overall application. Videos can be replaced with images with an additional added on hover play button (this requirement is extended in requirement #8).

2. Single page routing is implemented. This means that we can change the page (route) by not triggering a page refresh. Hint: Consider using the hash route for this and trigger page change with the *onhashchange* JS event (when the hash route is changed, page A becomes hidden, page B becomes visible). Reading this article is a good starting point.

3. The application is implemented in a mobile-friendly manner, which means that it can be used on any type of screen. The mobile-friendly design is included in Figma.

4. In the "Сè што треба да знаеш" page different videos should be shown according to which filters are active (the pills above the videos, i.e., "Најгледани"). For this, the video data objects should contain some category property in order to support the various filters (at least 3 filters should be present within this page, meaning there is no need to have all the filters as displayed on Figma). The selected filters are saved in session and are available for the currently logged in user even after refreshing/changing the page or logging out and then logging in again. Initially, no filter is active and all cards are displayed.

5. Adding new experiences (“искуства”) in the "Дискусии" / "Табла за дискусии" page: New experiances/posts should be stored in session for the currently logged in user and should be visible after refresh. The shared experience should contain the username of the author and the date and time when the comment was posted. Experiences from different logged in users should be visible on this page. The number of comments and reactions can be hardcoded.

6. Logging in to the application: For this you would need to send an HTTP POST request to "http://localhost:5000/api/authentication" after starting the REST API locally. The request data should look like this *{"username": "User123", "password": "Pass123"}* (you can use any of the other valid credentials, as mentioned above for the overall application testing). If the response is 200 OK (the username-password pair is part of the list of supported credentials mentioned above), then you should store into session a value that the user has successfully logged in and adequately adjust the appearance and behavior of the application based on this (i.e., only authenticated users can access the profile page, only they can post discussions, as well store the selected filters in the home page). Consider storing the logged in username in session based on the needs for other requirements.

7. Changing the email address and birth year for the currently logged in user: These changes should be stored in session for the currently logged in user and as such they

should be available after refresh, but also after the user logged out and logged in again. Hint: the changes should be linked with the currently logged in user.

8. Videos are used throughout the application: Instead of using images, as explained in requirement #1, for this requirement you would need to replace these images with actual videos. The content of the video is not important, what counts is that the video can be played within the application. Hint: Consider reading this page as a starting point.

Level 1: Beginner
Requirements #1 to #3 are implemented (3 requirements in total).

Level 2: Intermediate
Requirements #1 to #5 are implemented (5 requirements in total).

Level 3: Advanced
Requirements #1 to #8 are implemented (all 8 requirements).

Project Goals & Objectives:

- The student will verify his/her understanding and knowledge regarding HTML, CSS and JavaScript.

Project Step-by-Step Procedural Information:
Step 1: Review thoroughly the Figma application design.
Step 2: Plan on how to implement the UI application initially without JavaScript.
Step 3: Create an empty project in GitLab and commit the changes there step by step as you implement the UI application.
Step 4: Implement the UI application identically to how it is displayed in Figma.
Step 5: Identify all the application segments where JavaScript is needed and implement them one by one.
Step 6: Confirm that the application JS functionalities work properly.
Step 7: Confirm that the application can be cloned locally and run from your created GitLab repository.

Project Deliverables

A short textual description is included for how to run the project locally (maybe in the *README.md* file of the project). The project itself, with all the files needed for running it locally, is available on GitLab. Additionally, you should add the GitLab repository link in the Brainster portal and also, enable access to the person who will be reviewing your project.

Evaluation system

Criteria	Excellent 25p	Proficient 20p	Good 15p	Fair 10p	Poor 5p
Completeness	*The project is implemented with all simple and advanced JS functionalities and without styling differences.*	*The project is implemented with all simple, but missing advanced JS functionalities, and without styling differences.*	*The project is implemented with missing JS functionalities and without styling differences.*	*The project is implemented with missing JS functionalities and with a few (2-3) styling differences.*	*The project is implemented with missing JS functionalities and with many (+5) styling differences.*
Accuracy	*No styling requirements are missing.*	*Maximum 1 styling requirement is missing.*	*2 to 5 styling requirements are missing.*	*Over 5, but below 10 styling requirements are missing.*	*Over 10 styling requirements are missing.*
Efficacy	*No functional requirements are missing.*	*Maximum 1 functional requirement is missing.*	*2 to 5 functional requirements are missing.*	*Over 5, but below 10 functional requirements are missing.*	*Over 10 functional requirements are missing.*
Documentation	*A complete documentation is provided which allows for the project to be set almost anywhere.*	*Documentation is provided, with all steps included, but some steps miss some peculiarities, like local configuration checks.*	*Documentation is provided, with few (2-3) missing steps, but without invalid steps.*	*Documentation is provided with few (2-3) missing/invalid steps for local setup.*	*Documentation is provided with many/all missing steps or with some invalid steps for local setup.*

Deadline

3 weeks after its presentation, at 23:59 (end of the day).

Assessment Rules

- Fair Assessment: ethical considerations
  - Assessors should ensure that the assessments are conducted in a fair and ethical manner, respecting the principles of academic integrity and honesty.
- Reliability and Validity: enabling consistency
  - Assessments should be consistent and reliable, meaning that they yield consistent results when applied repeatedly to the same task or performance.
  - Assessments must accurately gauge the knowledge, skills, or abilities they are intended to evaluate, ensuring their validity as indicators.
- Feedback: as a method for continuous improvement
  - Assessors should offer constructive feedback that identifies strengths and areas for improvement. The Feedback should be specific and actionable, it should include thought provoking guides and should challenge the student to become better at a specific task.
- Late Submission Policy: assessing assignments after their deadline
  - Students should be allowed a grace period of 3 days (72 hours) to make a late submission on any assignment, with the notice that they will be deducted 20% from the total possible points.
- Plagiarism Policy: assessing assignments with matching solutions
  - In the event of suspected plagiarism, the assessor is required to promptly collaborate with the Student Experience Coordinator/Team as the initial step. Together, they will draft a notice to remind students of the strict prohibition against plagiarism, with potential repercussions for recurrent violations. The following actions will be considered in cases of repeated plagiarism:
    - If a submitted solution exhibits substantial similarity, exceeding 60%, with another student's work (individually or within a group), the respective challenge/project will incur a 50% reduction from the maximum points attainable.
    - In cases where a solution is identified as more than 90% identical to another student's work (individually or within a group), the project/challenge in question will receive a score of 0 points.
  - The use of generative AI is encouraged as a learning tool in our educational programs; however, students must engage with the material and contribute with original thought. Reliance on AI for complete content generation is strictly prohibited and will result in point deductions, official warning, or other academic penalties.
  - Upon completion of the assessment process for each challenge/project, the assessor is tasked with selecting the most complete, optimal, and creative solution and to showcase it by publishing it on the platform together with the assessment results.
- Timeliness: timeframe for delivering results and feedback to students
  - Feedback on projects should be provided within 14 days after the deadline has passed (depending on the project's complexity).