Web Development and Web Design

Version Spring 2020

Lab project

Lab 1. Static pages layout

Max 10 points
In scope of this lab student practice creating page layout. For this purpose HTML will be used for defining structure and CSS with SASS preprocessor for styling. No JavaScript should be used on the current stage. Student should show a teacher how SCSS is compiled to CSS.

Testing and presentation Your web project (static HTML+CSS with SASS) on **GitHub Pages**.

***GitHub Pages** is a feature of **GitHub** which allows for a repository to be deployed to the web.*
*Find **GitHub Pages** under **Settings**, select the **master branch**, and click **save**.*
By default, the repository will be deployed to username.github.io/repository.
GitHub Pages is automatically updated when the repository is updated.

Student should create static pages for the future single page application. All planned sections should be covered. Layout should support response/adaptive design. In the next labs static pages will be converted to templates and will be used with chosen framework.
Required pages and functionality:

- Login page
- Logout functionality
- Users listing with two types of users: Admin and Regular
- “Create user dialog” with at least following fields: Username, Password, Confirm password, Role, buttons Ok, Cancel.
- Information about selected user
- “Edit user” screen
- Delete user
- Error notifications
- Validation errors
- Other screens required for chosen variant (at least 3)

HTML requirements

- Use DOCTYPE to define version
- Use correct structure (html, head and body tags should be present)
- Define technic information in correct way: meta tags and styles injection should be defined within head tag
- Do not use inline styles: all styles should be injected by link tag

- Follow standards of mark-up: inline elements should be within block elements
- When quoting attributes values, use double quotation marks
- Use elements according to their semantics, i.e. header, nav, sidebar, main, article, section, footer
- Check HTML for validity (you can use https://validator.w3.org/)
- Use alternate text for images, ex. <img src="js-logo.png" alt="JS logo">
- Use the names of the classes based on the context
- Use commentaries for your blocks of code to describe what is going on within it
- Use only lowercase: this applies to HTML element names, attributes, attribute values, CSS selectors, properties, and property values.

CSS requirements

- Do not use inline styles
- Do not use global selectors
- Always use hyphens in class names. Do not use underscores or CamelCase notation.
- Always define generic font families like sans-serif or serif.
- If you use 0 as a value, do not add a unit (px, em, etc.) after it.
- Avoid very complex child and descendant selectors
- Use pseudo-classes, i.e. link, visited, hover, active
- Use pseudo-elements, i.e. first-child, before and so on
- Use transitions, animations
- CSS responsive

CSS preprocessors

- Do not use pure CSS
- Use SASS with SCSS syntax
- Styles should not be duplicated
- Use variables for all colors
- Create separate style file with all of color variables
- Use correct structure based on guide of preferable preprocessor
- Use mixins where you can (at least in one place)
- Limit nesting to 1 level deep. Reassess any nesting more than 2 levels deep. This prevents overly-specific CSS selectors.
- Avoid large numbers of nested rules. Break them up when readability starts to be affected. Preference to avoid nesting that spreads over more than 20 lines.
- Always place @extend statements on the first lines of a declaration block.
- Where possible, group @include statements at the top of a declaration block, after any @extend statements.

Git repository should be organised in the following way:

- Repository shall contain main branch *development*
- There should be at least 1 commit for each lab
- Commits shall be named reasonably: "User listing static page" is good, "Listing" - wrong.
- Commits for each lab shall be pushed to a separate branch with a reasonable name
- After a lab will be done, code shall be merged to *development* via *merge request*
- Branch with lab shall NOT be deleted
- In the end there should be 5 branches: 4 with each lab and 1 *development*.

Git and GitHub

- Git can keep track of changes made to code, synchronize code between different people, test changes to code without losing the original, and revert back to old versions of code.
- GitHub is a website that stores Git repositories on the internet to facilitate the collaboration that Git allows for. A repository is simply a place to keep track of code and all the changes to code.
- Git commands:
    - git clone <url> : take a repository stored on a server (like GitHub) and downloads it
    - git add <filename(s)> : add files to the staging area to be included in the next commit
    - git commit -m "message" : take a snapshot of the repository and save it with a message about the changes
    - git commit -am <filename(s)> "message" : add files and commit changes all in one
    - git status : print what is currently going on with the repository
    - git push : push any local changes (commits) to a remote server
    - git pull : pull any remote changes from a remote server to a local computer
    - git log : print a history of all the commits that have been made
    - git reflog : print a list of all the different references to commits
    - git reset --hard <commit> : reset the repository to a given commit
    - git reset --hard origin/master : reset the repository to its original state (e.g. the version cloned from GitHub)
- When combining different versions of code, e.g. using git pull, a merge conflict can occur if the different versions have different data in the same location. Git will try to take care of merging automatically, but if two users edit, for example, the same line, a merge conflict will have to be manually resolved.
    - To resolve a merge conflict, simply locally remove all lines and code that are not wanted and push the results.

Список пропонованих варіантів Проектів.
Інші теми необхідно погодити з викладачем.

- Варіант 1. Створити сервіс переказу коштів між користувачами, кожен користувач має власний гаманець та можливість переказувати чи отримувати кошти від іншого користувача.
- Варіант 2. Створити сервіс коротких (404 символи) заміток (із тегами) для кожного користувача із можливістю перегляду, редагування і видалення, а також надавати доступ редагувати замітку іншими користувачами (до 5 користувачів). Також надати можливість бачити статистику користувача, скільки повідомлень, коли редаговані і ким.
- Варіант 3. Створити сервіс оголошень + CRUD із двома рівнями повідомлень. Оголошення повинні бути локальними та публічними. Локальні оголошення тільки для користувачів, що знаходяться в тому ж місці. Публічні для всіх, навіть для не користувачів сервісу.

- Варіант 4. Створити сервіс кредитування користувачів на основі даних, що користувач вводить при реєстрації, кошти для кредитування видаються із бюджету 517 000 грн ставка 30%. Також реалізувати можливість погашення кредиту.
- Варіант 5. Написати сервіс статей (2000 символів). Статті є публічними для всіх, зареєстровані користувачі можуть редагувати статтю та очікувати на схвалення її модераторами (користувачі із більшими правами). Передбачити варіант редагування, коли стаття на розгляді модератором, а інший користувач її теж редагує. Модератори мають бачити статті, які очікують їх схвалення.
- Варіант 6. Написати сервіс простий інтернет магазин. Користувачі можуть купувати один із 8 товарів, які є в обмеженій кількості на складі, не допустити можливості продажу одного і того ж товару кільком користувачам.
- Варіант 7. Створити сервіс для резервування аудиторій на певну дату час та проміжок часу від 1 години до 5 днів. Користувачі мають можливість резервувати аудиторії, а також редагувати, скасовувати та видаляти їх. Застерегти користувачів від накладок (два користувачі не можуть зарезервувати аудиторію на певний період час)
- Варіант 8. Написати сервіс для купівлі та бронювання квитків на концерти, події і т.д. Користувачі мають можливість купувати квиток, бронювати квиток,
- скасовувати бронь. Унеможливити купівлю чи бронювання одного і того ж квитка кількова користувачами.
- Варіант 9. Написати сервіс для створення плейлистів. Користувач може мати як приватні (видимі тільки для нього), так і публічні (видимі для всіх) плейлисти. Публічні плейлисти можуть редагувати всі користувачі.
- Варіант 10. Створити сервіс для календаря подій. Користувач має можливість створювати подію, редагувати її, видаляти, долучати інших користувачів до події, переглядати перелік всіх створених події, та подій до яких він долучений.
- Варіант 11. Створити сервіс для проведення онлайн занять. Повинні бути користувачі двох рівнів – викладачі та учні. Викладачі можуть створювати, видаляти, редагувати курс, переглядати перелік створених курсів, долучати студентів до курсу. До курсу може бути долучено не більше 5 студентів. Студенти можуть переглядати всі дані лише тих курсів, до яких вони долучені. Також студенти можуть надсилати на участь у якомусь курсі, а викладач має можливість прийняти або відхилити запит.
- Варіант 12. Створити сервіс для збереження та редагування рейтингу студентів. Для зберігання даних про студента використовувати json. Також реалізувати можливість отримання списку кращих за рейтингом студентів.
- Варіант 13. Написати сервіс для роботи з сімейним бюджетом на спільному рахунку. В сім'ї повинно бути не менше 3 людей. Кожен користувач має можливість переглядати бюджет, додавати в нього кошти, знімати кошти на свій персональний рахунок. Також передбачити можливість збереження та виведення переліку витрат та доходів як сім'ї загалом, так і кожного її учасника.
- Варіант 14. Створити сервісів для прокату авто. Користувачі сервісу можуть бути двох рівнів – адміністратори та пасажири. Адміністратори можуть додавати та видаляти авто із системи, редагувати інформацію про авто. Пасажири можуть переглядати каталог та бронювати авто на певний час.
- Варіант 15. Написати сервіс для роботи аптеки. Провізор може додавати препарати в базу, видаляти, редагувати інформацію про них. Користувач може переглядати

інформацію про препарати, здійснювати купівлю, якщо немає препарату в наявності, то його можна додати в попит.

- Варіант 16. Написати сервіс для роботи кінотеатру. Адміністратор може складати розклад показу фільмів, з урахуванням тривалості фільму та зайнятості залу, редагувати розклад, видаляти та додавати фільми.

In the next parts:
Lab 2. Client-server interaction with pure JS, no jQuery. AJAX. Linter. **Max 10 points**
Lab 3. Basic single page application with user management. Framework. **Max 10 points**
Lab 4. Add project specific functionality to SPA. **Max 15 points**
Lab 5. Unit tests. **Max 15 points**

Max 60 points for all 5 labs
Перевірка на плагіат