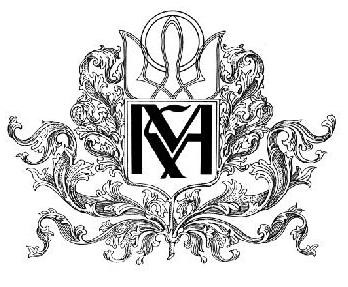
Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики



**Аналіз сучасних підходів до проектування архітектури веб застосунків на прикладі сервісу опитування**

**Текстова частина до курсової роботи**

**за спеціальністю „Комп’ютерні науки” 6.050103**

Керівник курсової роботи

ст.викладач Вовк Н.Є

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(підпис)*

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2020 р.

Виконав студент

Федюченко М.І

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультімедійних систем,

Доцент., к. ф.-м. н. О.П.Жижерун

(підпис)

„\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на курсову роботу

студенту Федюченку Михайлу Ігоровичу факультету інформатики 3-го курсу

ТЕМА Аналіз сучасних підходів до проектування архітектури веб застосунків на прикладі компоненту користувача для сервісу опитувань

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Календарний план
3. Анотація
4. Вступ
5. Основні типи архітектур сучасних веб додатків
6. SPA архітектура та найкращі практики
7. Огляд створеного додатку та використаних практик
8. Висновки
9. Список використаної джерел

Дата видачі „\_\_\_” \_\_\_\_\_\_\_\_\_ 2017 р. Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Завдання отримав \_\_\_\_\_\_\_\_\_\_

(підпис)

**Календарний план виконання роботи:**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Назва етапу курсової роботи | Термін виконання етапу | Примітка |
| 1. | Отримання теми курсової роботи. | 15.10.2019 |  |
| 2. | Огляд технічної літератури за темою роботи | 30.10.2019 |  |
| 3. | Виконання аналізу архітектурних підходів побудови веб-додатків | 15.02.2020 |  |
| 4. | Написання першого розділу | 20.02.2020 |  |
| 5. | Аналіз найкращих практик при побудові односторінкового додатку | 20.03.2020 |  |
| 6. | Написання другого розділу | 30.03.2020 |  |
| 7. | Створення веб-застосунку для опитування студентів на основі SPA архітектури. | 01.04.2020 |  |
| 8. | Написання останнього розділу | 09.04.2020 |  |
| 9. | Корегування роботи згідно із зауваженнями керівника | 10.05.2020 |  |

СтудентФедюченко М.І.

Керівник Вовк Н.Є.

**“\_\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Зміст

*Анотація*

У роботі розглянуто сучасні підходи до побудови веб-застосунків, детально розглянуто SPA-підхід та його переваги. Розроблено веб застосунок на основі SPA-підходу та з використанням досліджених практик, що забезпечують гнучкість застосунку, масштабованість, ефективність та багаторазовість використання.

## ВСТУП

З розвитком людства , об’єми створеної людиною інформації збільшуються і для отримання її виникає потреба створення нових джерел інформації, чи покращення-модифікації вже існуючих. Веб-браузер – це найпоширеніше у світі програма, потрібність котрої є беззаперечною. Певне кожен користувався веб-ресурсами, щоб знайти потрібний матеріал, чи переглядав відео, фотографії та іншу інформацію представлену в мережі інтернет. У цьому нам допомагають веб застосунки, котрі не потребують встановлення, є доступними для будь-якої платформи через більшість браузерів. А також можливість обробки великих об’ємів інформації, що здійснюється за лічені секунди у хмарних сховищах з відображенням користувачу без використання потужностей використовуваного девайсу. Це викликає попит на створення ефективних веб-застосунків, з можливістю їх багаторазового використання, масштабування при виникненні нових потреб та гнучкості використання. Усе це залежить від архітектурної складової та принципів їх побудови.

Актуальність правильної архітектури тільки зростає з бажанням людини ще більше автоматизувати процеси власної діяльності, тому зараз складається тенденція на використання практик, що призводять до загальної ефективності в розробці додатків, технічному обслуговуванні коду та часу розробки.

Метою курсової роботи є визначення основних сучасних підходів до побудови архітектури веб-додатків та з’ясування найкращих практик при створенні односторінкового застосунку.

Текстова частина курсової роботи складається з трьох розділів.

У першому розділі розглядаються основні сучасні архітектурні підходи при побудові веб застосунків, а також їх порівняння. Наведено огляд архітектурних моделей та технологій для їх створення.

У другому розділі окреслюються шляхи підвищення ефективності при побудові SPA веб-застосунку, обґрунтовуються найкращі з розглянутих практик.

Третій розділ присвячено опису архітектурної побудови SPA веб-додатку на основі сервісу опитування.

Постановка задачі.

1. Виконати аналіз архітектурних підходів при побудові веб-додатків, зокрема

* На MPA архітектурі
* На SPA архітектурі
* На Microservice архітектурі
* На Serverless архітектурі

1. Виконати порівняльний аналіз архітектурних підходів.
2. Виконати детальне дослідження практик та архітектурних моделей при побудові клієнтського компоненту веб-додатку на основі SPA-архітектури.
3. Розробити аргументовану структуру клієнтського компоненту з урахуванням досліджених практик та моделей.
4. Зробити висновки щодо актуальності та доречності використаних архітектурних практик.
5. Основні архітектурні підходи при побудові веб-застосунків

Перш за все важливо зрозуміти, що таке веб-додаток і в чому його особливість. Веб-додатком називають тип програми побудованої на основі клієнт-серверної архітектури і його особливість в тому, що сам веб-додаток розташований і виконується на віддаленому сервері, тоді як клієнт отримує тільки результати власної роботи. Робота такої програми заснована на опрацюванні запитів користувача та надсилання результатів клієнту через мережу інтернет, тому і не залежить від операційної системи певного користувача. Зазвичай за відображення результатів запитів та отримання вхідних даних для подальшого відправлення на сервер відповідає браузер (Google Chrome, Opera, Mazila, тощо).

Практично будь-який веб-додаток складається з двох основних компонентів

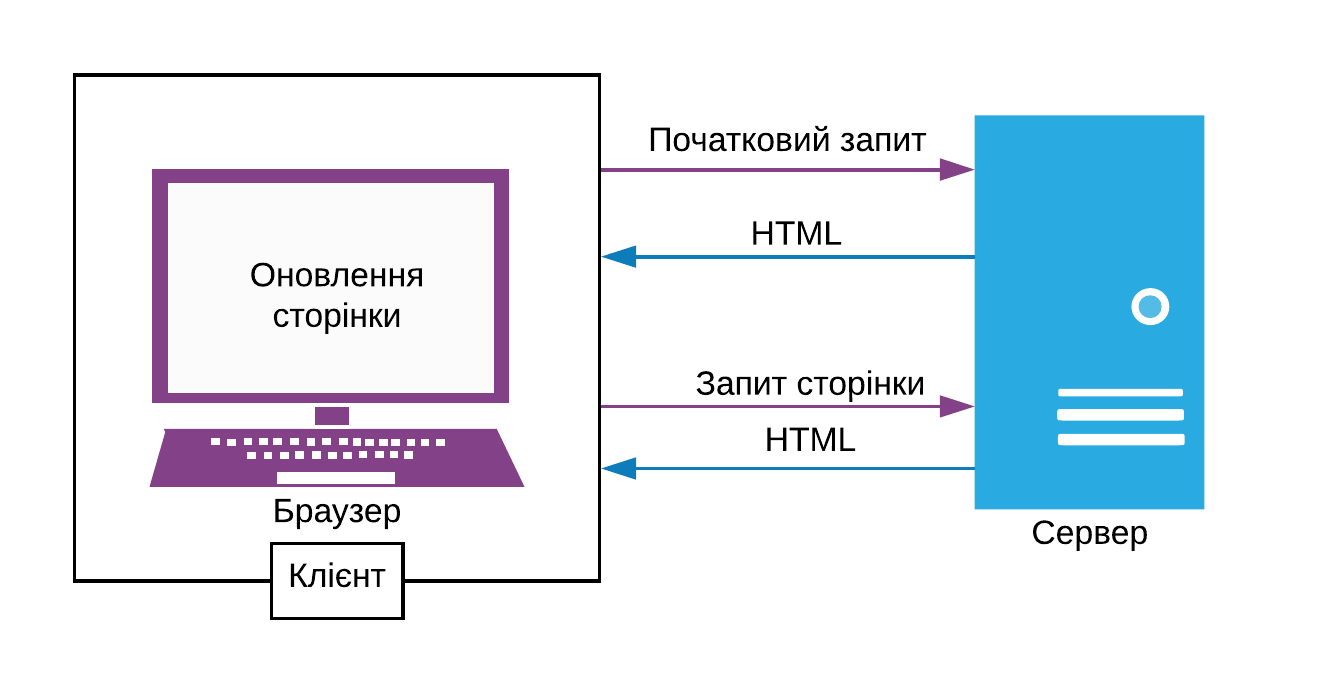
* Клієнтська компонента – код, що знаходиться в браузері та реагує на певний ввід користувача, є представленням функціональності веб- застосунку ,користувач має повний доступ до цієї компоненти.
* Серверна компонента – код , що знаходиться на сервері та відповідає на HTTP запити, відповідальний за збереження інформації, складається щонайменше з бази даних та логіки програми, дана компонента недоступна для користувача.

Архітектура веб-додатків описує взаємодію між компонентами що працюють одночасно, базами даних та системами проміжного програмного забезпечення в мережі Інтернеті. Тобто від того як розподілена логіка програми між серверною та клієнтською компонентами залежить тип архітектури веб-додатку.

* 1. MPA та SPA архітектури
     1. MPA архітектура

Багатосторінковий тип архітектури (MPA) – це традиційний тип архітектури для веб застосунків, при якому виконується завантаження усієї сторінки, та перезавантаження на абсолютно нову, якщо користувач взаємодіє із застосунком.

Багатосторінкові програми - це традиційні веб-програми, які завантажують всю сторінку та відображають нову, коли користувач взаємодіє з веб-додатком. Браузер виконує новий запит до сервера при переході на іншу сторінку і знову виконує завантаження всіх ресурсів і навіть тих компонентів, що можуть повторюватися на усіх сторінках. Відповідно це відображається на продуктивності та швидкості, адже потрібно заново завантажувати ті самі елементи. Основними технологіями для розробки клієнтської компоненти для такого типу архітектури є HTML[] та CSS[].



З діаграми \_\_\_ можна помітити, що при доданні нових функціональних можливостей вартість змін буде досить високою. Розробка клієнтської та серверної компоненти дуже пов’язані між собою, що й ускладнює саму розробку.

Перевагами використання даної традиційної архітектури є можливість легкої оптимізації кожної сторінки для пошукових систем, адже існує можливість додання метатегів[] для будь-якої сторінки.

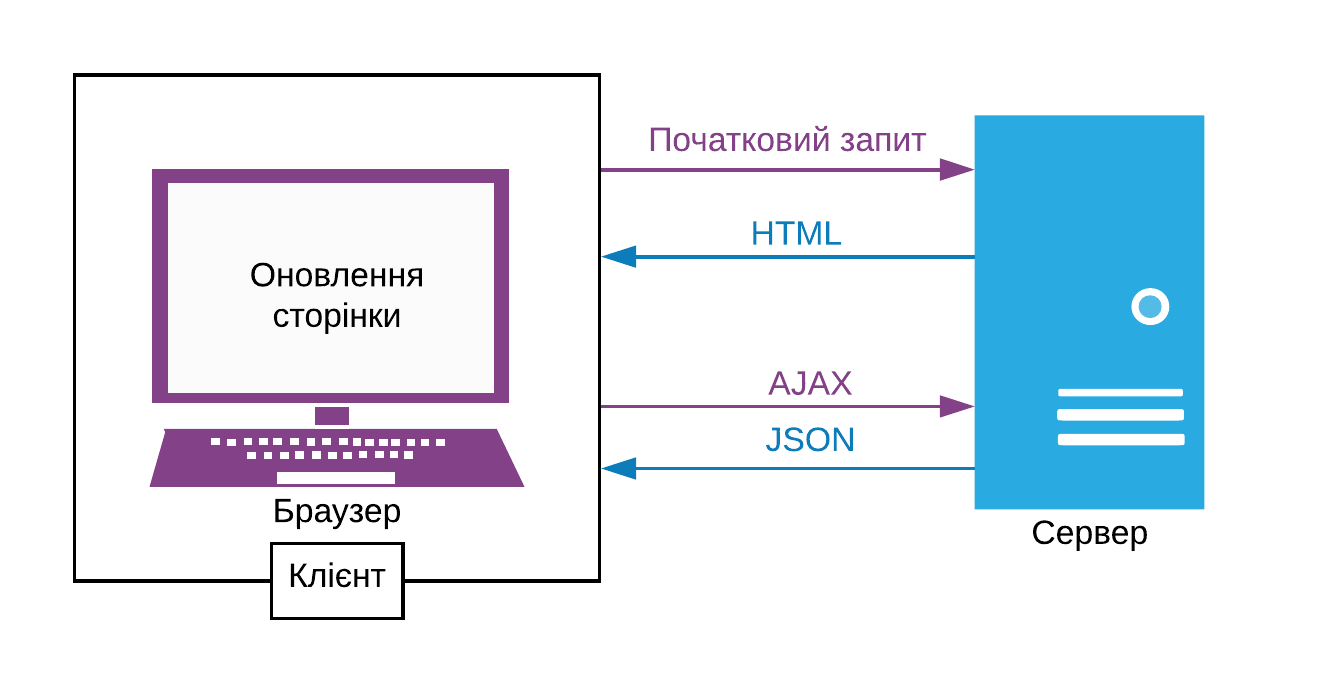
* + 1. SPA архітектура

До двохтисячних років розробники шукали ефективний підхід для створення таких веб-застосунків, що були б схожими на програми для настільних комп’ютерів. Були випробовані різні технології, наприклад Java applets[], Adobe Flash[], and Microsoft Silverlight[]. Хоч технології були й різними, однак вони мали одну спільну мету – перенести потужність й можливості стандартної настільної програми у крос-платформене середовище веб-браузерів.

Можливість реалізації даної ідеї почали існувати на початку двохтисячних років, з активної інтеграцією технології AJAX. Котра бере початок з контролю ActiveX у браузері Microsoft’s Internet Explorer, що використовувався для асинхронного надсилання й отримання даних. Це призвело до того що така контрольна функціональність була офіційно додана до більшості браузерів, як XMLHttpRequest (XHR) API.

Концепція SPA архітектури підняла технологію веб-розробки на новий рівень розширивши сторінкові методи маніпуляції AJAX на увесь застосунок.

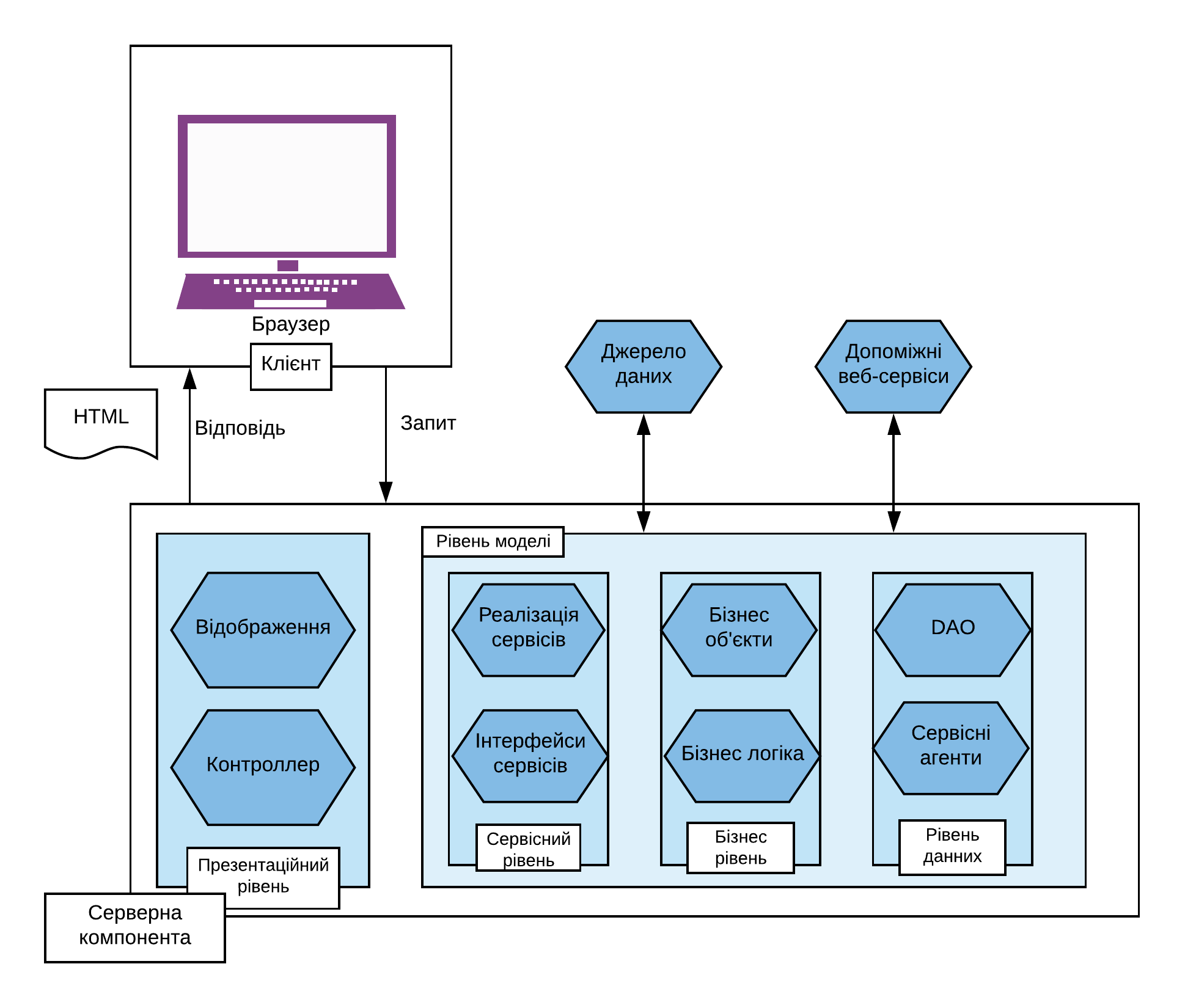
При використанні SPA архітектури існує одна повна HTML-сторінка, що складається з багатьох відображень, котрі є частиною DOM[]. Після початкового завантаження сторінки усі необхідні інструменти для створення відображень завантажуються та готові до використання. За потреби, нове відображення генерується локально у браузері і динамічно додається в DOM за допомогою JavaScript.



SPA архітектура передбачає використання декількох підходів для відображення даних із серверної компоненти, наприклад коли відображення генерується повністю на стороні клієнтської компоненти, а дані надсилаються та отримаються тільки під час бізнес-транзакцій, що виконуються асинхронно через XHR API. Зазвичай формат обміну даних предсталяє собою JSON[]. Як і у випадку генерації даних серверною компонентою, вихідний HTML має спеціальні блоки для заповнення даними. Але це не повний шаблон HTML для повної сторінки, а тільки частина відображення.

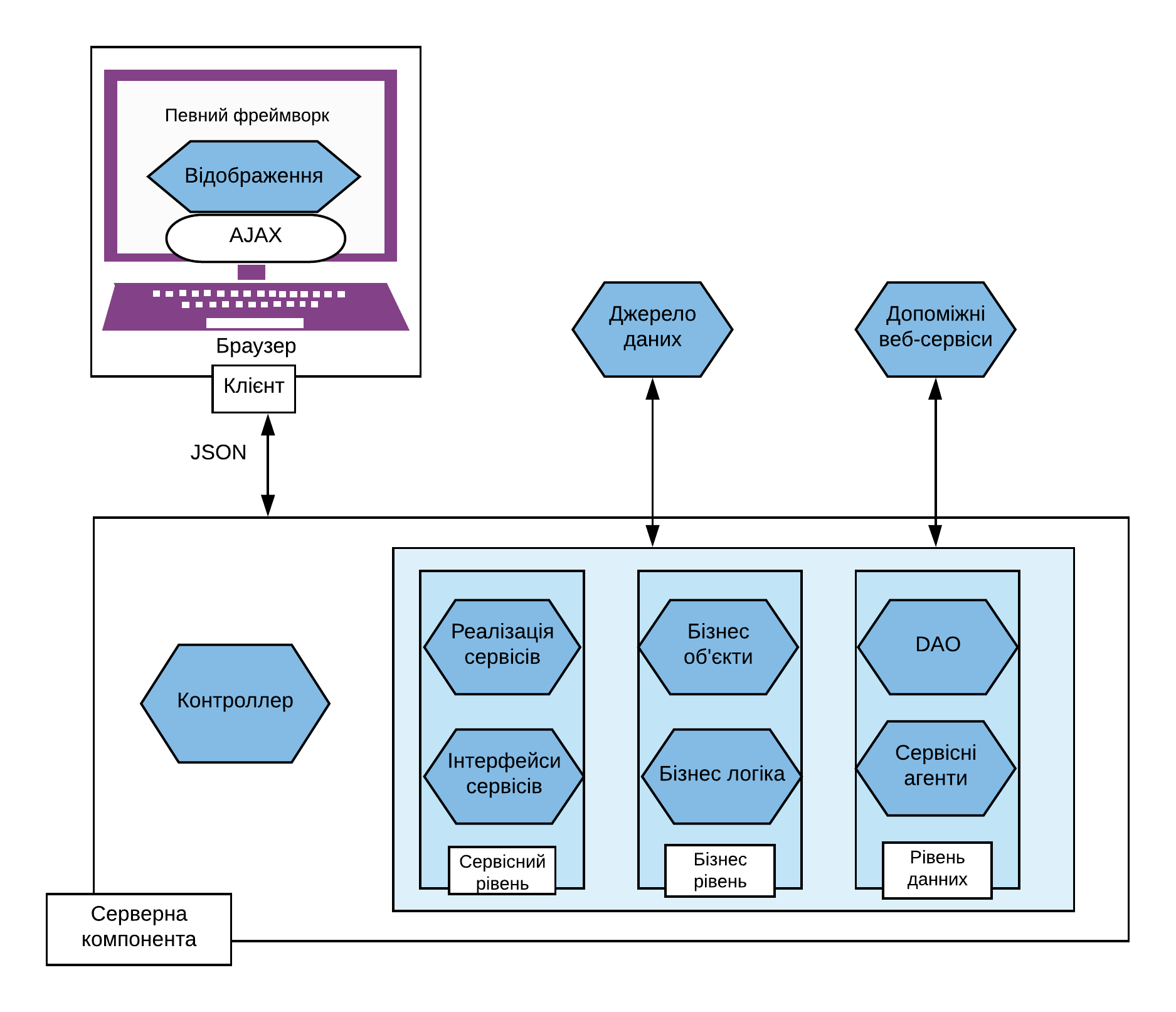
* + 1. Порівняння SPA та MPA архітектурних підходів

Проаналізуємо архітектурний підхід на основі MPA для традиційного веб-застосунку.



Бачимо, що кожний запит на нове відображення призводить до подорожі на сервер. Запит надсилається серверній компоненті, якщо потрібні нові дані. На стороні серверної компоненти запит перехоплюється контролером, що взаємодіє з рівнем моделі за допомогою сервісного рівня, котрий визначає усі потрібні компоненти для виконання завдання рівня моделі. Дані вилучаються об’єктом доступу до даних (DAO[]), або сервісним агентом. Необхідні зміни виконуються на бізнес рівні. Далі контроль передається назад презентаційному рівню, де логіка відображення певним чином формує отримані дані для обраного відображення. Після об’єднання даних з відображенням, новостворене відображення повертається браузеру у якості HTML сторінки. Браузер оновлює сторінку для показу нових даних.

SPA підхід майже такий самий, як і традиційний MPA. Ключові зміни полягають у наступному: відсутність повного оновлення сторінки браузером, логіка відображення знаходиться в клієнті, а транзакції на серверній компоненті можуть бути представлені лише даними.



З діаграми \_\_\_ можна побачити, що серверна компонента більше не бере участі у відображенні даних, якщо не здійснюється часткова візуалізація.

Можна визначити такі переваги використання SPA архітектури над MPA:

1. Легше обслуговування програмного коду – принципи побудови SPA веб-застосунків допомагають розділити код на різні сфери, наприклад JavaScript код зберігається поза HTML окремими частинами. Також зменшується кількість зв’язків між клієнтською компонентою та серверною.
2. Динамічне перемальовування частин екрану – оскільки SPA завантажує структуру HTML сторінки заздалегідь, для отримання нових сторінки з сервера не потрібно виконувати запит на руйнування наявної.
3. Скорочений часу очікування – чим менше людина витрачає часу на очікування, тим більше шанс, що вона залишиться на сайті й повернеться у майбутньому.
4. Генерація відображення клієнтською компонентою – код , що керує логікою відображення інтерфейсу для користувача зберігається на стороні клієнта замість сервера, що дає перевагу в незалежній підтримці та оновленні обох компонент.
5. Швидкі та ненавантажені транзакції – виконання асинхронних транзакцій роблять тип цієї архітектури дуже швидким, адже після початкового запиту тільки дані надсилаються та отримаються серверною копмонентою. Вміст, що повертається при MPA підході включає в собі і HTML розмітку.
   1. Мікро сервісний тип архітектури

Мікро сервісна архітектура – це форма сервісно-орієнтованої архітектури технологія розробки програмного забезпечення, яка структурує додаток як колекцію незалежних мікро-сервісів, що комунікують між собою використовуючи певні механізми, зазвичай HTTP/REST. Кожен мікро-сервіс є самодостатнім і повинен реалізувати конкретну бізнес-потребу, тому його можливо створити й запустити незалежно від інших , навіть на зовсім іншій мові програмування.

Усі більше постачальників послуг (Twitter, eBay, Netflix та Amazon) надають перевагу мікро сервісній архітектурі. Наприклад до 2012 року Walmart Canada використовували монолітний тип архітектури, що в піках активного використання сервісу призводило до обробки шести мільйонів сторінок на хвилину, що займало багато часу, тому й зменшувало кількість купівель через сервіс. Після оновлення архітектурного підходу на мікро сервіси їм вдалося збільшити дохід на двадцять відсотків за рахунок того, що час простою було скорочено і компанія отримала змогу використовувати дешевші сервери, збільшити кількість покупок через мобільні пристрої на дев’яносто вісім відсотків.

* + 1. Філософія мікро сервісної архітектури

Філософія мікро сервісів дуже близька до філософії сімейства відомих операційних систем Unix : (“Зробіть одну річ і зробіть її добре”  by [Peter (Raymond) H. Salus](https://en.wikipedia.org/wiki/Peter_H._Salus" \o "Peter H. Salus) in A Quarter-Century of Unix (1994):[[1]](https://en.wikipedia.org/wiki/Unix_philosophy#cite_note-taoup-ch1s6-1))

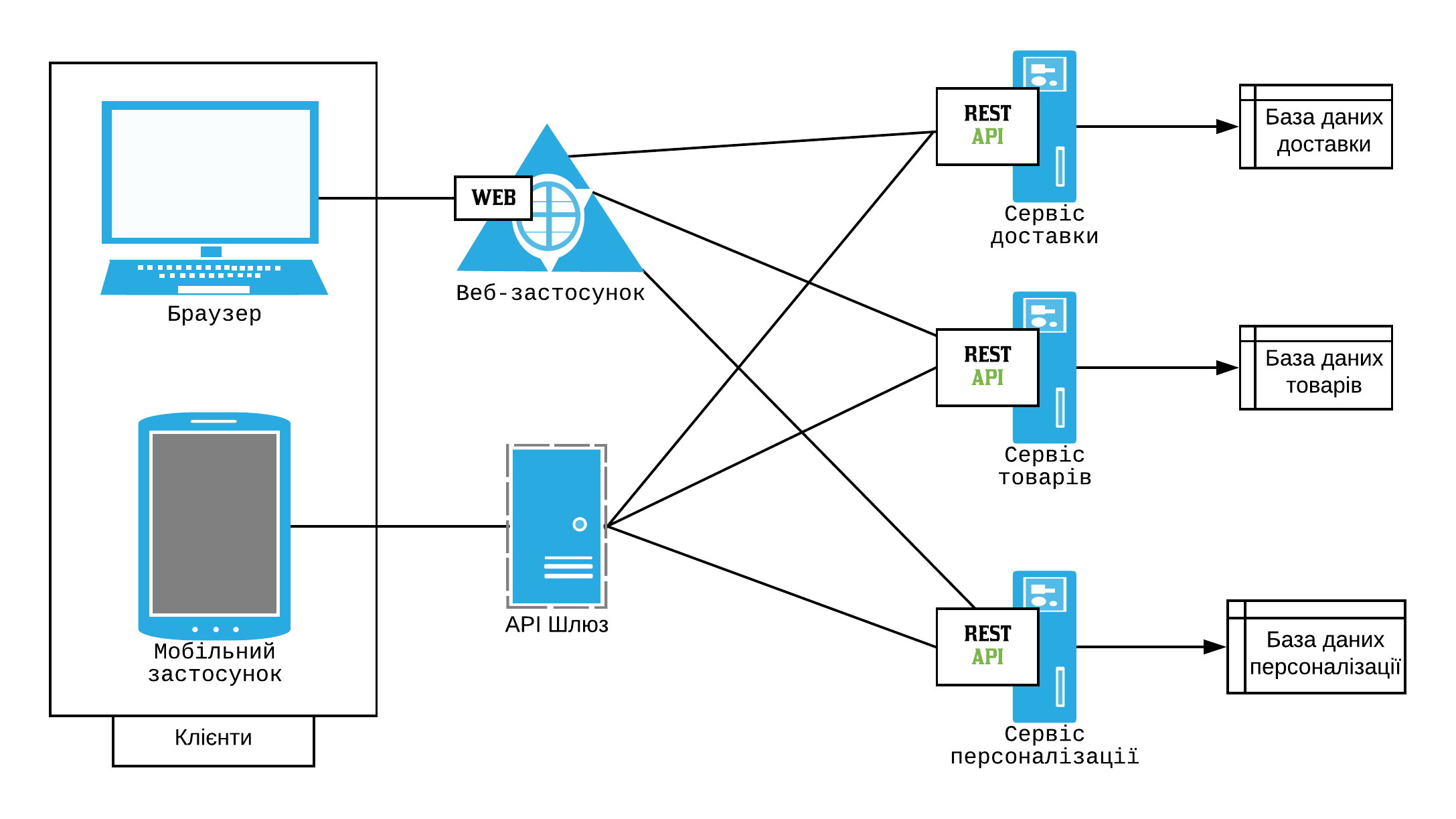
Архітектура мікро сервісів використовує бібліотеки, однак її головний спосіб розбиття застосунку шляхом поділу його на сервіси. Перевага незалежності розвертання кожного сервісу зумовлена тим, що при зміні логіки одного з них не потрібно буде виконувати повторне розвертання усіх сервісів програми, а тільки того, що зазнав змін. Даний архітектурний підхід допомагає не порушувати інкапсуляцію компонентів за рахунок використання віддалених викликів сервісами програми.

Архітектурний підхід починається з різних типів клієнтів, що можуть бути будь-якими пристроями, котрі намагаються виконувати певні можливості керування, наприклад пошук, налаштування тощо.

Оскільки клієнти не звертаються до сервісів напряму, шлюз API є вхідною точкою, що виконує переадресацію запитів клієнтів у відповідні мікро сервіси. Першою перевагою використання API шлюзу є те, що ми можемо виконувати зміни над сервісом без попередження клієнта, а також що сервіси можуть використовувати протоколи спілкування не пристосовані для вебу.

Існують два основні типи повідомлень для комунікації сервісів між собою:

* Синхронні повідомлення – у випадку, якщо клієнт очікує відповіді від сервісу, прийнято використовувати REST, адже це розподілене середовище у котрому кожна функціональність представлена відповідним ресурсом для проведення операцій
* Асинхронні повідомлення – у випадку, коли клієнт не очікує відповіді. Зазвичай використовуються такі протоколи, як AMQP, STOMP, MQTT, чия основа повідомлень є чітко визначеною і ці повідомлення мають бути сумісними між реалізаціями.



Особливість котру ми спостерігаємо на рисунку полягає в тому, що кожній бізнес потребі було виділено окремий мікро сервіс, для обробки власних даних та виконання різних функціональних можливостей. Ці мікро сервіси мають власне середовище розгортання для виконання своїх функціональних потреб та власні бази даних для фіксації даних. Усі мікро сервіси спілкуються між собою через сервер, що не зберігаю стан, який є або шиною повідомлень, або REST клієнтом. Усі внутрішні точки з'єднані з шлюзом API. Отже, кожен, хто підключається до шлюзу API, автоматично підключається до повної системи.

* + 1. Переваги та недоліки мікро сервісної архітектури

Тенденція на мікро сервісну архітектуру набуває популярності, адже в міру збільшення складності програмного забезпечення, можливість розбиття функціональних областей застосунку на колекції незалежних сервісів може принести багато переваг розробникам програмного забезпечення, що передбачають:

1. Масштабованість – мікро сервіси, що складаються з компонентів легко можуть бути інтегровані з іншими програмами чи сервісами через інтерфейси, наприклад REST.
2. Прискорена розробка програмного забезпечення – через можливе використання різних мов програмування стає ширшим вибір кола розробників. А також різні команди можуть працювати паралельно над окремими компонентами, без очікування результатів один-одного. Інші ж зацікавлені сторони можуть в цей час вдосконалювати вже реалізовані компоненти.
3. Можливість повторного використання – оскільки мікро сервіси не є одним проєктом і розташовані навколо певних бізнес потреб, то легко можуть бути повторно використаними в інших проєктах, що також і зменшує час розробки.
4. Краще розуміння кодової бази – оскільки кожен сервіс являє собою єдину функціональну область або бізнес потребу, то це покращує подальшу підтримку та налагодження.
5. Ефективна ізоляцій можливих проблем – немає потреби у постійному переході між структурними шарами застосунку, розробники знають, де саме потрібно шукати проблеми, котра потребує вирішення. Якщо зачіпається один сервіс, його можна легко видалити або вирішити без пошкодження інших частин програм

Мікро сервісний тип архітектури, як і будь який інший тип не є ідеальним, виявлено і певні недоліки:

1. Складність інтеграційного тестування – зумовлене можливістю існування компонентів що належать іншим системам, чи середовищам.

Адже перед початком тестування потрібне підтвердження кожного залежного сервісу.

1. Комунікація – оскільки канали зв’язку мають бути визначені у певних інтерфейсах, можливе виникнення помилок між мікро сервісами при недотриманню визначеного інтерфейсу.
2. Потреба у належному моніторингу – кожний сервіс залежить від власного API та платформи розгортання й для належного моніторингу потрібно слідкувати за багатьма сутностями та мати контроль над усією інфраструктурою.
   1. Serverless архітектура

Serverless архітектура – це підхід до створення та запуску веб-додатків та сервісів без необхідності керування усією інфраструктурою. Такий підхід включає у себе використання сторонніх послуг “BackEnd as Service ” (BaaS), або коду, що контролюється тимчасовими контейнерами на платформі “Function as a Service” (FaaS). При використанні даної архітектури збільшується залежність від постачальників сторонніх послуг.

До 2006 року більшість розробників використовували власні фізичні сервери, розміщені у центрах обробки даних, для розгортання серверної компоненти власних веб-застосунків. Але в жовтні 2006 року новий відділ Amazon Web Services (AWS) американської компанії Amazon оголосив про запуск нової технології Elastic Compute Cloud (EC2). Що дозволило стороннім компаніям використовувати EC2, як інфраструктуру для сервісів (IaaS). Використання IaaS – можна визначити, як інфраструктурний аутсорсинг. EC2 дав змогу орендувати обчислювальну потужність, запускати свої власні серверні застосунки замість того, щоб купляти власне обладнання для розгортання цих застосунків.

* + 1. Інфраструктурний аутсорсинг

Будь-яка форма використання IaaS, або ж інфраструктурний аутсорсинг принаймні частково передбачає ідею економії від масштабу, тобто що виконання тієї самої дії у сукупності дешевше, ніж сума самостійного виконання цих дій незалежно один від одного, завдяки ефективності, що можна використати.

Інфраструктурний аутсорсинг відображає п’ять переваг IaaS:

* Підвищена гнучкість масштабування – адже стає доступним широкий вибір подібних ресурсів, котрі можна використати, а потім ефективно утилізувати без зайвих залишкових даних.
* Швидкість розробки застосунку – скорочення часу на розробку починаючи від концепції до релізу.
* Дешевша вартість ресурсів – менша вартість за ті ж самі можливості.
* Зменшення ризиків – від суб’єкта використання вимагається менше знань та надається жива операційна підтримка.
* Зниження вартості робочої сили – для виконання робіт над інфраструктурою потрібно менше людей та часу.

Однією із найсучасніших форм інфраструктурного аутсорсингу і став Serverless підхід.

1.3.2 Serverless технології

Як і будь-яку інший підхід – Serverless поділяються на дві паралельні сфери:

1. Поняття “Serverless” вперше було використано для опису програм, де за управління логікою та станом на сервері відповідають сторонні хмарні-застосунки та сервіси. Зазвичай це програми мобільні додатки, або веб-застосунки побудовані на SPA архітектурі, що використовують масштабну екосистему хмарних баз даних, служб аутентифікації та інших хмарних технологій. Такі типи сервісів і називають “BackEnd as a Service” (BaaS).
2. А також Serverless – може бути застосунок, де логіка серверної компоненти все ще написана розробником, але на відміну від традиційної архітектури, вона запускається в обчислювальних контейнерах, що не зберігають стан, викликаються при виникненні подій, а також є тимчасовими і керуються сторонніми сервісами. Такий тип називається “Function as a Service” (FaaS).

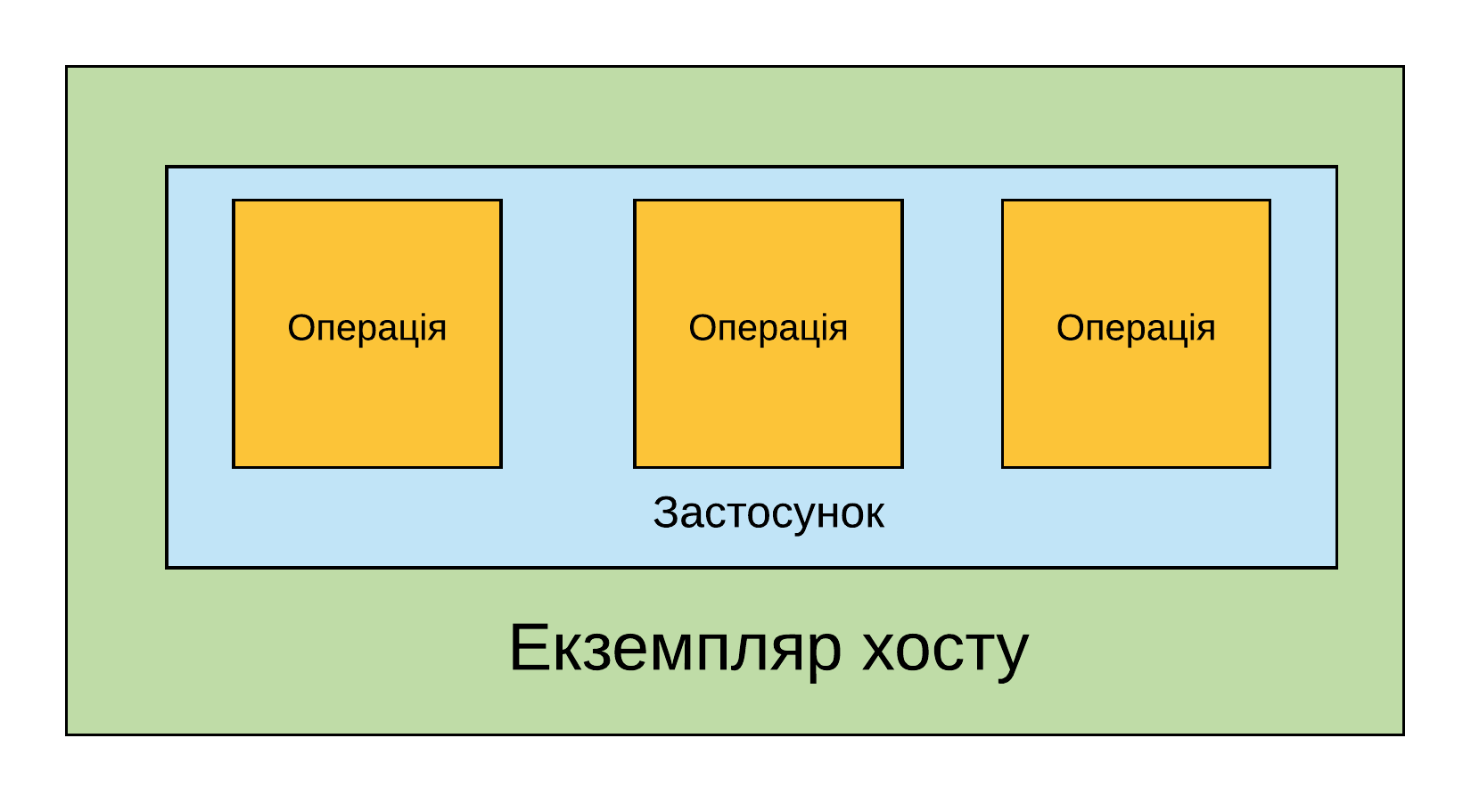
1.3.2.1 BackEnd as a Service

Сервіси BaaS – це сторонні не вбудовані компоненти, котрі можна додати до програми за допомогою API. Вони прибирають потребу у створені та використанні власного сервера посередника, а також керують компонентами, що зберігають дані від нашого імені. Також такі сервіси дозволяють покластися на сторонню, вже реалізовану логіку. Наприклад багато застосунків, в котрих наявна автентифікація мають схожу логіку, тому для збереження часу та ресурсів використовуються такі продукти, як “OAuth”, чи “Amazon`s Cognito”, що дозволяють веб-додаткам мати аутентифікацію, але без потреби створення коду для реалізації цієї функції.

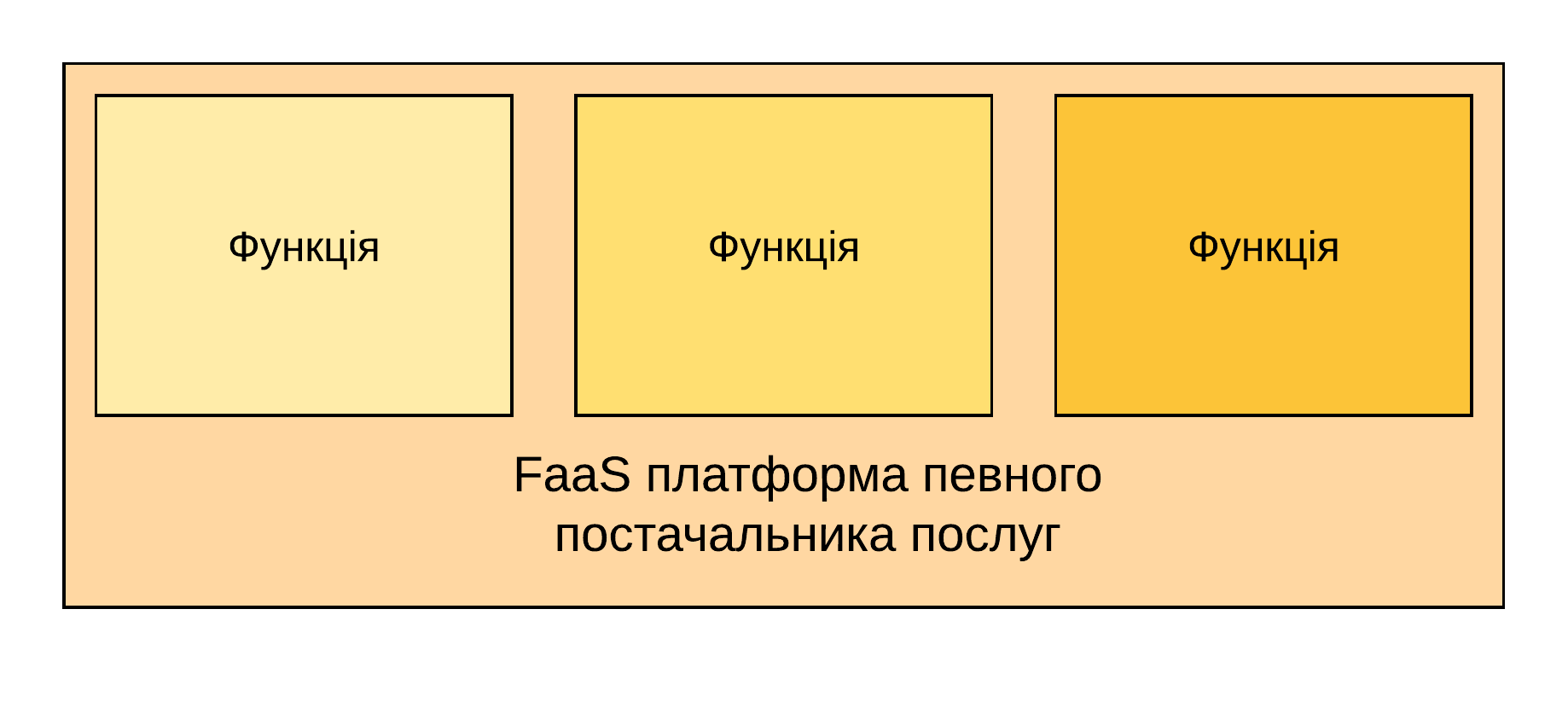
BaaS дає змогу розробникам зосередитися на створенні клієнтської компоненти, що як результат дозволяє швидше створювати та запускати нові веб-додатки.

1.3.2.2 Function as a Service

FaaS - це новий спосіб створення та розгортання програмного забезпечення серверної компоненти, що орієнтований на розгортання окремих функцій, чи операцій. При стандартному розгортанні серверної компоненти, зазвичай починають з хост-екзмепляром і розгортають додаток у хості, як на діаграмі \_\_\_.



У середовищі FaaS ми завантажуємо код для нашої функції провайдеру FaaS, а вже сам провайдер виконує усе інше для забезпечення ресурсів, керування процесами, тощо, як на діаграмі \_\_\_\_\_\_. Тобто йде зосередження на окремих операція, чи функціях, що відображають логіку програми. FaaS, як підхід не вимагає коду для певного фреймворку, чи бібліотеки.

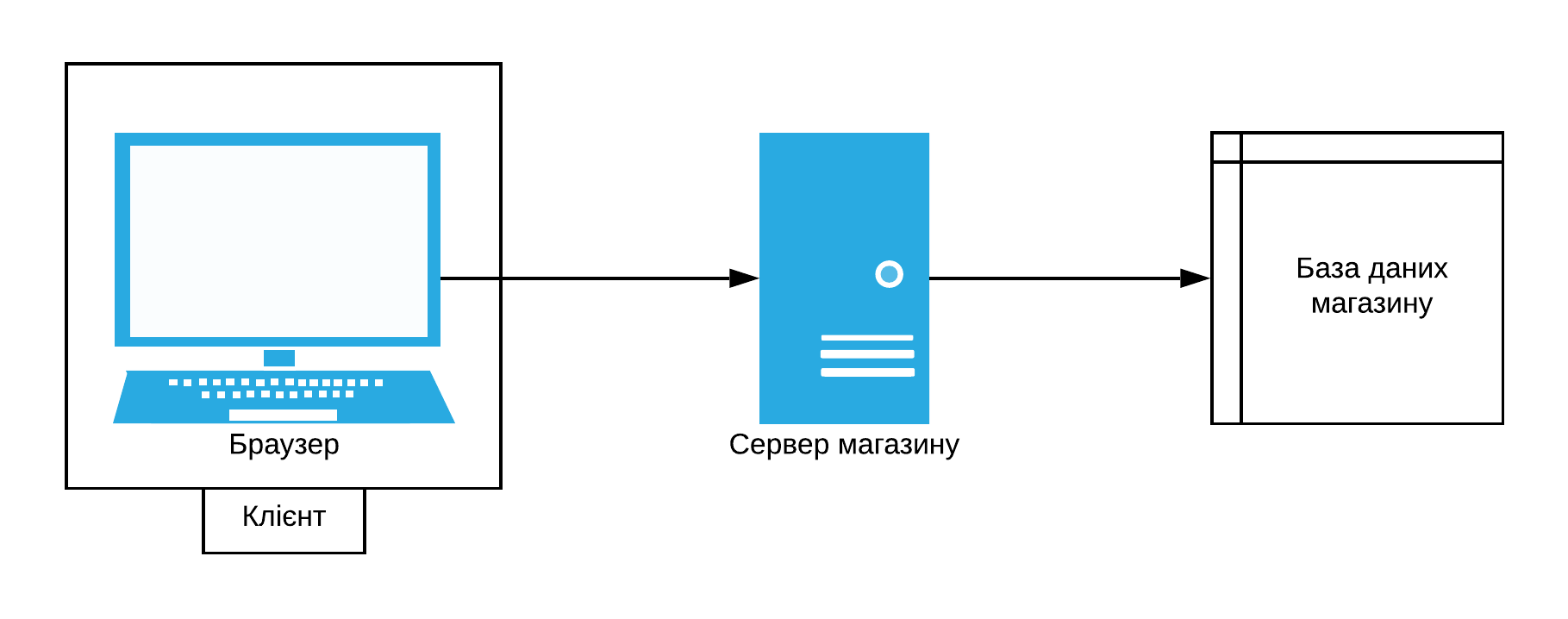


Функції FaaS - це звичайні програми, якщо йдеться про мову програмування та навколишнє середовище. Платформа FaaS налаштована для прослуховування конкретної події для кожної операції. Коли ця подія відбувається, платформа постачальника створює лямбда-функцію, а потім викликає її при запуску події. Наприклад в Amazon AWS такими типами подій є оновлення файлів у “S3”, час запланованого завдання. Більшість провайдерів дозволяють запуск функцій у відповідь на вхідні HTTP-запити. На прикладі того ж самого Amazon AWS за таку поведінку відповідає шлюз API. FaaS функції, як правило, обмежені в часі для реагування на відповідь. Тобто для певних класів довготривалих завдань виникає потреба створення кількох скоординованих функцій.

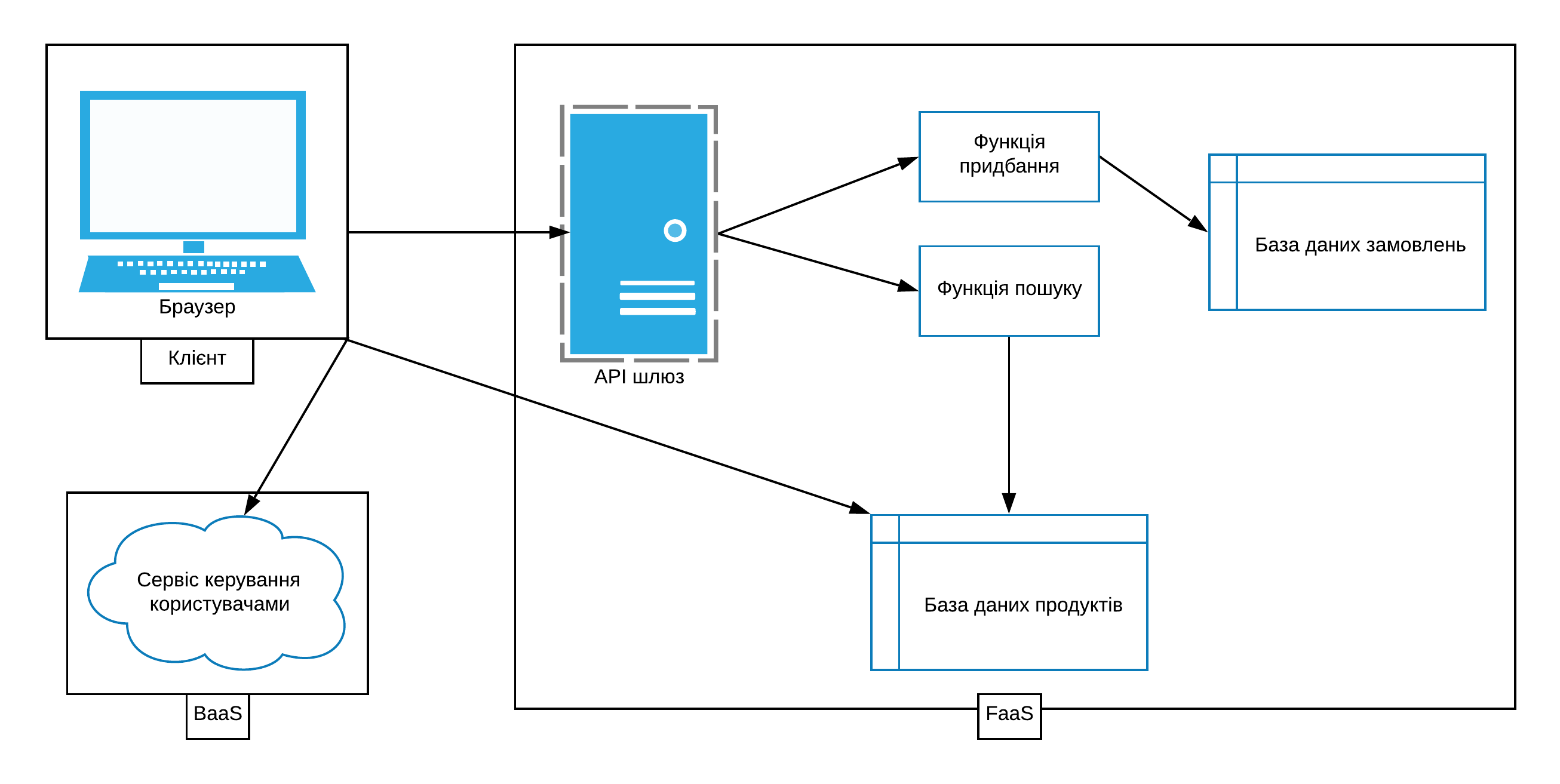
1.3.2.3 Приклад використання FaaS та BaaS

Ключовим моментом, що поєднує FaaS та BaaS та відносить їх до Serverless архітектури є те, що немає потреби керувати процесами у серверної компоненти. Уся логіка, котру ми можемо закодувати власноруч, або інтегрувати у сервіс стороннього постачальника – працює в гнучкому середовищі, як і стан, що зберігається в цьому ж середовищі.

На прикладі типового веб-застосунку для замовлення товарів можна зазначити основні переваги використання Serverless підходу.



Використовуючи стандартну трьох-рівневу архітектуру на діаграмі \_\_\_\_ можна побачити, що велика частина логіки – пошук, навігація, транзакції, керування користувачами – реалізовано серверною компонентою, що робить клієнтську компоненту дуже несвідомою.



На діаграма \_\_\_ використовуючи Serverless архітектуру, ми замінили логіку керування користувачами на сторонній сервіс BaaS. Через використання іншого сервісу BaaS ми дозволяємо клієнту прямий доступ до підмножини бази даних, що повністю розміщена на сторонньому сервісі. Можна помітити, що тепер певна логіка, котра знаходилася на сервері магазину тепер знаходиться в межах клієнта, що дозволяє відстежувати сеанси користувача, конвертувати отримані дані з бази даних для представлення клієнту. Тепер клієнт може мати архітектуру односторінкового веб-застосунку.

Оскільки ми хочемо зберегти деякий функціонал для пошуку товару на серверній компоненті, але без постійно працюючого сервера ми імплементуємо FaaS функцію, яка дає відповіді на HTTP запити через шлюз API. Після цього новостворена FaaS функція як і клієнт може використовувати базу даних продуктів. FaaS функцію, що реалізує придбання товару ми теж інтегруємо на серверну компоненту, а не на клієнтську в міркуваннях безпеки.

Підкреслити те, що у початковій версії усім потоком керування, безпекою керував серверний застосунок. У Serverless версії уся логіка розбита між сервісами, що орієнтовані на власні бізнес потреби – можна помітити схожість ідеї з мікро сервісною архітектурою.

1. Клієнтський компонент при використанні SPA архітектури

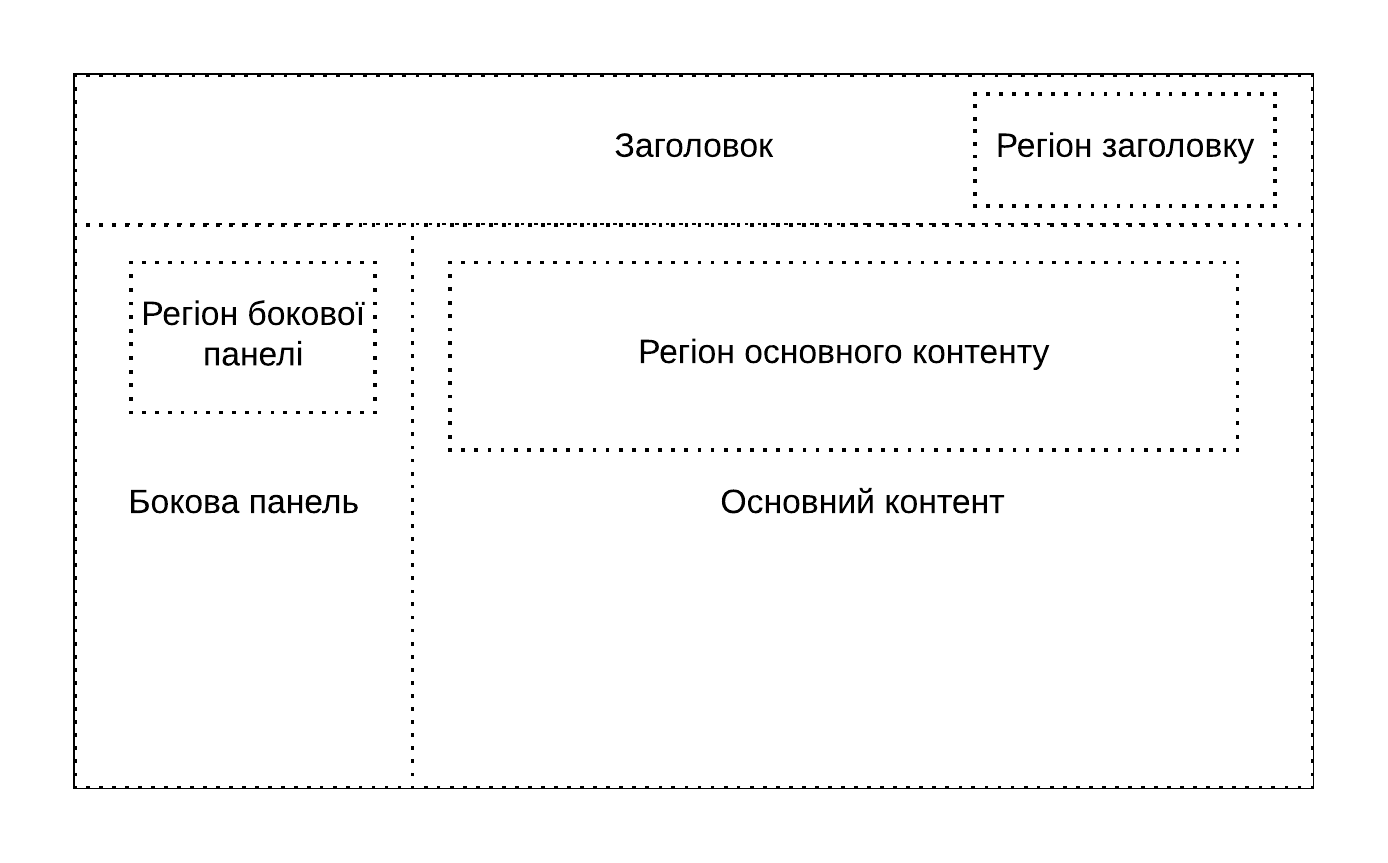
Клієнтська компонента побудована на основі SPA архітектури пропонує набагато більше можливостей, ніж традиційний інтерфейс користувача для веб-сайту. Адже якісно написаний SPA-клієнт – це практично настільна програма.

Розробники що мали бачення успішного переходу від традиційних MPA веб-застосунків до SPA, зрозуміли, що застарівші практики та структури повинні змінюватися. Фокус завдань перейшов на архітектурний талант, дисципліну та тестування на стороні клієнтської компоненти.

* 1. Головна оболонка

Побудова клієнтської компоненти SPA починається зі створення початкового файлу HTML або оболонки. Цей файл HTML завантажується тільки один раз і служить вихідною точкою для решти програми. Браузер виконує це євине завантаження. Подальші частини програми завантажуються динамічно та незалежно від оболонки, без повного перезавантаження сторінки. Модуль оболонки надає форму та структуру веб-додатку. Можливі модулі взаємодії додаються до оболонки за допомогою API. Вона ж погоджує модулі функцій з бізнес-логікою та універсальними інтерфейсами браузера, такими як URI[] або Cookies[]. Наприклад, коли користувач своїми діями змінює стан програми, оболонка координує внесені зміни.

Зазвичай на початку розробки оболонка має мінімальну структуру і складається з одного пустого тегу[] DIV[], що тримає у собі решту програми. Початковий контейнер DIV може містити дочірні контейнери, що візуально поділяють екран на логічні зони як це продемонстровано на діаграмі \_\_\_\_. Зони допомагають розділити видиму ділянку на керовані фрагменти вмісту. Але наприклад React виконує виправлення DOM замість заміни конкретних зон.



Сторінками веб-додатку – є незалежні розділи, що ще називають відображеннями. Відображення – це фактично частина програми з котрою взаємодіє кінцевий користувач.

Для MPA архітектури сторінки є закінченими HTML структурами як можна побачити на діаграми \_\_\_\_



Для SPA архітектури та ж частина веб сайту матиме повний HTML файл із заповнювачами для фрагментів HTML, котрі зберігаються у файлах вадображеннях як на діаграмі \_\_\_\_.



Можна зазначити, що головна оболонка (Shell) відповідає за:

1. Координування модулів відображень
2. Управління станом клієнтської компоненти веб-додатку.
3. Відображення та управління контейнерами відображень.
   1. DOM та Virtual DOM

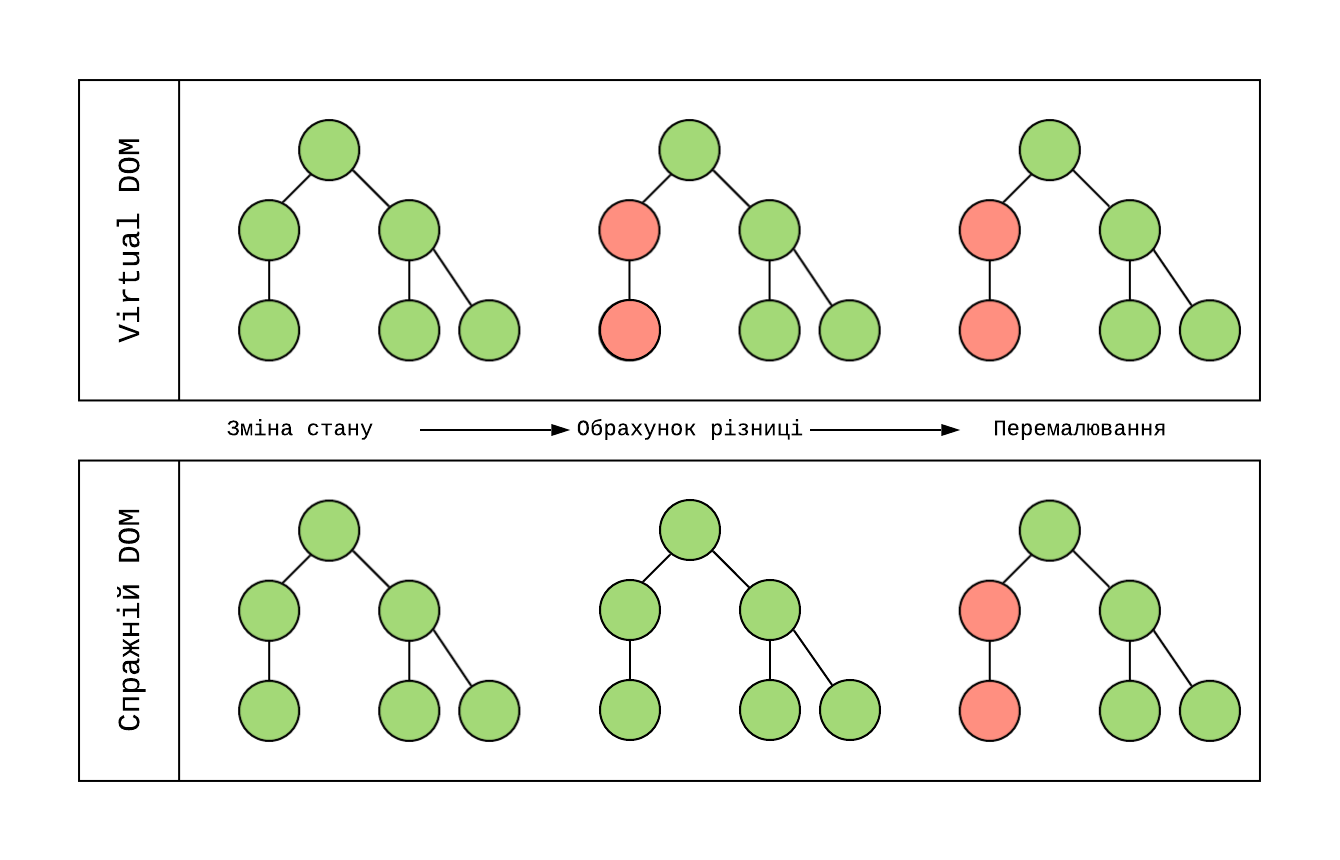
DOM або ж World Wide Web Consortium Document Object Model – це платформа та мовно-нейтральний інтерфейс, який дозволяє програмам динамічно отримувати доступ та оновлювати вміст, стиль та структуру дукументів. Тобто DOM представляє інтерфейс вашої програми. Кожного разу коли відбувається зміна стану інтерфейсу програми , DOM оновлюється, щоб представити цю зміну.

DOM представлений у вигляді такої структури даниї, як дерево. Тому зміни та оновлення відбуваються швидко. Але після зміни потрібно повторно перемалювати оновлений елемент та усі його дочірні елементи, що робить DOM повільним. Тобто чим більше в програмі компонентів інтерфейсу користувача, тип дорожчими будуть оновлення DOM.

Тому виникла концепція Virtual DOM , що працює значно ефективніше за справжній DOM. Віртуальний DOM - це лише віртуальне представлення DOM. Кожного разу, коли стан нашої програми змінюється, віртуальний DOM оновлюється замість реального.

Коли нові елементи додаються до інтерфейсу користувача, створюється Virtual DOM. Кожен елемент є вузлом на цьому дереві. Якщо стан будь-якого з вузлів змінюється, то створюється новий Virtual DOM. Далі виконується порівняння з попереднім Virtual DOM.

Після цього Virtual DOM обчислює найкращий метод внесення наявних змін у справжній DOM як на діаграмі \_\_\_\_. Що забезпечує мінімальну кількість операцій над реальним DOM.



* 1. React для SPA веб-додатку

React – це бібліотека JavaScript для розробки інтерфейсів користувача. Компоненти інтерфейсу користувача створюється за допомогою JavaScript, а не спеціальної мови шаблонів, тому такий підхід називають створенням сумісних інтерфейсів, що і є основою філософії React.

Компоненти React UI – є самодостатніми , функціональними блоками для певної специфіку. Такі компоненти мають як і динамічну логіку, так і візуальне предсталення. React використовує компонентну архітектуру – при використанні котрої легше підтримувати, розширяти та заново використовувати компоненти ніж при монолітній архітектурі.

* + 1. Переваги React

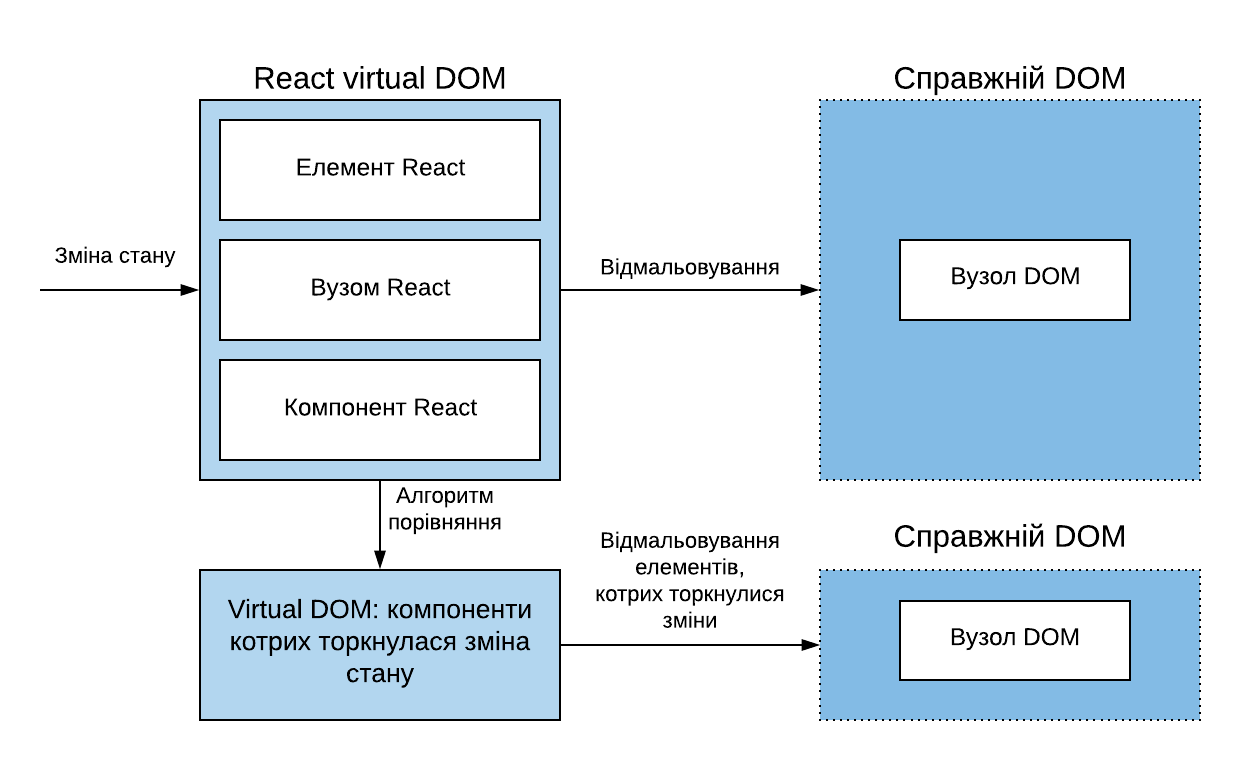
React було створено для вирішення проблеми у розробці та керуванні складними інтерфейсами користувача для веб-додатків. У дискусії на React Podcast [React Podcast, “8. React, GraphQL, Immutable & Bow-Ties with Special Guest Lee Byron,” December 31, 2015, <http://mng.bz/W1X6>.] згадується, що розробник React вирішував проблему наявності кількох джерел даних для оновлення да автоматичного заповнення у Facebook. Дані надходили асинхронно із серверної компоненти. Визначати, куди потрібно вставити нові рядки, щоб використовувати повторно елементи DOM, ставало все складніше. Уолк вирішив щоразу генерувати елементи DOM. Це рішення було елегантним у своїй простоті: інтерфейси як функції. Пізніше з'ясувалося, що генерування елементів у пам'яті відбувається надзвичайно швидко. Команда React придумала алгоритм, який дозволяє уникнути зайвих проблем з DOM. Це зробило React дуже швидким (та дешевим з точки зору продуктивності). Висока ефективність та зручна архітектура на основі компонентів – стало виграшної комбінацією. Можна зазначити такі переваги використання React:

1. Швидкість інтерфейсу користувача - React забезпечує надзвичайну продуктивність завдяки своєму Virtual DOM та алгоритму розумного узгодження.
2. Простіші програми – компонентна архітектура, що є в React в комбінаціїї з простим JavaScript, зручні водночас потужні й прості абстракції DOM.
3. Невелика кількість коду для створення – завдяки розвинутій екосистемі компонентів розробники отримують доступ до великої кількості різноманітних бібліотек.
   * 1. Використання DOM React-ом

У React кожен фрагмент інтерфейсу користувача є компонентом зі станом. React використовує схему спостереження і слідкує за змінами стану. Коли стан компоненту змінюється, React оновлює дерево Virtual DOM. Як тільки було виконано оновлення React порівнює поточну версію Virtual DOM з попередньою версією. Такий процес називається “diffing”.

Коли React дізнається, які Virtual DOM-об’єкти змінилися – він оновлює їх у справжньому DOM. Це робить ефективність набагато кращою в порівнянні з маніпулюванням реальним DOM на пряму. Тобто розробникам не потрібно турбуватися про те, як маніпулювати атрибутами, щоб змінити відображення, бо обробка подій, або оновлення DOM відбувається за кадром. Virtual DOM існує тільки у пам’яті JavaScript.

На діаграмі \_\_\_\_ показано на високому рівні, як працює Virtual DOM, коли відбувається зміна даних.



React виконує оновлення тільки потрібних елементі, тобто Virtual DOM та справжній DOM – однакові. Наприклад якщо у нас є елемент span і розробник доповнює його через стан компонента, то виконається оновлення тільки тексту елемента, а не самого елементу. Це призводить до підвищення продуктивності порівняно з наданням цілих наборів елементів або, тим більше, цілих сторінок.

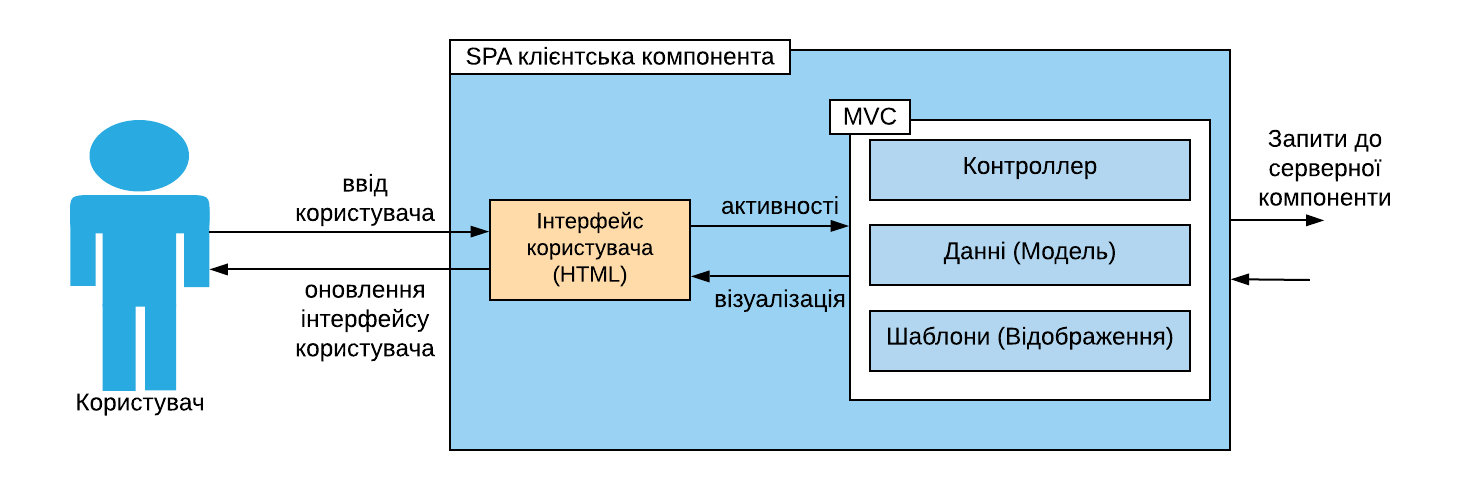
* + 1. Побудова React застосунку з MVC підходом

Концепція Model-View-Controller (MVC) дозволяє розділити данні, відображення та обробку дій користувача на окремі три компоненти:

1. Модель, що відповідає за:
   1. Зміну власного стану у відповідь на запити;
   2. Данні та методи роботи з ними;
   3. Не знає як візуалізувати власні дані;
2. Відображення – відповідає за візуалізацію інформації.
3. Контролер – контролює ввід даних користувачем і за допомогою моделі та відображення виконує потрібну реакцію.

Така структура забезпечує гнучкість за рахунок чіткого розділення. Стає простіше обслуговувати та використовувати код, створюється підтримка асинхронних реалізацій.

React не вимагає використання MVC архітектури для власної реалізації, але у даній курсовій роботі мною було використано саме такий концепт.



На діаграмі \_\_\_\_\_ контролер диктує, які потрібно використовувати шаблони (відображення) та обирає дані котрі необхідно отримати. Контролер робить запит на отримання даних, а потім заповнює шаблони цими даними для візуалізації інтерфейсу користувача у формі HTML. Усі дії користувача передаються назад у код. Сам React розміщується в блоці “Шаблони” і є рівнем відображення. Тому його можна використовувати для візуалізації HTML.

# Список литературы

Raymond, E. S. (б.д.). *The Art of Unix Programming.* Addison-Wesley Professional. Получено 7 5 2020 г., из http://www.catb.org/~esr/writings/taoup/html/