



Навчальний проект





Навіщо потрібні навчальні проекти?

- Розібратись з технологіями
- Отримати практичний досвід їх використання
- Отримати досвід роботи в команді
- Навчитись дотримуватись дедлайнів 😊
- Зробити щось цікаве та корисне
- Основна мета – навчитись, а не зробити “ідеальний” проект
 - Не боятись використовувати нові технології
 - Краще недоробити щось цікаве, ніж в черговий раз здати той самий (чи трохи змінений) проект

Проект

- Реалізувати веб-застосунок
- Головний результат – працюючий застосунок
 - Проте додаткові матеріали (вимоги, архітектура, презентація, ...) будуть корисними
- Можна робити індивідуально
- Можна робити в команді (орієнтовно до 4)
 - Відповідно більш серйозний проект

	Можна	Бажано	Обов'язково
Вибрати тему, технології, визначитись з вимогами	хоч зараз	до 25 вересня	до 9 жовтня
Перша демонстрація, розгорнутий проект, репозиторій з кодом	до наступної лекції	до 23 жовтня	до 6 листопада
Майже готовий проект	на наступній лекції	до 20 листопада	до 4 грудня

Дати (проект)

вересень 2017						
нд.	пн.	вт.	ср.	чт.	пт.	сб.
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

жовтень 2017						
нд.	пн.	вт.	ср.	чт.	пт.	сб.
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

листопад 2017						
нд.	пн.	вт.	ср.	чт.	пт.	сб.
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

грудень 2017						
нд.	пн.	вт.	ср.	чт.	пт.	сб.
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						



Бажано



Обов'язково

- Проекти можна демонструвати під час лекцій
- Або віддалено (email, TeamViewer, ...)

Що використовувати

- Немає обмежень на використанні технології
 - Хоча якщо використовуються екзотичні технології чи їх комбінації – треба пояснити свій вибір
- Має використовуватись мережева взаємодія
 - Сервер
 - Клієнт (веб, чи мобільний клієнт, чи ...)
 - Взаємодія за протоколом HTTP (наприклад, RESTful service, можна WebSockets)



Чому вільний вибір технологій?

- У кожного має бути можливість вивчити щось нове
 - Різний попередній досвід
 - Різні вподобання щодо технологій
 - Фіксовані технології = хтось нічому не навчиться, а інші змушені вивчати щось для них нецікаве
- Можливість чомусь навчитись у колег
 - Кожен учасник проекту отримує глибокий досвід використання технологій
 - Цим досвідом варто ділитись
 - Можливість порівняння різних технологій
 - Вільний вибір технологій = більше різних технологій
- Вибір технологій – важлива частина реальних проектів
 - Цьому теж варто вчитись
 - У навчальних проектах невдалий вибір не матиме таких серйозних наслідків, як у проектах з реальними замовниками та дедлайнами



Використання сучасних технологій



- Важливо не просто продемонструвати свої знання, а й вивчити щось нове
- Технології повинні сприяти реалізації проекту
 - Чому саме ця технологія підходить для саме цього проекту
 - “Просто вивчити технологію” – нормальне пояснення; проте тоді варто обирати відповідну тему проекту
- Більша команда має більші можливості
 - Або взяти більше технологій
 - Або глибше їх вивчити і використати
- В кінці проекту – оцінити, наскільки технологія була корисною/зручною (ретроспектива)

Як можна вибирати тему

- Вибирайте тему, яка буде Вам цікава
- Зацікавила якась задача?
 - Придумайте для неї технології
- Зацікавила якась технологія?
 - Придумайте для неї задачу



Командна розробка

- Проект варто робити в команді
 - Хоча можна й індивідуально
- Оптимальний розмір команди – 3-4 учасника
 - Хоча можна трохи більше чи менше
- Індивідуальна робота має бути видимою
 - Repository commits: кожен за себе, а не один за всіх
 - Хто автор документу? (revision history table)
 - Засоби командної роботи: issue tracker, Trello, Slack, ...



Ролі в команді

- Визначитись з ролями в команді
 - Team lead
 - Backend/frontend
 - Відповідальні за окремі компоненти
 - Відповідальні за окремі технології
 - UI/UX, deployment/infrastructure, testing, security, ...
- Ролі можуть бути гнучкими
 - В одній ролі – декілька учасників
 - Один учасник в декількох ролях
 - Ролі змінюються з часом / залежно від етапу проекту



Етапи проекту

1. Обрати тему, визначитись з технологіями, визначити основні вимоги
2. Викласти код в репозиторій, розгорнути проект на зовні доступному сервері, перша демонстрація
3. Майже готовий проект

1 етап

- Мета: визначитись, що робити
- Вибрати тему
 - Варто погодити її з викладачем до вибору технологій та опису вимог
- Вибрати технології реалізації
 - Сервер та клієнт (backend + frontend)
 - Приблизно визначитись з використанням мов програмування, бібліотек/фреймворків
 - Приблизно визначитись, хто що буде робити
 - Надалі можливі зміни – але з поясненням, чому попередньо обрані технології не підійшли
- Короткий опис вимог
 - Основні функціональні та нефункціональні вимоги
 - В довільній формі (текстовий документ, таблиця, записи в issue tracker, ...)
 - Не варто витрачати багато часу на занадто детальні вимоги (1-2 сторінки = нормально, 0,5 сторінки = теж нормально, 20 сторінок = мабуть, забагато)
 - Надалі можливі зміни – можна як додавати нові вимоги, так і прибирати раніше додані; треба буде пояснити причини змін



2 етап

- Мета: налаштувати інфраструктуру
- Початкова версія коду
 - Можливо, без реалізації жодних вимог
 - Але варто використовувати ті самі технології
 - Також можуть бути прототипи окремих функцій
 - Чи перші версії реалізації
- Викласти в репозиторій
 - Репозиторій має бути постійно доступним викладачу
 - Public or private (в останньому випадку додати викладача з правами на читання)
- Розгорнути на загальнодоступному сервері
 - localhost не підійде 😊
 - Можна використати безкоштовні хмарні провайдери
 - Обов'язково надіслати посилання викладачу
- Продемонструвати першу версію
 - Можна під час лекцій (на перервах, після лекцій)
 - Можна просто надіслати посилання на e-mail



3 етап



- Мета: продемонструвати майже готовий проект
- Основні функції реалізовано і готові для використання
- Можливі невеликі проблеми – їх можна буде виправляти далі
 - До кінця семестру/консультації/іспиту
 - Але чим пізніше надіслати – тим більше шансів, що не вистачить часу перевірити...
- Можна демонструвати на лекціях чи надсилати посилання на e-mail

Що здавати

- Основний результат проекту – отримані знання та досвід
 - Варто провести ретроспективу наприкінці проекту, щоб їх зафіксувати
 - Варто також фіксувати нові знання та досвід протягом проекту
- Видимий результат проекту
 - Розгорнутий та працюючий застосунок
 - Код в репозиторії
- Додаткові документи – за бажання та якщо корисні
 - Більш детальний опис вимог
 - Архітектура
 - Презентація проекту
 - Інформація про реальне використання проекту, користувачів, ...
 - ...
- Додаткові документи дозволять отримати більше балів
 - Але навіть без них можна отримати повні 20 балів лише за якісно реалізований застосунок

Дедлайни

- По два дедлайни для кожного етапу
- Бажані дедлайни
 - Варто ставити саме їх за мету
 - Бонусні бали, якщо їх досягнути
 - Сприятиме отриманню автоматів у спірних ситуаціях
 - Пропуск бажаних дедлайнів не призводить до зниження балів
- Обов'язкові дедлайни
 - На 2 тижні пізніше за бажані
 - Остаточна мета, якщо не вдалось вкластись у бажаний дедлайн
 - За пропуск обов'язкових дедлайнів знімаються бали (-5 балів за кожен пропущений тиждень)
 - Ці бали можна буде відновити з використанням бонусів
 - але це не привід їх втрачати 😊



Оцінювання



- 20 балів за проект в цілому
- Бали знімаються за недоліки в проекті
 - ~ 15 балів – незначні проблеми, що (майже) не заважають використанню
 - ~ 10 балів – більш суттєві проблеми, можуть затруднювати використання чи робити окремі функції недоступними
 - ~ 5 балів – критичні проблеми, більшість функцій недоступні, проблеми з безпекою
 - ~ 0 балів – проект не реалізовано взагалі

Рекомендації, Best Practices

- На лекціях будуть обговорюватись рекомендації щодо реалізації різних аспектів проекту
- Цих рекомендацій варто дотримуватись
 - Якщо порушено багато рекомендацій, і немає аргументів чому – можуть зніматись бали
- Далі на наступних слайдах – приклади рекомендацій
 - Це лише орієнтовна версія – більш остаточні рекомендації будуть в наступних лекціях

Server-side (Backend)

Implementation Checklist – MUST

- ☐ Clear separation of UI, business logic and data access code
- ☐ Don't store access credentials (passwords, keys, tokens) in source code
- ☐ Don't use GET for requests that change server state
- ☐ Don't pass sensitive data (e.g. session IDs) in URLs
- ☐ Validate all user inputs on server
- ☐ Prevent common attacks: SQL Injection, XSS, CSRF
- ☐ Never store passwords in plaintext – store only bcrypt (or hash+salt – not recommended)
 - ☐ Don't use obsolete crypto algorithms such as MD5, SHA1
- ☐ Don't support obsolete protocols such as SSL3
- ☐ Use **HttpOnly** option on cookies

Implementation Checklist – SHOULD

- ☐ Use web framework instead of writing everything from scratch
- ☐ Single source of truth for data model
 - ☐ ORM
 - ☐ Code generation
- ☐ Store DB structure in source control
 - ☐ SQL scripts
 - ☐ Migrations
- ☐ Don't serve static content (CSS, JS, images) from the same server as dynamic content
- ☐ Avoid copy-paste in templates
 - ☐ Template inheritance
 - ☐ Reusable blocks
 - ☐ Parametric templates
- ☐ Don't hardcode user-visible text – use some i18n framework
- ☐ Implement consistent URL structure
- ☐ Prefer routing to URL rewriting

Implementation Checklist – SHOULD

- ☐ Implement Post-Redirect-Get
- ☐ Implement nonce for post data
- ☐ Validate user inputs on client when possible
- ☐ Check for common cases of wrong inputs: too large, too small, outside range, forbidden symbols
- ☐ Cache static content for long time (e.g. 1 year)
 - ☐ If it changes – change URL
- ☐ Cache dynamic content for short time (e.g. 1 min – 1 hour) depending on update frequency

Implementation Checklist – SHOULD

- ☐ Allow sufficiently long and complex passwords
- ☐ Use different salt value for each password (bcrypt does this automatically and is the preferred method)
- ☐ Don't reveal too much information with password recovery option
 - ☐ Don't replace password until user has confirmed e-mail
 - ☐ Don't reveal if there is a user with this e-mail
 - ☐ Recovery code only active for a short time (e.g. 24h)
- ☐ Consider implementing authentication through third-party providers (e.g. using OAuth)
- ☐ Don't send sensitive information without HTTPS
- ☐ Consider using HTTPS everywhere on website
 - ☐ Use **secure** option on cookies
- ☐ Enable forward secrecy for HTTPS
- ☐ Don't use cookies to store client-side data (use localStorage instead)

Client-side (Frontend)



Implementation Checklist – MUST

- ☐ Separate document structure (HTML) and appearance (CSS)
- ☐ Use `<!DOCTYPE html>`
- ☐ Add `<meta name="viewport">` with appropriate parameters
- ☐ Never execute external or user-supplied JavaScript code
 - ☐ `eval()`
 - ☐ Dynamic `<script>` elements
 - ☐ `innerHTML`

Implementation Checklist – SHOULD

- ☐ Don't use table layout (unless representing table)
- ☐ Use meaningful names for classes/IDs in CSS
- ☐ Minify and combine CSS files to reduce page load time
- ☐ Use caching on CSS files (e.g. 1 year)
- ☐ Check your HTML and CSS code for older snippets of code copied from previous projects / other sites
 - ☐ Unneeded vendor prefixes
- ☐ Clear default styles on all relevant elements
- ☐ Consider using cross-browser frameworks or polyfill instead of manually adding support for older browsers
- ☐ Use validators to check for common errors
- ☐ Use IE conditional comments only to include additional libraries for old IE versions; don't use it throughout code
- ☐ Avoid CSS hacks

Implementation Checklist – SHOULD

- ☐ Consider using CSS preprocessors
- ☐ Use responsive grid layout (e.g. framework or generator)
- ☐ Don't restrict zooming
- ☐ Check how your site behaves under unusual conditions
 - ☐ Screen size
 - ☐ Pixel density
 - ☐ Zoom
- ☐ Avoid using plugins (Flash, Silverlight, Java Applets, ...)



Implementation Checklist – SHOULD

- ☐ Use JavaScript frameworks/libraries to simplify development
- ☐ Use JavaScript tools
 - ☐ Static analysis
 - ☐ Unit testing
 - ☐ Documentation
- ☐ Enable strict mode (“use strict”;
- ☐ Use `textContent` instead of `innerHTML` to set element text
- ☐ Avoid complex DOM changes
 - ☐ Use `DocumentFragment` to make changes outside of DOM
- ☐ Validate user inputs on client when possible
- ☐ Use CORS to make cross-origin AJAX calls
- ☐ Minify and combine JavaScript files
- ☐ Use caching on JavaScript files (e.g. 1 year)
- ☐ Use feature detection instead of parsing `userAgent`

Implementation Checklist – SHOULD

- ☐ Avoid Flash or other plugin-based technologies for new projects
- ☐ Use semantic tags when appropriate
- ☐ Use built-in validation for common field types (email, url, ...)
- ☐ Consider using <canvas> or SVG for graphics
- ☐ Use localStorage instead of cookies to store data on client
- ☐ Use postMessage to communicate between documents (iframes) from different domains
- ☐ Consider using WebSockets or SSE instead of AJAX if data exchanges are frequent and/or initiated by server
- ☐ Remove unneeded vendor prefixes when using JS or CSS features that are already supported by all browsers

Deployment

Implementation Checklist – MUST

- ☐ Don't hardcode sensitive data (such as passwords, keys, tokens) in application code
 - ☐ Don't store such data in revision control
 - ☐ If such data was leaked to public repository – change it immediately (change password, regenerate key, revoke token,...)
- ☐ Implement backup strategy for all important data – both user data (DB) and project data (code, config)
 - ☐ Test that backups work

Implementation Checklist – SHOULD

- ☐ Consider using IaaS or PaaS to host your web applications
- ☐ Implement automated scaling (adding or removing instances) based on load
- ☐ Consider using CDN for static data (JS, CSS, images)
- ☐ Consider using CloudFlare for DDoS protection (and other features)
- ☐ Consider implementing all or some factors from <http://12factor.net/>
- ☐ Use SSH-based tools to access remote servers
 - ☐ Keep your private keys secure
- ☐ Automate build and deployment process
- ☐ Consider using container-based technologies (e.g. Docker) to package and run applications

Web Services / Web APIs

Implementation Checklist – MUST

- ☐ Don't use JSONP web services (unless you trust service provider absolutely)
- ☐ If your web service has external clients, when switching to new version of API, always keep older versions of the service running for some time (at least few months)
- ☐ In OAuth2, always use state parameter to prevent CSRF attacks

Implementation Checklist – SHOULD

- ☐ Cache web service responses when possible
 - ☐ Provide a way for the user to refresh stale caches
- ☐ Reduce number of web service calls by using one big request instead of many small ones
- ☐ Prefer connecting to external web services from the client, not from the server
- ☐ Use modern cross-domain access method when possible (CORS, postMessage)
- ☐ Prefer Access-Control-Allow-Origin: incoming.domain to Access-Control-Allow-Origin: *
- ☐ Prefer OAuth2 Authorization Code (Web Server) Flow



Implementation Checklist – SHOULD

- ☐ Don't add restrictive robots.txt for public sites
- ☐ Consider adding rel="nofollow" to external links that users can create on your site to prevent spam
- ☐ Implement search on your site using either full-text search engine or custom search from one of providers
- ☐ Implement OpenSearch for your site so that users will be able to access your search from their browser
- ☐ Add social buttons on relevant pages on your site
- ☐ For public website, always add relevant <meta name="description" > on all pages with static or predictable content
 - ☐ Don't add <meta name="description" > for pages with user-created content

Performance

Implementation Checklist – MUST

- ☐ Measure performance
 - ☐ Set clear goals
 - ☐ Avoid premature optimization
- ☐ Minify and combine CSS and JavaScript files
- ☐ Enable gzip
- ☐ Cache static content for long time (e.g. 1 year)
 - ☐ If it changes – change URL

Implementation Checklist – SHOULD

- ☐ Avoid extra work if possible
 - ☐ Cache results
 - ☐ Use less accurate calculations
 - ☐ Use external services
- ☐ Create an index for each common query in DB
- ☐ Consider using NoSql database
- ☐ Cache dynamic content for short time (e.g. 1 min – 1 hour) depending on update frequency
- ☐ Use CDN for static content
- ☐ Reduce number and size of cookies
- ☐ Consider supporting HTTP/2 for clients that understand it

Implementation Checklist – SHOULD

- ☐ Optimize HTML structure
 - ☐ Put styles at top
 - ☐ Put scripts at bottom
 - ☐ Use async scripts
- ☐ Use efficient JavaScript
 - ☐ Avoid complex event handlers, especially for frequent events
 - ☐ Avoid live DOM manipulation
- ☐ Use efficient CSS
 - ☐ Avoid generic selectors (*, tags)
 - ☐ Avoid unneeded selectors
 - ☐ Remove unneeded vendor prefixes when unprefixed property is already widely supported
- ☐ Check for and remove old code (e.g. copied from previous projects, other web sites, tutorials)
- ☐ Avoid using complex frameworks to implement simple features

Security

Implementation (meta)Checklist – MUST



Preview

- ☐ Follow security best practices
 - ☐ Check your web site / web application for common vulnerabilities
- ☐ Stay informed about newest security issues
 - ☐ Patch critical security problems ASAP
- ☐ Think about security on all stages of the development process
- ☐ Implement “secure by default” configuration when shipping product
- ☐ Implement security on multiple levels
- ☐ Use the least possible permissions
- ☐ Minimize the time of working with sensitive data
 - ☐ Don't store passwords in plain text – use salted hash instead
 - ☐ Use secure types to store sensitive data in memory
 - ☐ Don't leak sensitive data to logs, backups, ...
- ☐ Use well-known and formally proven cryptography primitives, security protocols, implementation libraries
- ☐ Balance security and usability concerns

Implementation Checklist – MUST

- ☐ Don't try to invent and implement your own cryptography algorithms
- ☐ Use well-known implementations of cryptography primitives
 - ☐ Read documentation carefully
 - ☐ Pass correct parameters
 - ☐ Use latest versions
- ☐ In OAuth2, always use state parameter to prevent CSRF attacks

Implementation Checklist – SHOULD

- ☐ Prefer using existing security protocols over inventing new or modified protocols
- ☐ Use forward secrecy in HTTPS
- ☐ Avoid implementing cryptography primitives in browser
 - ☐ Use HTTPS to load scripts securely
 - ☐ Implement cryptography in browser extensions
 - ☐ Use `window.crypto` for cryptography primitives
- ☐ Prefer OAuth2 Authorization Code (Web Server) Flow



Implementation Checklist – SHOULD

- ☐ Have at least one person on a team with some training in security issues
- ☐ Consider using services of security consulting / penetration testing companies to check your products

Висновки

- Основна мета навчального проекту – навчитись, а не здати абищо найпростіше
- Більше уваги змісту, а не формі
- Вільний вибір технологій
- Взаємодія в команді = важливо
 - Має бути видимою і зафіксованою в артефактах проекту
- Варто дотримуватись рекомендацій / best practices

Виникли питання чи коментарі?

Будь ласка, надсилайте їх на адресу
zhereb@gmail.com

Нагадую – за знайдені помилки
чи незрозумілі місця на слайдах
будуть виставлятись бонусні бали