Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА (ПРОЕКТ)

на тему:

“Розробка вбудованого браузеру для проектів з відкритим кодом”

Студента 2 курсу, 5 групи,
Спеціальності
121 «Інженерія програмного забезпечення»
Спеціалізації
«Інженерія програмного забезпечення»

Таха Загер Ваелійович

_____
підпис студента

Науковий керівник
Доктор технічних наук
Професор

Криворучко Олена Володимирівна

_____
підпис керівника

Гарант освітньої програми
Доктор технічних наук
Професор

Криворучко Олена Володимирівна

_____
підпис керівника

КИЇВ – 2019

ВСТУП

Важливою складовою будь-яких наукових досліджень є інструментарій, за допомогою якого вони здійснюються. Для різних обчислень, моделювань, побудови графіків та іншого створено велику кількість програмного забезпечення, яке здатне вирішити будь-які питання. Але основною проблемою є його ціна, складність у використанні, надмірний функціонал. Тому логічним рішенням для таких проблем є розробка программного забезпечення з відкритим кодом що допоможе в розробці та використані тому що кожен хто захоче зробити внесок в розвиток продукту та зробити його краще, зможе це зробити в будь-якій точці планети та в будь-який час.

Актуальність данної тему обумовлена тим що останнім часом програми з відкритим кодом витісняють нативні додатки оскільки вони доступні з будь-якої платформи, і це дозволяє працювати без зайвих проблем, для їх функціонування потрібен код та вміння його запустити. Також вони не створюють ніякого навантаження на пристрій користувача, тому що всі вираховування проводяться на сервері а не локально.

Останнє десятиліття – це новий етап розвитку веб-технологій та Інтернету під назвою Web 2.0. Однією з його рис є поява композитних веб-додатків, які інакше називаються mashup. Такий підхід до побудови додатків став можливим завдяки сервіс-орієнтованій архітектурі програмного забезпечення. Mashup використовує інструментарії двох або більше сервісів та, використовуючи їх функціональність, створює новий. Це мало великий поштовх до розвитку різних ресурсів, зокрема соціального напрямку.

Об’єкт дослідження: веб-браузер для проектів з відкритим кодом.

Предмет дослідження: процесс створення веб-браузеру для проектів з відкритим кодом.

Метою випускної кваліфікаційної роботи (проекту) є створення веб-браузеру для проектів з відкритим кодом за допомогою JavaScript

Метод і технології розробки: В роботі використовуються сучасна мова програмування вищого рівня JavaScript з використанням сучасних фреймворків та бібліотек.

Результат роботи: створення веб-браузеру для проектів з відкритим кодом з максимально простим інтерфейсом та можливістю налаштування для вбудову в інші проекти.

Наукова новизна: В сучасному житті все більше проектів з відкритим кодом які мають потенціал розвитку з малої ідеї до чогось великого, до компанії або продукту який так чи інакше зможе зробити якийсь аспект життя людини простішим та зрозумілішим. Саме цьому проекти з відкритим кодом набувають все більшої популярності а ніж великі проекти які не можна розвивати будь-кому та будь-де. Тому вбудовані браузери також з відкритим кодом мають не аби який внесок в розвиток будь якого проекту з відкритим кодом.

Зміст

РОЗДІЛ 1
ОСНОВНІ ВІДОМОСТІ ТА ПРОБЛЕМАТИКА ПРОЕКТІВ З ВІДКРИТИМ КОДОМ

1.1 Основні відомості про відкрите программне забезпечення.

Відкрите програмне забезпечення (англ. open-source software) – програмне забезпечення з відкритим сирцевим кодом.

Сирцевий код таких програм доступний:

- для перегляду і вивчення,
- за наявності дозволу ліцензії – зміни, що дозволяє користувачеві узяти участь у доопрацюванні відкритої програми,
- використовувати код для створення нових програм – через запозичення сирцевого коду, якщо це дозволяє сумісність ліцензій
- виправляти в ній помилки,
- вивчення використаних алгоритмів, структур даних, технологій, методик та інтерфейсів (оскільки сирцевий код може істотно доповнювати документацію, а за відсутності такої сам служить документацією).

Термін відкрите програмне забезпечення (англ. open source) був створений разом з визначенням в 1998 році Еріком Реймондом і Брюсом Перенсом, які стверджували, що термін free software (вільне програмне забезпечення) в англійській мові неоднозначний і бентежить багатьох комерційних підприємців.

Переважна більшість відкритих програм є одночасно вільними. Визначення відкритого і вільного програмного забезпечення не цілком збігаються один з одним, але близькі, і більшість ліцензій відповідають обом.

Відмінність між рухами відкритого ПЗ і вільного ПЗ полягає в основному в пріоритетах. Прихильники терміна «open source» наголошують на ефективності відкритих сирців як методу розробки, модернізації та супроводу програм. Прихильники терміна «free software» вважають, що саме права людини на вільне поширення, модифікацію і вивчення

використовуваних ним програм є головною перевагою вільного відкритого ПЗ.

На думку Річарда Столлмана, розрекламованість "Open Source" дещо шкодить вільному програмному забезпеченню, бо деякі розробники і користувачі відкритого ПЗ зовсім не проти власницького програмного забезпечення, і люди зупиняються на Open Source, не доходячи до понять про свобод[1]. Він зазначає, що деякі ворожі до вільного програмного забезпечення компанії - наприклад, Microsoft –використовують тільки вираз «open source», при цьому, ймовірно, навмисно уникаючи виразу «free software».

За словами Брюса Перенса, відкрите програмне забезпечення завжди було лише способом пояснити підприємцям ідею вільного програмне забезпечення, і це йому вдалося.

Незважаючи на прагнення авторів визначення позбутися неоднозначності слова free, вираз open source теж дуже часто використовується для позначення сутностей, що суперечать визначенню Open Source Initiative або не мають до нього ніякого відношення, але здатних призвести до плутанини. Наприклад, спецслужби США використовують open source у значенні «відкрите джерело» (OSINT, Open Source Intelligence), що згадано в оголошенні на сайті Реймонда.

Існують також програми, що мають на думку деяких відкритий сирцевий код, але не є вільними, наприклад, UnRAR, розпакувальник RAR-архівів. Його сирцевий код перебуває у відкритому доступі, але ліцензія забороняє використовувати його для створення RAR-сумісних архіваторів[8]. Так само існує цілий клас програм, званих комерційним ПЗ з відкритим сирцевим кодом або Open Core, які використовують термін «Open Source» стосовно до невільного програмного забезпечення.

Одна з причин, чому код взагалі почали робити відкритими швидкий пошук помилок. Здається, усе логічно: чим більше програмістів подивляться і скористаються розробкою, тим швидше будуть знайдені і виправлені всі баги. Та на практиці так відбувається не завжди.

Згадаймо про OpenSSL, протокол шифрування, що захищає значну частину інтернету. Це проект з відкритим вихідним кодом, який понад десять років створював Стівен Генсон з невеликою командою. Наприкінці 2011 року вони відповідали за підтримку кодової бази, що містила півтора мільйона рядків коду, більшість з яких написав або затвердив сам Генсон. Колосальна відповідальність для однієї людини, якщо згадати, що OpenSSL захищав сервери двох третин усіх активних сайтів, чат-, імейл-сервери, VPN й інфраструктуру урядових та фінансових інституцій.

Кілька тижнів Стівен Генсон та його колега Робін Сегельман працювали над кодом OpenSSL, зрештою затвердили його, але пропустили одну помилку. Баг, що з часом став відомий як Heartbleed, дозволяв зловмисникам перехоплювати інформацію, що передавалась на будь-який сайт під захистом OpenSSL (а це, нагадаємо, дві третини сайтів взагалі).

Сегельман визнав, що ця помилка була доволі звичайною і помітити її було б нескладно, просто він не помітив. Баг Heartbleed був у коді OpenSSL два з половиною роки, допоки його не знайшли і не усунули в 2014-му. Та він і досі живе на тисячах пристроїв, де його навряд колись виправлять.

Тож, як бачимо, відкритий доступ до коду не убезпечує його від помилок. Те, що безліч людей можуть доопрацювати чи виправити продукт, не означає, що вони це зроблять. І на цьому прикладі стають помітними усі проблеми й суперечки, пов’язані із самим явищем відкритого коду.

Як зазначив у своєму блозі Стів Маркес, колишній генеральний директор OpenSSL Foundation, причина Heartbleed була у виснаженні розробників і відсутності фінансування. За словами Маркеса, фонд працював із бюджетом, що складався з менше ніж 2000 доларів пожертвувань й менше мільйона доларів на рік від контрактів. Фонд не міг укладати більше контрактів, тому що розробники і без того працювали на повну ставку та мали сім'ї, у них просто не було часу.

Очевидно, що OpenSSL - надто серйозний і великий проект, однієї крихітної і перевтомленої команди для нього було замало.То й же ж Стів

Маркес звинувачував комерційні установи та уряд, котрі просто використовують OpenSSL, сприймають його як належне і не вкладають у розиток проекту нічого.

Він, як і Стівен Генсон, покинув OpenSSL у 2017 році. Та спочатку ці двоє все ж подбали про майбутнє свого проекту. Основна група OpenSSL виросла до сімох розробників і проект має фінансування як мінімум до 2021 року. Гроші надає переважно Linux Foundation, спеціально створений проект, що розподіляє кошти серед open-source проектів, що мають значення для безпеки у мережі. Саму ініціативу спонсорують якраз ті ж великі гравці: Amazon, Google, IBM, Microsoft, Facebook та Intel. Допоки вони дають гроші, OpenSSL безпечний.Оскільки відкритий код використовує так багато компаній, розробники популярного ПЗ мусять розгрібати безліч проблем. Вони обробляють все більше звітів про помилки, feature-запитів, рев'ю і комітів коду тощо, кінця цьому немає.

У той же ж час програмісти, які займаються відкритим кодом, теж мусять мати справу з корпоративними користувачами. Останні не завжди розуміють норми спільноти, умови і обов’язки працівників, якщо йдеться, наприклад, про збої у роботі програми. Користувачі придбали ПЗ, яке не працює, тож байдуже, хто там винен. Зрештою і покупці, і корпоративні працівники, і розробники відкритого коду – усі незадоволені.Багато девелоперів вважають, що проблемам можна зарадити грошима. Якби гігантські технологічні компанії вкладали більше ресурсів у продукти, від яких самі ж і залежать, все було би добре. Розробники могли б більше часу працювати над відкритим кодом і це стимулювало б спільноту програмістів загалом.

Однак недостатньо просто кинути більше грошей. Збільшення фінансування створює свої проблеми: як ці гроші розподіляються і що спонсори хочуть у відповідь за них. Є ризик того, що приплив капіталу зруйнує ідейну основу спільноти, на якій відкритий код тримався майже півстоліття.

1.2 Економічна складова open-source проектів

Все почалось з лабораторії штучного інтелекту MIT на початку 80-х років. Тоді піонери комп'ютерних наук, як от Марвін Мінскі, спілкувались та навчали нове покоління хакерів, таких як Річард Столлман і Гай Стіл, котрі згодом докорінно змінять світ програмування. Стіл зіграв важливу роль в документуванні та створенні мов Lisp і Scheme. Столлман заклав фундамент усього руху open-source. Нещодавно Річард Столлман розповів New Left Review, як лабораторія штучного інтелекту сприяла культурі співпраці та радикальної відкритості. Тоді гігантський комп'ютер лабораторії ще не був захищений паролями, а двері завжди були відчинені.

У 1983 році він опублікував повідомлення в групі Usenet – такому собі протофорумі - і сказав, що створить операційну систему та віддасть безкоштовно кожному, хто зможе її використовувати. ОС він назвав GNU – це абревіатура для Gnus Not Unix. Своєрідний виклик домінуючій ОС того часу Unix.

GNU був першою ластівкою у русі вільного програмного забезпечення, принципи якого Столлман висловив у Маніфесті GNU 1985 року:

> “Золоте правило вимагає: якщо програма мені подобається, я маю вільно поділитися нею з іншими. Продавці програмного забезпечення хочуть розділяти користувачів, щоб володарювати; вони нав’язують думку, що користувачам не варто ділитися з іншими. Я відмовляюсь діяти у такий спосіб.”

Тут варто зауважати, що Столлман використовує слово «вільно» (free), але не має на увазі «безкоштовно». Вільність передбачає звільнення коду від обмежень щодо його використання, але не нульову вартість.

Основні принципи руху вільного програмного забезпечення (free software movement) були офіційно кодифіковані у 1989 році, коли Столлман опублікував GNU General Public License (GPL), тепер більш відомий як copyleft, що став основою вибуху в розробці вільного ПЗ.

Через два роки Лінус Торвальдс (Linus Torvalds), амбітний фінський студент, використав GPL для випуску Linux. Ядро Linux часто використовувалося в поєднанні з ПЗ GNU, і за три десятиліття GNU / Linux став однією з найпоширеніших операційних систем у світі. Після Linux десятки інших відомих open-source програм були випущені під GPL-compliant license, зокрема ПЗ веб-сервера Apache і механізм бази даних MySQL.

У часи розпалу доткомівської бульбашки рух Столлмана з вільним програмним забезпеченням став альтернативним баченням майбутнього. На відміну від цифрових повітряних замків, обіцяних капіталістами Кремнієвої долини, вільне програмне забезпечення було реальним і працювало. Столлман та його команда продемонстрували, що можна створити чудові програми, які задовільняють потреби користувачів і базуються на технічній майстерності та етичних переконаннях розробників.

Потім у 1997 році програміст на ім'я Ерік Реймонд опублікував есе The Cathedral and the Bazaar з аналізом процесу розробки вільних програм. В основі тексту Реймонда була ідея, яку він назвав «Законом Лінуса»: при достатній кількості очей баги випливають на поверхню. Тобто будь-які помилки в коді будуть швидко знайдені та виправлені, якщо над цим працює достатньо людей.Так Реймонд доводив ефективність створення вільного ПЗ. Наслідком стало те, що проекти з відкритим кодом змогли розвивались швидше. Адже кожен міг думати над вдосконаленням коду, його виправленням та надсилати розробку основній команді проекту.Важко переоцінити вплив аналізу Реймонда на рух вільного програмного забезпечення. Після його публікації власники браузеру Netscape, однієї з найцінніших розробок того часу, надихнулись і зробили код проекту відкритим. Очевидно, що маніфест Реймонда привернув увагу ряду верхівки Кремнієвої долини, яка усвідомила прихований бізнес-потенціал вільного ПЗ.

В основі руху за вільне програмне забезпечення була етика, а вона, як відомо, шкодить бізнесу. Тому в 1998 році група прибічників open-source зібралася, щоб з'ясувати, як зробити вільне ПЗ привабливим для

промисловості. Розробники хотіли змінити репутацію й імідж продукту, аби його захотіли купити корпорації.

«Ретроспективно нам здавалося зрозумілим, що термін “вільне програмне забезпечення” завдав неабиякої шкоди за ці роки», – написав Реймонд в есе Open Sources: Voices from the Open Source Revolution. Він зазначив, що словосполучення free software занадто асоціювалось з комунізмом і порушенням прав інтелектуальної власності. Так виникло поняття open source.

Ця маркетингова кампанія зі зміною іміджу мала феноменальний успіх. Програмне забезпечення з відкритим вихідним кодом дотепер лежить в основі технологічних платформ та послуг, які більшість з нас використовує щодня. Візьмемо хоч Microsoft, чий колишній генеральний директор Стів Балмер якось назвав Linux та інші проекти з відкритим кодом «раком». Зараз Microsoft позиціонує себе чемпіоном і фанатом open-source. Не пасуть задніх і Google, Facebook, Amazon, IBM та навіть уряд США. Вільне (free) програмне забезпечення, якщо й згадується взагалі, зазвичай ховається під загальним терміном «вільне та відкрите програмне забезпечення» або «FOSS» (Free and Open Source Software).

Більшість компаній з Кремнієвої долини швидко прийняла відкрите програмне забезпечення. Економісти ще довго намагалися пояснити, як ці проекти, що порушили всі правила ринку, могли бути настільки успішними. Попереднє пояснення (про етику, свободу і альтруїзм) не здавалося адекватним, коли мова йшла про розвиток грандіозних та успішних проектів, як от Linux. Ще не відбувалось нічого подібного лише завдяки доброзичливості й альтруїзму.

Ця аномалія спричинила шквал досліджень на початку 2000-х років. Економісти шукали, де ж розробники мали вигоду, та прагнули пояснити розвиток відкритих проектів так, щоб не порушилась традиційна система: раціональний виробник + споживач.

У 2000 році економісти Джош Лернер і Жан Тироль (Josh Lerner & Jean Tirole) опублікували найвідоміше економічне пояснення розвитку відкритих проектів. У праці The Simple Economics of Open Source Лернер і Тірол перелічили переваги, які отримували розробники від проектів open-source. Економісти залишались економістами, з їх погляду головним мотивуючим фактором не могло бути просто бажання дати світові вільне програмне забезпечення.

Вигодами, що їх отримували розробники, вважались або гроші, або користь від готового продукту для себе (тобто гроші), або просування у кар'єрі (яка принесе гроші), або визнання з боку колег.

Праця Лернера й Тіроля стала своєрідним Євангелієм у галузі. Сьогодні це найбільш цитована стаття з економіки відкритого коду

Уся ця історія з економістами і відкритим кодом характерна для початку нульових, але екосистема FOSS дуже змінилася за останні 15–20 років. Ці зміни значною мірою зумовлені створенням Git у 2005 році. Сервіси на Git (передовсім створений через 3 роки GitHub) значно прискорили темпи розробки відкритих проектів та різко знизили бар'єр для входу нових учасників.

Це була нова проблема, що її Лернер і Тірол передбачили майже за десять років до появи GitHub. Вони ставили під сумнів, чи зможе керування open-source проектами пристосуватись до ще більшої кількості учасників.Швидке зростання кількості учасників руху часто називають підтвердженням його розвитку. Однак все більше розробників FOSS стали говорити про вигорання через підтримку репозиторіїв з відкритим вихідним кодом.

Зростав потік компаній, що залежать від відкритих проектів, і зростав потік звернень та запитів від цих компаній до розробників. І кожна вважає свої справи пріоритетними, тож перенавантаження програмістів стало серйозною проблемою.

Багато розробників FOSS почали замислювати: чи є стійкою в масштабі модель розробки програмного забезпечення, яка спирається на доброзичливість окремих волонтерів? Деякі розглядали це як культурну проблему, що зникне, якщо вчити учасників бути хорошими,терплячими і відмовлятись від грошових внесків. Інші говорили про недостатнє фінансування як джерело всіх бід. Треті заперечували, що системна проблема існує взагалі.

1.3 Висновки до розділу 1

В даному розділі розглянуто та розкрите питання проблематики безприбуткових проектів з відкритим програмним кодом в сучасному світі. Розказана часточка становлення та розвитку перших проектів з відкритим кодом. Та розглянуто та проаналізовано основні проблеми та розвиток економічного аспекту програм з відкритим кодом в сьогоденні з чого можна зробити висновок що проекти з відкритим кодом можуть змінити світ.

РОЗДІЛ 2
ПОБУДОВА ВЕБ-ДОДАТКІВ ТА ОСНОВНІ БАЗОВІ ВЕБ ТЕХНОЛОГІЇ

2.1 Принципи побудови веб додатків.

На даний момент існують і успішно застосовуються різні види технологій побудови Web-додатків. Всі такі додатки мають спільну мету – реалізацію бізнес – логіки на стороні сервера і генерацію коду для клієнта. Також у всіх цих додатків однакова архітектура взаємодії сервера і клієнта та загальний протокол взаємодії - HTTP. Загальна логіка роботи кієнт-серверних програми представлена на рисунку 1.

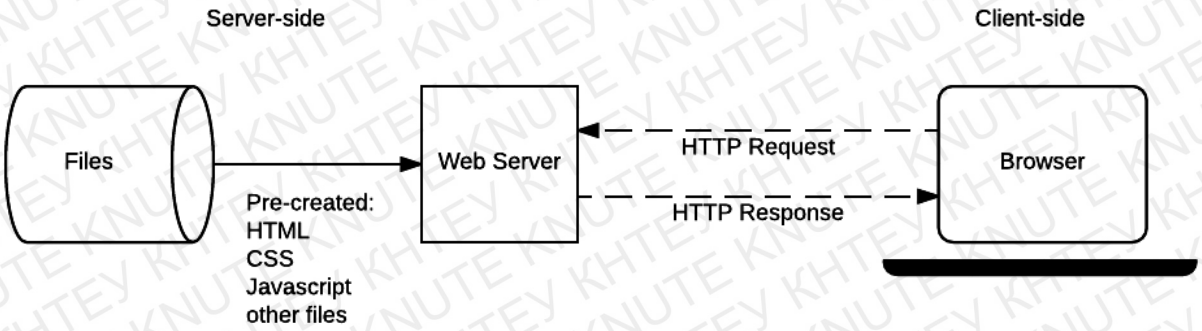

Рис. 2.1– Архітектура роботи клієнт-серверних додатків

Як видно з малюнка, робота серверних додатків відбувається в три основних етапи:

1. Запит. Клієнт, використовуючи web - браузер, ініціює запит до сервера.
2. Обробка запиту, підготовка відповіді. Після отримання запиту web - сервер проводить обробку запитуваного ресурсу. У разі, якщо запитуєтьсястатичний ресурс, такий як HTML сторінка, малюнок, документ, ця інформація форматується для протоколу HTTP і передається клієнтові в якості відповідь. Якщо ж запитується динамічний ресурс, запит передається на обробку відповідного контейнеру web - додатків, де і відбувається подальша робота.
3. Після формування, дані передаються клієнту за допомогою протоколу HTTP в якості відповіді. Відповідь містить дані (зазвичай HTML код,

абодвійкові дані), а також додаткові параметри, що передаються в заголовках HTTP відповіді.

Робота додатків серверної сторони завжди відбувається за описаним вище сценарієм. Очевидно, що такий підхід створює складності при створенні web- додатків, основною яких є відсутність стану у web - додатка (так зване stateless programming). Це означає, що додаток працює виключно в режимі запит- відповідь, не маючи даних про попередні кроки користувача або будь-якої іншої постійної інформації. Для вирішення цієї проблеми застосовується поняття користувальницької сесії, яка дозволяє зберігати дані на сервері протягом сеансу роботи користувача.

Однак , наявністю сесій , складності при створенні web - додатків повністю не усуваються. Чим більше можливостей надає платформа для додатків серверної сторони в подоланні цих труднощів, тим швидше і ефективніше може вестися розробка. Далі будуть розглянуті різні підходи до створення клієнт- серверних додатків, їхні переваги й недоліки, а також розглянуті конкретні платформи.

2.2 Вимоги до клієнт-серверних додатків.

При розгляді платформ для створення додатків необхідно виділити два основних існуючих підходу:

1. Безпосередня обробка запитів і формування відповідей.
2. Вбудовування програмного коду в шаблони HTML сторінок.

Перший підхід надає найбільші можливості по управлінню обробкою і підвищенню продуктивності. Він передбачає передачу всіх даних про запит безпосередньо виконуваного коду, який може як сформувати відповідь зі сторінкою для користувача, так і відкрити на передачу потік двійкових даних, наприклад для передачі зображення. Однак при такому підході всі дані для передачі формуються програмним шляхом, що уповільнює розробку простих сторінок і ускладнює взаємодію між верстальником і програмістом.

Прикладами цього підходу служать технології CGI, Java Servlets. Другий підхід використовує шаблони сторінок користувача, оформлені особливим чином, що дозволяє вставляти в них ділянки програмного коду.

Цей підхід особливо ефективний при створенні простих додатків, основна інформація в яких статична, а динамічна інформація може бути згенерована простими програмними конструкціями. При розробці складних програмних систем цей варіант ускладнює взаємодію між компонентами і ускладнює реалізацію складної архітектури. Також він менш ефективний за продуктивністю і призводить до обмеження можливостей реалізації складних сторінок. Прикладами цього підходу служать дуже популярні на даний момент технології PHP, ASP, JSP.

Крім різного підходу до генерації сторінок платформи розробки в різному ступені задовольняють сучасним вимогам, що висуваються при створенні складних Web систем. Найбільш важливі з цих вимог, наявність яких робить систему привабливою для використання, наведені нижче:

- Платформна незалежність.
- Мова реалізації.
- Продуктивність, масштабованість.
- Можливості розширення і інтеграції.
- Простота використання, наявність засобів розробки.
- Наявність необхідних програмних бібліотек.

Отже, ми визначили ряд вимог, необхідних для сучасної платформи розробки. Нижче розглядаються найбільш популярні на даний момент платформи, їх особливості, а також оцінка з точки зору наведених критеріїв.
Швидкий розвиток інформаційного Web - середовища призвів до того, що вимоги до Web-додатків суттєво змінилися. Зокрема спостерігається тенденція о створення багатих Web-додатків, тобто додатків, інтерфейс яких надає можливості, що не відрізняються від можливостей звичайного додатку, який призначений для настільної системі. Але при роботі програм, що

підтримують мережеву взаємодію, усунути затримку відповіді, пов’язану з передачею даних через мережу Інтернет, принципово неможливо.

Пом’якшити негативний ефект від затримки даних дозволяє технологія Ajax. Але застосування цієї технології повністю змінило структуру та принципи роботи Web-додатків. В сучасних мережевих програмах все більше функцій виконується на клієнтському боці, тому обсяг коду клієнтської частини Web-додатку суттєво збільшується і робота над нею виконується групою розробників. В результаті виявилося, що мова JavaScript, яка застосовується для написання Ajax-додатків, має специфічне застосування і не відповідає вимогам до інструментальних засобів розробки та налагодження програм.

2.1. Типи додатків веб-додатків

Розширення - (англ. extension) можуть бути використані для зміни поведінки наявних функцій або для додавання нових можливостей. Розширення особливо популярні у Firefox, оскільки розробники Mozilla створювали браузер, як досить мінімалістичну програму, що мало запобігти росту кількості помилок і запобігти громіздкості програми, зберігаючи при цьому високий степінь розширення, таким чином індивідуальні користувачі зможуть додати функції, яким вони віддають перевагу.

Розширення технологій

- CSS (Cascading Style Sheets)
- DOM (Document Object Model) - використовується для зміни XUL в реальному часі або зміни вже завантаженого HTML
- JavaScript - основна мова браузера Mozilla
- XPCOM (кросплатформлена модель компонентних об’єктів)
- XPConnect
- XPI
- XUL (XML-мова інтерфейсу користувача) - використовується для визначення інтерфейсу користувача, та взаємодії з ним.

Розширення, зазвичай, використовуються, щоб додати нові можливості до програми. Приклади можливостей, які можуть бути додані за допомогою розширень: читачі RSS, менеджери закладок, пенали, клієнтські програми для окремих веб-сайтів, менеджери протоколу FTP, електронна пошта, жести мишки, перемикання проксі-серверів, засоби веб-розробки, тощо. Багато розширень Firefox виконують функції, які раніше раніше були частиною Mozilla Suite, наприклад, ChatZilla, клієнт IRC та календар.

Зміна зовнішнього вигляду веб-сторінок для користувача Багато розширень можуть змінювати вміст веб-сторінки при її відтворенні на екрані. Наприклад, розширення Adblock може запобігти завантаженню рекламних зображень. Інше популярне розширення Greasemonkey, дозволяє користувачеві встановити скрипти, які змінюють цільові підмножини сторінок на ходу, у спосіб, що є програмним розширенням таблиць каскадних стилів.

2.2. Технологія Web 2.0

Саме поява та розвиток Web 2.0 дозволили створення динамічних веб-додатків і надали можливість створення віртуальної лабораторії функціонально-логічного програмування . Появу терміну Web 2.0 пов’язують зі статтею Тіма О’Реллі від 30 вересня 2005 року, в якій автор прив’язав появу великої кількості сайтів, об’єднаних деякими загальними принципами, із загальною тенденцією розвитку інтернет-спільноти, і назвав це явище Web 2.0, як противага «старому» Web 1.0.

Незважаючи на те, що значення цього терміну до цього часу викликає безліч суперечок, ті науковці, що визнають існування Web 2.0, виділяють декілька основних аспектів цього явища — Web-служби, Ajax, Mash-up, Теги і т.п.

Web-служби — програми, взаємодія з якими здійснюється через Web (протокол HTTP) а обмін даними відбувається в форматі XML, JSON та подібних. В результаті ПЗ може використовувати Web-служби замість самостійно реалізовувати потрібні функціональні можливості.

Ajax або Asynchronous JavaScript and XML — підхід до побудови Web-програм, при якому Web-сторіка асинхронно та без перезавантаження отримує потрібні користувачу дані з сервера. Дуже часто Ajax вважають синонімом Web 2.0, але це абсолютно не вірно — Web 2.0 не прив'язаний до будь-яких технологій і є скоріше тенденцією розвитку Інтернету.[5]

Mash-up — сервіс, що дозволяє використовувати інформацію з інших сервісів як джерело інформації, пропонуючи користувачу нові функціональні можливості для роботи. В результаті такий сервіс може стати новим джерелом інформації для інших mash-up сервісів. Виникає мережа залежних один від одного сервісів, інтегрованих один з одним.

Теги — ключові слова, що описують певний об'єкт, або відносять його до певної категорії. Це мітки, що надаються об'єкту, щоб визначити його місце серед інших об'єктів. Поява і швидке розповсюдження блогів, що активно використовують теги, також вписується концепцію Web 2.0

Багаті Web-програми — програми, що мають функціональність та можливості традиційних програм, але працюють в браузері і активно взаємодіють з сервером. Завдяки цьому створюється система, що дозволяє виконувати роботу, пов'язану з створенням та обробкою інформації більш ефективною.

Інтерфейс користувача таких програм більше нагадує інтерфейс класичних програм ніж web-програм тому ефективно використовувати такі програми можуть навіть ті користувачі, що мають мінімальні знання про Інтернет.

Технологія Web 2.0 включає в себе:

- синдикацію
- протоколи передачі даних
- браузери з плагінами та розширеннями
- клієнтське ПЗ
- Типовий Web 2.0 сайт використовує такі технології:

- Cascading Style Sheets — розділення вмісту та оформлення

2.3 Web – технології для побудови веб-додатків.

2.3.1 Базові технології Web

HTML — стандартна мова розмітки документів для Web, де всі Web-сторінки створюються за допомогою HTML (або XHTML). Мова HTML інтерпретується браузером у вигляді документу, зручному для людини.
HTML створювався в 1991-1992 роках як мова для обміну науковою та технічною документацією, яка зручна для людей, що не є спеціалістами з верстки. Вона успішно мінімізує проблеми зі складністю SGML шляхом визначення невеликої кількості структурних та семантичних елементів (які розмічаються тегами), які використовуються для створення простих, але гарно оформлених документів. Також, крім спрощення структури документу, у HTML міститься підтримка гіпертексту. Мультимедійні можливості були додані пізніше.

Текстові документи, які містять код на мові HTML, обробляються спеціальними програмами, які відображають документ у форматованому вигляді. Такі програми, що називаються браузерами, забезпечують зручний графічний інтерфейс для взаємодії користувача із сервером — запит Web-сторінок, їх відображення та відправлення введених користувачем даних на сервер.

Від початку HTML був спроектований і створений як засіб структурування та форматування документів, без їх прив’язки до засобів відображення. Але сучасні застосування HTML далекі від його початкових задач — додані мультимедійні можливості, з’явилися засоби для створення складних графічних оформлень, додана можливості підключення плагінів та розширень.

Для створення динамічних сторінок було розроблений цілий ряд технологій — JavaScript, Java Апплети, Adobe Flash, Microsoft Silverlight.

Реалізації деяких з них інтегровані в браузери (JavaScript), для роботи з іншими потрібно підключати спеціальні плагіни (доступні безкоштовно на Web-сайтах розробників або поставляються разом з операційними системами чи браузерами).

В середині 90х років розгорнулось боротьба між розробниками найбільш популярних (на той час) браузерів — Netscape Navigator та Microsoft Internet Explorer за ринок інтернет-браузерів. Основний спосіб боротьби — розробка та впровадження нових технологій, що були не сумісні з іншими браузерами. В результаті навіть на сьогоднішній день не вдалося досягти повної сумісності між усіма браузерами, хоча їх розробники та консорціум W3C, який займається стандартизацією Web-технологій, докладають максимум зусиль для цього.

З іншого боку, в результаті цієї боротьби, з'явився ряд технологій, що займають ключову роль в розвитку сучасного Web, серед них — JavaScript та Ajax. Зараз важко знайти сайт, побудований згідно принципів Web 2.0, який би не використовував Ajax або JavaScript.

2.3.2 Загальні відомості про Ajax

Ajax — група методів Web-розробки, що використовуються для створення Web-програм з багатими можливостями та мережевою взаємодією, що базується на «фоновому» обміні даними браузера з Web-сервером. В результаті сторінка не перезавантажується повністю і Web-програма стає швидкою та зручною.

Ajax це не самостійна технологія, а скоріше концепція використання декількох суміжних технологій. Ajax базується на двох основних принципах: використання технології взаємодії із сервером за допомогою JavaScript об'єкта XMLHttpRequest без перезавантаження усієї сторінки використання DHTML для динамічної зміни вмісту сторінки та реагування на дії користувача.

Для передачі даних від сервера до клієнта використовуються формати XML або JSON. Класична модель web-програм пов'язана не лише з

використанням базових web-технологій, а і з специфічним способом роботи з web-програмою, при якому web-браузер є лише низькорівневим терміналом. Він не має інформації про те, який етап роботи виконується користувачем. Він лише отримує готову сторінку в форматі HTML і відображає її користувачу.

У web-програмах, побудованих за допомогою технології Ajax, частина функціональних можливостей переноситься з сервера на клієнт. На деякі дії користувача така web-програма може реагувати самостійно. Якщо наявних можливостей не вистачає для виконання ініційованих користувачем дій то відбувається взаємодія із сервером, при цьому користувач може виконувати інші дії. Оскільки HTML документ присутній на стороні клієнта протягом всього часу роботи з web-програмою, то він здатний зберігати всю інформацію про її стан.

Технологія динамічного завантаження вмісту існувала і раніше — за допомогою атрибуту src можна було завантажити зовнішній сценарій JavaScript, який змінить поточну сторінку. Але цей метод не є дуже вдалим через обмеження атрибуту src та додатковому навантаженні на сервер, бо він має виконати додаткові дії для генерації спеціального сценарію JavaScript, що містить інструкцію, як модифікувати поточну сторінку в нову.

Засоби, що використовуються в рамках технології Ajax не єдиний спосіб забезпечити асинхронний обмін даними з сервером. Наприклад Macromedia Flash (починаючи з 4ї версії) може завантажувати дані в форматі XML або CSV з серверу без перезавантаження сторінки. Але цю технологію не можна використовувати для створення багатих web-програм бо вона в основному використовується для роботи з мультимедійними даними і малопридатна для динамічної зміни вмісту сторінки.

Пізніше Microsoft створила об’єкт XMLHttpRequest в Internet Explorer 5, що і став основою Ajax.

Переваги Ajax:

- Створення web-програм, що мають інтерфейс та багаті можливості, подібні до звичайних програм — при цьому, завдяки активній взаємодії з сервером, web-програм мають значні переваги над звичайними програмами.
- Економія трафіку — замість завантаження усієї сторінки достатньо завантажити відносно невелику частину, що змінилася.
- Зменшення навантаження на сервер — серверу не потрібно кожного разу генерувати усю сторінки, а лише ту частину, що змінилася.
- Прискорення реакції інтерфейсу — оскільки завантажується лише частина сторінки то користувач бачить результат своїх дій швидше.

Недоліки Ajax:

- Відсутня інтеграція із стандартними інструментами браузера — не працює кнопка «Назад», сторінку, згенеровану за допомогою Ajax не можна додати в закладки.
- Проблема з індексуванням сайту пошуковими роботами — у них відсутня підтримка JavaScript.
- Використання JavaScript та DOM, що мають різну реалізацію в різних браузерах та навіть різних версіях браузерів.

2.3.3 HTML та XHTML

Використано для отримання веб-інтерфейсу що виводить аналізовані данні. HTML — Мова розмітки гіпертекстових документів —стандартна мова розмітки веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML). Документ HTML оброблюється браузером та відтворюється на екрані у звичному для людини вигляді.

HTML є похідною мовою від SGML, успадкувавши від неї визначення типу документу та ідеологію структурної розмітки тексту. Попри те, що HTML — штучна комп'ютерна мова, вона не є мовою програмування. HTML разом із каскадними таблицями стилів та вбудованими скриптами — це три основні технології побудови веб-сторінок.

HTML впроваджує засоби для:

- створення структурованого документу шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації із Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об'єктів до тексту.

XHTML (Extensible Hypertext Markup Language) — мова розмітки, що має таку саму виразну силу як і HTML але відповідає синтаксичним правилам XML. В той час як HTML побудовано на основі правил SGML, XHTML побудовано на основі правил XML, суворішої підмножини правил SGML. Оскільки XHTML документи мають бути коректними XML документами, їх обробку можна здійснювати стандартними інструментами обробки XML документів на відміну від HTML, який вимагає порівняно складніших, важчих і повільніших синтаксичних аналізаторів. XHTML можна розглядати як, багато в чому, перетин HTML і XML, оскільки цей стандарт є переформуванням HTML засобами XML. XHTML 1.0 став рекомендацією консорціуму W3C 26 січня 2000. XHTML 1.1 став рекомендацією W3C 31 травня 2001.[6]

XHTML 1.0 є «реформуванням трьох типів документів стандарту HTML 4 засобами XML 1.0».[1] World Wide Web Consortium (W3C) також продовжує підтримку Рекомендації HTML 4.01 та активну роботу над специфікаціями стандартів HTML5 і XHTML5. В поточному документі Рекомендацій XHTML 1.0, який було опубліковано та переглянуто до серпня 2002 року, W3C зазначив, що, "Сімейство XHTML є наступним кроком в еволюції Інтернету. Шляхом переходу на XHTML сьогодні, розробники контенту можуть увійти в світ XML з усіма супутніми перевагами, залишаючись впевненими в зворотній та майбутній сумісності їхнього контенту.

Проте в 2004 році, незалежно від W3C було створено Робочу групу з технологій застосування гіпертексту у Вебі (WHATWG), для роботи по вдосконаленню звичайного HTML не заснованого на XHTML. Більшість великих виробників браузерів не бажали реалізовувати функції з нових проектів стандартів W3C XHTML оскільки вважали, що вони не відповідають сучасним потребам розвитку Інтернету, а W3C захопився формалізмом XML і не реагує на реальні вимоги виробників. Apple, Mozilla та Opera сформували робочу групу WHATWG, яка почала працювати над стандартом HTML5, який допускав, але не вимагав застосування XML. У 2007 році, Робоча група W3C HTML проголосувала за офіційне визнання HTML5 і роботу над ним як наступне покоління стандарту HTML. У 2009 році консорціум W3C дозволив добігти до кінця терміну дії Статуту Робочої групи XHTML 2, визнавши, що HTML 5 буде єдиним наступним поколінням стандарту HTML, як з XML, так і не-XML серіалізацію.

XHTML був розроблений з метою зробити HTML більш розширюваним і підвищити сумісність з іншими форматами даних. HTML 4 побудований на основі та є застосуванням стандартної узагальненої мови розмітки (SGML), однак специфікація SGML складна, і як веб-браузери, так і Рекомендація HTML 4 не були повністю сумісними з нею. Стандарт XML, затверджений в 1998 році, пропонував простіший формат даних, ближче за духом до HTML 4. Існували сподівання, що за допомогою переходу на формат XML, HTML стане сумісним із загальними інструментами XML; а проксі сервери зможуть перетворювати документи, у разі необхідності, для пристроїв з обмеженими можливостями, таких як мобільні телефони. Завдяки використанню просторів імен, XHTML документи могли б включати фрагменти інших, основаних на XML, мов, таких як Scalable Vector Graphics і MathML. Нарешті, відновлення роботи дала б можливість розділити HTML на компоненти для повторного використання (XHTML Модулі) і очистити неохайні частини мови.

2.4. Висновки до розділу 2

В даному розділі розділі розглянуті основні принципи побудови веб-додатків також було розглянуто основні вимоги до клієнт-серверних додатків та їх типи для більш чіткого розуміння для чого можливо використовувати розроблений веб браузер. Також проаналізовано технологію Web 2.0. та основні технології роботи вебу як такого. Розглянуто метод Ajax та HTML та XHTML як основи роботы вебу.

РОЗДІЛ 3
РОЗРОБКА ВЕБ- БРАУЗЕРУ ДЛЯ ПРОКТІВ З ВІДКРИТИМ КОДОМ WEXOND ІЗ ЗАСТОСУВАННЯМ МОВИ JAVASCRIPT

3.1 Області застосування javascript

За наявності складної задачі з управління елементами HTML в Web по допомогу до JavaScript має сенс звертатися в таких випадках:

- Перевірка коректності даних, введених у форму.
- Не використовуючі сервер CGI-програми. Тут вважається ситуація, коли потрібно або використовувати JavaScript для створення додатка, або застосовувати CGI-програми, що запускаються на сервері.
- Інтерактивна робота в динамічному HTML. Якщо елементи на сторінці мають жорстку прив'язку, то використовувати засоби DHTML і створювати сценарії для їх управління не варто.
- Створення прототипів CGI-сценаріїв. Іноді необхідно, щоб CGI-програма була упроваджена в додаток, оскільки це потенційно усуває проблеми несумісності між типами і версіями брузерів. А ще простіше створити за допомогою JavaScript прототип CGI-програми.
- Розвантаження серверу. Якщо вами використовується дуже завантажений Web-вузол, то корисно відмовитися від частого звернення до CGI- програми на користь застосування сценаріїв JavaScript, що виконують ті ж дії.
- Надання динамізму «нерухомим» сторінкам.
- Створення «інтелектуальних» Web-сторінок.

Перш ніж приступити до створення серйозних сценаріїв, необхідно скласти правильне уявлення про ті об'єкти, безпосередньо для яких і писатимуться сценарії. Об'єктами подаються елементи управління форм (текстові вікна і кнопки), а також (у останніх версіях броузерів) зображення. Проте, в моделі подано й інші об'єкти, які не такі наочні з огляду зовнішнього вигляду сторінки

і програмної його основи. Їх призначення, проте, стає цілком зрозумілим, якщо розглядати дескриптори, використовувані в HTML елементи, призначені для генерації вмісту сторінки, – прикладом може бути багатофреймова сторінка.

Для того, щоб дозволити сценарію управляти всіма цими об'єктами, а також допомогти розробникам сторінок якось упорядкувати величезну кількість об'єктів на сторінках, творці броузерів створили об'єктну модель документа (document object model або DOM). Ця модель є чимось на зразок прототипу або структури організації об'єктів на сторінці.Включені в броузери об'єктні моделі значно вдосконалені в останніх версіях броузерів.

Властивості об'єктної моделі, які доступні в одній версії броузера або броузерах одного виробника, дуже добре використовувати в тому випадку, якщо відомо, що аудиторія, на яку розрахований даний програмний продукт, використовує виключно цей тип або версії броузерів (наприклад, в корпоративній мережі). Зусилля всіляких організацій з розробки стандартів призвели до створення специфікацій для синтаксису і набору властивостей об'єктних моделей, що забезпечило, в порівнянні з оригінальними розробками, велику гнучкість. Найбільшим удосконаленням є те, що елемент HTML став об'єктом, який сценарії можуть використовувати повністю на свій розсуд (ця можливість подана ще в об'єктній моделі IE 4). Концепція DOM, побудована на основі стандартної об'єктної моделі, реалізована, з різним ступенем підтримки в броузерах IE 5+ і NN 6+ (в останньому випадку W3C DOM витриманий набагато сильніше). Якби на ринку домінували виключно броузери, які підтримують стандарт W3C DOM, то це дозволило б набагато простіше, ніж зараз створювати міжброузерне рішення і високодинамічні документи.

3.2 Ієрархічна структура моделі об'єктів

В ієрархічній моделі об'єкти згруповані відповідно до рівнів. Об'єкти мають свій рівень ієрархії. В рамках будь-якої ієрархічної структури чітко

визначені ролі кожної структурної одиниці і взаємовідношення цих структурних одиниць між собою.

Розглянемо ключові об'єкти і їх взаємозв'язки з іншими об'єктами.

Об'єкт вікна. Вверху ієрархічної структури знаходиться вікно (window). Цей об'єкт являє ту частину вікна браузера, в якій відображується вміст HTML-документа. У багатофреймовому середовищі кожен фрейм (або кадр) також є вікном. Оскільки всі події, що відносяться до документа, відбуваються саме у вікні, то воно – це найзагальніший елемент в ієрархічній структурі об'єктів. У ньому, в буквальному розумінні, розміщений документ.

Об'єкт документа. Кожен HTML-документ, завантажуваний у вікно броузера, стає об'єктом document (документ). В ієрархічній структурі положення об'єкта документа є дуже важливим. В об'єкті document міститься більшість решти типів об'єктів моделі, тобто в документі знаходиться все, що використовується в сценарії.

Об'єкт форми. Користувач не бачить на сторінці ні початку, ні закінчення форми, тільки її елементи. Але форма є особливим способом організації вмісту HTML-документа. Все, що знаходиться між дескрипторами <form> … </form>, є частиною об'єкта форми. У документі, якщо це продиктовано розумними міркуваннями, може використовуватися більше однієї пари дескрипторів форми <form>. Якщо це так, то структура об'єктів даного конкретного документа міститиме два або більше об'єктів форми замість одного.

Елементи управління форми. Точно так, як і елементи форми в HTML визначаються усередині пари дескрипторів <form> … </form>, також в об'єктах форми визначаються елементи цих об'єктів. Кожний з таких елементів форми – текстові поля, кнопки, перемикачі, прапорці і списки – це окремі об'єкти. На відміну від узагальненої моделі, кінцева модель для різних документів залежить від використовуваних в цьому документі дескрипторів HTML.

3.3. Іменування об'єктів, та точковий синтаксис в структурі програми.

Якнайкращий спосіб створення в сценаріях посилань на об'єкти полягає в тому, щоб привласнити кожному керованому в документі HTML-об'єкту, в сценарії власна назва. Підтримуючі сценарії броузери, такі, як останні версії Navigator і Internet Explorer, застосовують необов'язкові атрибути дескрипторів з назвою Name. Цей атрибут дозволяє привласнювати кожному об'єкту власне ім'я.

<form name="имя формы"> … </form>

При привласненні назв (або імен, а ще їх іноді називають ідентифікаторами), дотримуються таких правил:

- назви не можуть містити пропуски;
- в назвах не мають використовуватися символи пунктуації, за виключенням символів підкреслення;
- при їх привласненні як значення атрибуту Name вони повинні братися в лапки;
- назви не повинні починатися з цифри.

Специфікація HTML 4.0 має на увазі новий спосіб привласнення ідентифікаторів елементам HTML. Для цього використовується атрибут ID. Даний атрибут ID дуже корисний в деяких ситуаціях під час роботи з каскадними таблицями стилів (Cascading Style Sheets - CSS) і динамічним HTML. Але навіть при цьому атрибут Name є незамінним при використовуванні основних елементів, наприклад, елементи Frame, Form і Input. Останні версії броузерів можуть отримувати доступ до об'єктів по назві або ідентифікатору ID. Але розробники вважають за краще в основному для об'єктів елементів HTML використовувати атрибут ID.

У JavaScript крапка використовується для розділення елементів ієрархічного посилання. Ця угода була взята з мови Java, записи якого, у свою чергу, засновані на тих, що використовуються в C. Кожне посилання звичайно

починається з найглобальнішого рівня – для програмних продуктів клієнта, розроблених в JavaScript, це вікно, – і далі з використанням крапки (.) як роздільник виконується все велика конкретизація посилання.

Кожен об'єкт неповторюваний деякою мірою, навіть якщо в броузері такі об'єкти дуже схожі. Є три основні чинники, які безпосередньо визначають об'єкт. Вони дають уявлення про те, чим є даний об'єкт, як він виглядає і як за допомогою сценаріїв ним можна управляти. Цими трьома визначниками є властивості, методи і обробники подій. Вони, з огляду роботи в JavaScript, володіють величезною важливістю.

Властивість об'єкта – це характеристика об'єкта. Завдяки властивостям ми можемо ідентифікувати об'єкт. Кожній властивості відповідає певне значення (навіть якщо це значення не визначено, то властивість рівна null). Під час написання коду HTML для використовування в підтримуючих сценарії в броузерах властивості об'єктів встановлюються навіть без явного використовування коду JavaScript – хоча це і не так важливо. Найзагальніший спосіб початкової установки властивостей об'єктів HTML полягає у використовуванні атрибутів дескрипторів. Підключення JavaScript дозволяє використовувати додаткові атрибути, початкові значення яких можна задати під час завантаження документа. Наприклад, наступний дескриптор HTML використовується для того, щоб визначити об'єкт кнопки button. Тут же привласнюються значення для двох властивостей:

<input type="button" name="clicker" value="Hit Me…"></input>

Кнопка насправді володіє великою кількістю властивостей в даному прикладі, проте вказувати їх зовсім не обов'язково, також як і наведені. Для більшості властивостей передбачено використовувані за умовчанням значення.

Значення деяких властивостей може бути змінене в процесі завантаження документа і взаємодії користувача із сторінкою. Розглянемо такий приклад з використанням дескриптора опису текстового поля:

<input type="text" name="enrty" value=""User Name?></input>

Властивість Name даного об'єкта має значення entry. Коли відбувається завантаження сторінки, текстове значення атрибуту Value буде виведене в текстовому полі. Це приклад поведінки текстового поля в HTML з використанням атрибуту Value. Але, якщо користувач введе в текстове поле щось інше, то значення властивості Value зміниться – проте не в HTML, а в підтримуваній броузером копії об'єктної моделі, що зберігається в пам'яті. Тому, якщо сценарій звертається до текстового поля з метою отримання значення властивості Value, броузер видасть поточне значення властивості. У випадку, якщо користувачем вносилися зміни в текст, то це буде далеко не те значення, що вказане в коді HTML.

Для отримання доступу до властивості об'єкта використовується той самий тип синтаксису з використанням крапок, що і в ієрархічній структурі, описаній раніше для об'єктів. Кожна властивість належить своєму об'єкту, а посилання на властивість складається з посилання на даний об'єкт плюс ще одне додаткове розширення, що вказує на потрібну властивість. Тому для описаних вище дескрипторів кнопок і текстових об'єктів посилання на різні їх властивості виглядають так:

document . ім’яФорми . ім’яОб'єктаФорми . властивістьОб'єктаФорми

Оскільки одне вікно може містити тільки один документ, то в посиланні на об'єкт або його властивість в поточному вікні window можна не вказувати.

Методи. З об'єктом можна пов'язати будь-яке число методів (або їх може не бути взагалі). Щоб задіювати метод (викликати метод), в операторі JavaScript потрібно зробити на нього посилання – через відповідний об'єкт з

використовуванням після назви методу пари дужок, – як це показано у прикладі:

```
document . orderForm . submit ()
```

Сценарій використаний для того, щоб шляхом клацання на кнопці Submit (Відправити) відправити форму (яка називається orderForm) на сервер.

У деяких випадках, щоб метод міг виконати потрібне завдання, йому необхідно повідомити деяку додаткову інформацію. Кожна порція такого роду даних, передаваних методу, називається параметром або аргументом. Нижче наведено приклад використовування параметрів. Там задіяний метод write () об'єкту document.

```
document.write("Версія цього броузера " + navigator.appVersion)
document.write(" корпорації <b>" + navigator.appName + "</b>.")
```

Після того, як сторінка завантажується в броузер, кожен метод document.write() відправляє в поточний документ весь текст, що міститься в дужках. В обох наведених прикладах вміст буде відправлений як параметр, що складається із звичного тексту (взятого в лапки) і значення двох властивостей об'єкта: це властивості appVersion і appName об'єкта navigator. До речі об'єкт navigator не показаний в ієрархічній структурі об'єктів, оскільки в ранніх версіях броузерів цей об'єкт не включався в об'єктну модель документа.

Деяким методам потрібно вказувати більше одного параметра. В цьому випадку параметри потрібно відділяти один від одного комами.

```
window . moveTo (50,100)
```

У міру вивчення особливостей роботи в JavaScript і описуваних об'єктів документа, слід звертати особливу увагу на ті методи, які застосовані для

кожного подібного об'єкта. Це допомагає визначати, що може робити об'єкт і як ним можна управляти в сценаріях.

Обробники подій.Ще однією важливою характеристикою об'єктів у JavaScript є обробники подій. Подіями називається все, що відбувається в документі. Як правило, це результат дій користувача. Загальним прикладом дій користувача, продукуючих події, є клацання мишкою на кнопці або введення в текстове поле символу. Інші події, на зразок процесу завантаження документа у вікно броузера або поява помилки під час завантаження зображення, такими очевидними не є.

Щоб визначити, чи повинен якось реагувати об'єкт на подію, існує додатковий атрибут, який вводиться в HTML при описі об'єкта. Цей атрибут складається з назви події, знаку рівності (як і будь-який інший атрибут в HTML), після якого прямує інструкція, яка вказує, що потрібно робити при настанні конкретної події. Нижче наведено приклад дуже простого документа, в якому відображується всього одна кнопка, для якої визначений єдиний обробник події.

3.4 Розміщення сценаріїв у документах программи та описання основних операцій програми.

Щоб броузер міг взнати ті рядки коду, в документі HTML, які відповідають сценарію, цей код слід виділити за допомогою дескрипторів <script> … </script>. Такий підхід є загальним у HTML.

Для управління сценарієм в дескрипторі <script> можна використовувати різні атрибути, які залежать від типу використовуваного броузера. Один з атрибутів, підтримуваний броузерами, що управляють сценаріями, – це language. Цей атрибут надзвичайно важливий, оскільки в різних типах і версіях броузерів указуються різні параметри для використовуваної мови написання сценаріїв. Один з параметрів, який сприймають всі підтримуючі сценарії броузери – «JavaScript».

У специфікації HTML 4.0 атрибут визначення мови сценаріїв Language не підтримується. Замість нього пропонується використовувати атрибут Type.

Значення для цього атрибуту задається в стандарті MIME (Multipurpose Internet Mail Extensions). Робиться це таким чином: type="text/javascript". Атрибут Type підтримують тільки ті броузери, які відповідають стандарту W3C DOM (наприклад, IE5+ і NN6+). Разом з тим, атрибут Language все ще використовується і використовуватиметься досить довго. Тому у всіх прикладах нами використовується атрибут language.

Інші значення атрибуту Language - JavaScript № версії, JScript компанії Microsoft або VBScript.

Є ще один атрибут, який використовується в коді для того, щоб включати вміст зовнішнього файла з сценарієм у поточний документ. Атрибут Src указує на файл, що містить код сценарію. Такі файли повинні мати розширення .js. Дескриптор виглядатиме таким чином:

<script language="JavaScript" src="назваФайлу.js"></script>

На програмному рівні всі оператори, використовувані для ухвалення рішень, а також пов'язані з ними циклічні команди повторення називаються структурами управління (або управляючими структурами). Структури управління, за допомогою послідовності операторів сценарію, направляють потік обробки даних в потрібному напрямі. В основі використовуваних методів лежить схема ухвалення простих рішень й інших чинників.

Важливою частиною кожної управляючої структури є умовний оператор. Програма іноді містить умовні вирази, які можуть приймати значення True або False – одне з логічних значень булевого типу. Як правило, як подібні умовні конструкції використовуються ті, що містять в собі оператор порівняння. У програмуванні як правило порівнюються числа і рядки. Для різних програмованих ситуацій в JavaScript передбачені різні типи управляючих структур. Три найчастіше використовувані подібні структури – це умовний оператор if, умовний оператор if…else і оператор циклу for.

Існують інші управляючі структури, але деякі з них відносяться виключно до надбання Netscape і IE.

Найпростіший приклад ухвалення рішення в програмі – це ситуація, в якій залежно від справедливості певної умови в програмі вибирається той або інший алгоритм дій. Формальний синтаксис відповідної структури наведений нижче.

```
if (умова) {
оператор[и], виконуваний[е], якщо умова справедлива
}
```

Ключове слово If є обов'язковим. У круглих дужках вводиться вираз, значення якого має булевий тип. Це саме та умова, яка перевіряється при підході процесу виконання програми до даного місця. Якщо значення виразу рівне True (істина), то буде виконаний оператор або оператори, укладені у фігурні дужки, а після цього програма перейде до оператора, який знаходиться після закриваючої фігурної дужки. Якщо значення виразу рівне False (брехня), то оператори усередині фігурних дужок ігноруються і процес продовжується з першого оператора після закриваючої фігурної дужки.

У наведеному далі прикладі вважається, що змінна myAge має значення, задане їй раніше в сценарії (як саме, в даному випадку неважливо). В умовному виразі здійснюється перевірка порівняння значення змінної myAge з числом 18.

```
if (myAge < 18) {
alert ("Вибачте, ви ще молоді.")
}
```

Тип даних значення myAge має відповідати числу, щоб процес порівняння (за допомогою оператора порівняння <) вироблявся коректно. У всіх випадках, коли значення myAge менше 18, оператор у фігурних дужках буде виконаний, внаслідок чого на екрані з'явиться вікно попередження,

виконання сценарію буде продовжене з того оператора, який розташований безпосередньо після фігурної дужки, що закриває умовного оператора.

Не всі рішення в програмах ухвалюються так просто, як це показано вище на прикладі умовного оператора If. Замість того, щоб визначати конкретний маршрут виконання програми тільки у разі виконання або невиконання умови, часто зручно явно вказати два шляхи виконання програми залежно від того, виконується умова чи ні. Це не дуже помітна, але дуже важлива відмінність. У стандартному умовному операторі If у тому випадку, якщо значення контрольного виразу рівне False, ніякий спеціальний оператор не виконується. У разі ж, якщо і при негативній відповіді при перевірці умовного виразу (значення False) потрібно виконати набір певних дій, використовується конструкція if...else. Синтаксис цієї умовної конструкції наведений нижче.

if (умова) {

оператор[и] виконуваний[і], якщо умова=true

} else {

оператор[и] виконуваний[і], якщо умова=false

Тут наведено все, що вже відомо про конструкцію If. Головною відмінністю від розглянутого вище випадку є тільки наявність ключового слова Else. Воно введене для того, щоб вказати альтернативний шлях виконання програми в тому випадку, якщо значення умови, що перевіряється, рівне False.

Цикли. У реальному житті цикли означають повторення послідовності певних кроків або дій до тих пір, поки не виконається певна умова. Саме ця умова дозволяє розірвати зачароване коло повторень.

Цикл дає сценарію інструкцію періодично повторювати послідовність дій до тих пір, поки не буде виконано певну умову. Наприклад, програма JavaScript перевірки правильності даних може аналізувати кожен символ, що вводиться користувачем у текстовому полі, щоб переконатися в приналежності його до числових даних. Або, якщо у вас є набір даних,

збережених у вигляді списку, то в циклі можна перевіряти, де в цьому списку розташовується введений символ. Як тільки умова виконалася, сценарій припиняє виконання циклу і продовжує роботу з наступною після останнього оператора циклу інструкцією.

Найчастіше використовуваною в JavaScript циклічною структурою є оператор For. Він одержав свою назву за першим словом використовуваного в ньому оператора. Оператор циклу For є могутнім програмним засобом, оскільки з його допомогою можна багато раз підряд виконувати найскладніші оператори. Використовується цей оператор таким чином.

```
for (var i = 0; i < 5; i++) {
  console.log(i + ": Hello, JavaScript!");
}
```

Рис. 3.1 Оператор циклу

Квадратні дужки вказують на те, що розташовані в них параметри не обов'язкові для введення. Проте, поки у вас немає твердих навичок використовування оператора циклу For, хотілося б порекомендувати використовувати ці опції у кожному випадку. Розділ [початковий вираз] звичайно використовується для того, щоб встановити початкове значення для лічильника циклу. Під умовою вважається такий вираз, що використовувався в умовній конструкції If. У даному випадку цей вираз визначає умову, при настанні якого виконання циклу припиняється. Нарешті, [вираз оновлення] є оператором, який виконується щоразу після виконання всіх вставлених в тіло циклу операторів. Загальний принцип використовування циклу має на увазі ініціалізацію змінної і, значення якої постійно збільшується протягом виконання циклу. Так продовжується до тих пір, поки це значення не перевищить деякої максимальної величини, як це показано в прикладі:

```
if (typeof Object.prototype.clone === 'undefined') {
  Object.prototype.clone = function () {};
}
```

Рис. 3.2 Оператор If

Під startValue і maxValue в даному випадку вважається довільне числове значення, включаючи задані явно числа або змінні з числовими значеннями. У виразі оновлення використаний оператор, з яким до цього стикатися не доводилося. Оператор ++ збільшує значення і на 1 щоразу після виконання тіла циклу. Взагалі кажучи, вирази в тілі циклу можуть також використовувати значення змінної лічильника. Водночас, дуже важливо знати, як передчасно перервати виконання циклу і, особливо, в яких ситуаціях подібне переривання може знадобитися.

Функції. Під функцією розуміють набір приречених дій. Функції використовуються обробниками подій або операторами повсюдно в сценарії. Де це тільки можливо, всі хоч трохи складні набори операторів слід організовувати у функції, щоб згодом використовувати в інших документах. Функції – це ті будівельні блоки, які використовуються багато разів.

На відміну від інших мов, де використовується розділення на процедури (у яких виконуються послідовності дій) і функції (у яких виконуються дії і як результат обов'язково повертається значення), в JavaScript подібного розділення немає. Функція в JavaScript може повертати значення в зухвалого її оператора, проте ця вимога не є обов'язковою. Проте, коли під час використовування функції значення все ж таки повертається, зухвалий її оператор трактує цю функцію як звичний вираз – значення функції використовується саме в тому місці, звідки ця функція викликана.

Формальний синтаксис функцій виглядає так:

```
function* generateSequence() {
  yield 1;
  yield 2;
  return 3;
}
```

Рис. 3.3 Формальний синтаксис функції

На імена, які привласнюються функціям, накладаються ті ж обмеження, що і на назви елементів і змінних HTML. Можна порекомендувати

використовувати таку назву для функції, яке відображало б задачі, що покладаються на неї. В цьому випадку назву краще починати з дієслова, оскільки функція виконує дію, навіть якщо все, що вона робить, – це одержує або встановлює значення певної змінної.

Дуже корисно при написанні функцій задавати їй вузьку спеціалізацію. В принципі, можна згенерувати функцію завдовжки в декілька сотень рядків коду. Проте такі функції важко обслуговувати і відлагоджувати. Тому такі величезні конструкції краще розбивати на невеликі, більш сегментовані структури.

Будь-який виклик функції, включаючи і функції з інших операторів JavaScript, здійснюється за однією і тією же схемою: в кінці назви кожної функції ідуть дужки.

Можна визначити функцію так, що вона набуватиме значення своїх параметрів безпосередньо із зухвалого її оператора. Нижче показаний приклад документа, в який додана кнопка. Обробник події onClick цієї кнопки викликає функцію, передаючи їй при цьому текстові дані як параметр. Текстовий рядок, використовуваний у виклику обробником події є вкладеним рядком – це набір одинарних лапок усередині подвійних лапок, використовуваних у зовнішніх атрибутах обробника подій. Діапазон дії змінних. Змінні можуть визначатися як усередині функцій, так і поза ними. Змінні, визначені поза функціями, називаються глобальними змінними. Змінні ж, визначені в рамках функцій, називаються локальними змінними.

Глобальні змінні в JavaScript мають абсолютно інше, особливе значення, в порівнянні з більшістю інших мов. Для JavaScript межі глобальності для змінних тягнуться до розмірів поточного документа, завантаженого у вікно броузера. Тому ініціалізація змінної як глобальна вважає, що всі оператори сторінки (включаючи і ті, що розташовані в описі функції) отримують прямий доступ до значення цієї змінної. Оператори можуть відновлювати і змінювати значення глобальних змінних в будь-якому місці сторінки. Використовуючи термінологію програмування, можна сказати, що ці змінні мають глобальну

область дії, оскільки буквально всі елементи сторінки можуть їх використовувати для своїх потреб.

На відміну від глобальних змінних, локальні змінні визначаються усередині функцій. Змінні можна визначити за допомогою ключового слова Var (для локальних змінних його завжди обов'язково використовувати). Діапазон легітимності локальних змінних не виходить за рамки операторів у тілі опису функції. Жодна інша функція або оператор поза поточною не в силах отримати доступ до локальної змінної.

«Локальна область дії» визначається в тих випадках, коли для різних локальних змінних у рамках одного документа використовуються однакові назви. В більшості випадків подібна практика дуже збиткова, оскільки призводить до дуже неприємних помилок, які вкрай важко відстежувати. Водночас, в деяких випадках зручно використовувати одні і ті ж назви, скажімо, для лічильників у програмному циклі For. Це цілком безпечно, оскільки такі лічильники на початку запуску циклу щоразу наново ініціалізуються з початковим значенням. Проте, вставити один цикл For в інший такий же цикл, не використавши при цьому відмінну змінну для лічильника, не вдасться.

Основні программні файли це webpack.config.base.js, webpack.config.js, webpack.config.renderer.js, webpack.config.web.js. Основний файл в розробленому браузері який використовується в Electron це electron-builder.json.

Сценарії в цих файлах розміщені в наступному порядку та ієрархії відповідно до використання:

webpack.config.base.js це файл в якому описується використання бази:

```
/* eslint-disable */
const { resolve, join } = require('path');
const merge = require('webpack-merge');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const HardSourceWebpackPlugin = require('hard-source-webpack-plugin');
```

```
const createStyledComponentsTransformer = require('typescript-plugin-styled-components').default;
const ForkTsCheckerWebpackPlugin = require('fork-ts-checker-webpack-plugin');
const { BundleAnalyzerPlugin } = require('webpack-bundle-analyzer');
/* eslint-enable */

const INCLUDE = resolve(__dirname, 'src');

const dev = process.env.ENV === 'dev';

const styledComponentsTransformer = createStyledComponentsTransformer({
 minify: true,
 displayName: dev,
});

const config = {
 mode: dev ? 'development' : 'production',

 devtool: dev ? 'eval-source-map' : 'none',

 output: {
  path: resolve(__dirname, 'build'),
  filename: '[name].bundle.js',
  libraryTarget: 'commonjs2',
 },

 module: {
  rules: [
```

```
    {
      test: /\.(png|gif|jpg|woff2|ttf|svg)$/,
      include: INCLUDE,
      use: ['file-loader'],
    },
    {
      test: /\.tsx|ts$/,
      use: [
        {
          loader: 'ts-loader',
          options: {
            experimentalWatchApi: true,
            transpileOnly: true,
            getCustomTransformers: () => ({
              before: [styledComponentsTransformer],
            }),
          },
        },
      ],

      include: INCLUDE,
    },
    {
      test: /\.node$/,
      loader: 'awesome-node-loader',
      options: {
        name: '[contenthash].[ext]',
      },
    },
  ],
```

```
  },
```

В той час як у файлі webpack.config.js описані основні конфігурації браузеру для роботи з базою сайтів та відкривання сторінок у правильному фаорматі.

```
/* eslint-disable */
const { getConfig, dev } = require('./webpack.config.base');
const { spawn } = require('child_process');
const CopyPlugin = require('copy-webpack-plugin');
let terser = require('terser');
/* eslint-enable */

let electronProcess;

const mainConfig = getConfig({
 target: 'electron-main',

 watch: dev,

 entry: {
  main: './src/main',
 },

 plugins: [
  new CopyPlugin([
   {
    from: 'node_modules/@cliqz/adblocker-electron/dist/cjs/preload.js',
    to: 'preload.js',
    transform: (fileContent, path) => {
     return terser.minify(fileContent.toString()).code.toString();
    },
```

```
    },
    apply: compiler => {
     compiler.hooks.afterEmit.tap('AfterEmitPlugin', () => {
      if (electronProcess) {
       electronProcess.kill();
      }

      electronProcess = spawn('npm', ['start'], {
       shell: true,
       env: process.env,
       stdio: 'inherit',
      })
       .on('close', code => process.exit(code))
       .on('error', spawnError => console.error(spawnError));
     });
    },
   });
}

module.exports = [mainConfig, preloadConfig];
```

Код який описується у webpack.config.renderer.js відповідає за відпрацювання налаштувань описаних вище. Та для правильної роботи браузеру безпосередньо з сервером.

```
/* eslint-disable */
const { getConfig, applyEntries, getBaseConfig } = require('./webpack.config.base');
const { join } = require('path');
/* eslint-enable */
```

```
const PORT = 4444;

const appConfig = getConfig(getBaseConfig('app'), {
  target: 'electron-renderer',
  devServer: {
    contentBase: join(__dirname, 'build'),
    port: PORT,
    hot: true,
    inline: true,
    disableHostCheck: true,
  },
});
applyEntries('app', appConfig, [
  'app',
  'permissions',
  'auth',
  'form-fill',
  'credentials',
  'find',
  'menu',
  'search',
  'preview',
]);

module.exports = appConfig;
```

Опрацювання веб налаштуваннь оброблюється в webpack.config.web.js для прийому та передачі налаштувань по вимогам сайтів та для коректного опрацювання збережених налаштувань для окремих сайтів з локальної бази данних.

```
/* eslint-disable */
const { getConfig, applyEntries, getBaseConfig } = require('./webpack.config.base');
const { join } = require('path');
/* eslint-enable */
const PORT = 4445;
const webConfig = getConfig(getBaseConfig('web'), {
 target: 'web'
 devServer: {
  contentBase: join(__dirname, 'build'),
  port: PORT,
  hot: true,
  inline: true,
  disableHostCheck: true,
 },
 output: {
  libraryTarget: 'var',
 },
});
applyEntries('web', webConfig, ['settings', 'history', 'newtab']);
module.exports = webConfig;
```

3.5 Висновки до розділу 3

Веб-браузер з відкритим кодом який можливо використовувати як вбудований браузер у інших проектах з відкритим кодом для можливості удосконалення будь-ким для будь-кого. Було розроблено максимально корисний та інформативний інтерфейс який допоможе користувачеві працювати та використовувати усі можливості розробленого веб-браузеру максимально корисно та зручно.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В даній роботі розглянуто сучасні аспекти та розвиток проектів з відкритим кодом та їх економічна складова в сучасному світі. Проаналізувавши недоліки та переваги концепції у порівнянні з традиційним програмами було встановлено, що на сьогодні проекти з відкритим кодом мають перевагу у продуктивності, тому що одночасно багато людей може доповнювати та розвивати проект незалежно один від одного. В свою чергу проекти з відкритим кодом також мають переваги в багатоплатформності й відсутній необхідності додаткових бібліотек та додаткових ресурсів Однак у майбутньому межи між ними будуть все більш розмиватися.

Данна тема на сьогодні є актуальною тому що проектів з відкритим кодом з кожним днем все більше і вони дуже стрімко розвиваються завдяки тому що майже кожен бажаючий може прийняти активну участь у роботі та розвитку проекту чи програми.

Також встановлено що концепція проста й була придумана вже давно, проте набула популярності тільки нещодавно, оскільки з’явились інструменти, фреймворки та технології що дозволяють значно зменшити великий об’єм що необхідний для реалізації даної концепції та значно спростити процес розробки.

Розроблений інтерфейс програми є максимально інформативним та дружнім до користувача завдяки максимально простому дизайну та не важкому для розуміння дизайну.

Для розробки проекту використовувалася сучасна та популярна мова програмування вищого рівня JavaScript для найбільш кращої інтеграції у інші проекти та оточення платформ з якими може працювати будь-хто.

В результаті розроблено веб-браузер який можливо використовувати як окрема так і насамперед як вбудований безпосередньо у вже готовий проект та з максимально зрозумілою архітектурою для доопрацювання будь-ким охочим.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Криворучко О.В. Моделі і методи управління проектами: Навчальний посібник/С.Д. Бушуєв, С.В. Цюцюра, О.В. Криворучко та ін. - К.: КНУБА, 2016..
2. Olena V. Kryvoruchko, Svitlana V. Tsiutsiura, Kateryna I. Kyivska, Mikola I.Tsiutsiura, , Andij M. Dmytrychenko, FORMATION OF A GENERALIZED INFORMATION MODEL OF A CONSTRUCTION OBJECT. International Journal of Mechanical Engineering and Technology (IJMET) Scopus Indexed. Volume 10, Issue 02, February 2019, pp.
3. Відкрите програмне забезпечення – URL: https://uk.wikipedia.org/wiki/Відкрите_програмне_забезпечення
4. Табунщик Г.В. Проектування та моделювання програмного забезпечення сучасних інформаційних систем : навчальний посібник / Г.В. Табунщик, Т.І. Каплієнко, О.А. Петрова. – Запоріжжя : ЗНТУ, 2016. –
5. Jacko J.A. Human-Computer Interaction Handbook (3rd Edition). / Julie A. Jacko– CRCPress:Taylor& Francis Group 2012
6. Анопрієнко О. Я. Методика проектування Web-додатків з використанням платформи Java EE / Анопрієнко О. Я. // Інформатика і компютерні технології – 2010 – No VI – С. 160 - 165 .
7. Software as a Service: Strategic Backgrounder: [Електронний ресурс]. – Режим доступу: http://www.siia.net/estore/ssb-01.pdf. – Дата доступу:28.07.2019
8. UP2010 Virtual Cloud Conference – Microsoft’s PaaS Solution: [Електронний ресурс]. - Режим доступу: http://channel9.msdn.com/posts/UP2010-Virtual-Cloud-Conference--Microsofts-PaaS-Solution. – Дата доступу: 14.08.2019
9. Сервісно-орієнтоване програмування: [Електронний ресурс]. - Режим доступу: http://www.programsfactory.univ.kiev.ua/content/books/2/68. – Дата доступу: 14.07.2019

10. Офіційний сайт AngularJS. – Режим доступу https://angularjs.org/– Дата доступу : 27.04.2019.
11. FG-coud-technical-report: [Електронний ресурс].– Режим доступу: http://www.itu.int/en/ITU-T/focusgroups/cloud/Documents/FG-coud-technical-report.zip. – Дата доступу: 05.02.2019
12. Network Virtualisation – Opportunities and Challenges: [Електронний ресурс]. – Режим доступу: http://archive.eurescom.eu/~pub/deliverables/documents/P1900-series/P1956 /D1/P1956-D1.pdf. – Дата доступу: 8.05.2019
13. Merrill D. Mashups: The new breed of web app. / Duane Merrill / [Електронний ресурс].– Режим доступу: http://www.ibm.com/developerworks/library/x-mashups/. – Дата доступу: 23.04.2019
14. Wolfram|Alpha Webservice API Reference: [Електронний ресурс].– Режим доступу: http://products.wolframalpha.com/api/documentation.html. – Дата доступу: 24.03.2019

ДОДАТКИ

Додаток А

Лістинг файлу webpack.config.base.js

```
/* eslint-disable */
const { resolve, join } = require('path');
const merge = require('webpack-merge');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const HardSourceWebpackPlugin = require('hard-source-webpack-plugin');
const createStyledComponentsTransformer = require('typescript-plugin-styled-components').default;
const ForkTsCheckerWebpackPlugin = require('fork-ts-checker-webpack-plugin');
const { BundleAnalyzerPlugin } = require('webpack-bundle-analyzer');
/* eslint-enable */
const INCLUDE = resolve(__dirname, 'src')
const dev = process.env.ENV === 'dev';
const styledComponentsTransformer = createStyledComponentsTransformer({
  minify: true,
  displayName: dev,
});
const config = {
  mode: dev ? 'development' : 'production',
  devtool: dev ? 'eval-source-map' : 'none'
  output: {
    path: resolve(__dirname, 'build'),
    filename: '[name].bundle.js',
    libraryTarget: 'commonjs2',
  },
  module: {
    rules: [
      {
        test: /\.(png|gif|jpg|woff2|ttf|svg)$/,
        include: INCLUDE,
        use: ['file-loader'],
      },
      {
        test: /\.tsx|ts$/,
        use: [
          {
            loader: 'ts-loader',
            options: {
              experimentalWatchApi: true,
              transpileOnly: true,
              getCustomTransformers: () => ({
                before: [styledComponentsTransformer],
              }),
            },
          },
        ],
        include: INCLUDE,
      },
      {
```

```
        test: /\.node$/,
        loader: 'awesome-node-loader',
        options: {
          name: '[contenthash].[ext]',
        },
      },
    ],
  }
  node: {
    __dirname: false,
    __filename: false,
  },
  resolve: {
    modules: ['node_modules'],
    extensions: ['.js', '.jsx', '.tsx', '.ts', '.json'],
    alias: {
      '~': INCLUDE,
    },
  }
  plugins: [
    // new BundleAnalyzerPlugin()
  ],
}
function getConfig(...cfg) {
  return merge(config, ...cfg);
}

const getHtml = (scope, name) => {
  return new HtmlWebpackPlugin({
    title: 'Wexond',
    template: 'static/pages/app.html',
    filename: `${name}.html`,
    chunks: [`vendor.${scope}`, name],
  });
};
const applyEntries = (scope, config, entries) => {
  for (const entry of entries) {
    config.entry[entry] = [`./src/renderer/views/${entry}`];
    config.plugins.push(getHtml(scope, entry));

    if (dev) {
      config.entry[entry].unshift('react-hot-loader/patch');
    }
  }
};
  if (dev) {
    config.plugins.push(new ForkTsCheckerWebpackPlugin());
  }
  config.entry[`vendor.${name}`] = [
    'react',
    'react-dom',
    'mobx',
```

```
    'mobx-react-lite',
    'styled-components',
  ];
  return config;
}
module.exports = { getConfig, dev, getHtml, applyEntries, getBaseConfig };
```

Лістинг файлу webpack.config.js

```
/* eslint-disable */
const { getConfig, dev } = require('./webpack.config.base');
const { spawn } = require('child_process');
const CopyPlugin = require('copy-webpack-plugin');
let terser = require('terser');
/* eslint-enable */

let electronProcess;

const mainConfig = getConfig({
  target: 'electron-main',

  watch: dev,

  entry: {
    main: './src/main',
  },

  plugins: [
    new CopyPlugin([
      {
        from: 'node_modules/@cliqz/adblocker-electron/dist/cjs/preload.js',
        to: 'preload.js',
        transform: (fileContent, path) => {
          return terser.minify(fileContent.toString()).code.toString();
        },
      },
      {
        from: require.resolve('electron-extensions/background-preload'),
        to: 'extensions-background-preload.js',
      },
      {
        from: require.resolve('electron-extensions/content-preload'),
        to: 'extensions-content-preload.js',
      },
      {
        from: 'node_modules/mouse-hooks/mouse.js',
        to: 'mouse.js',
      },
    ]),
  ],
});

const preloadConfig = getConfig({
  target: 'electron-renderer',

  watch: dev,

  entry: {
```

```
    'view-preload': './src/preloads/view-preload',
  },

  plugins: [],
});

if (process.env.START === '1') {
  mainConfig.plugins.push({
    apply: compiler => {
      compiler.hooks.afterEmit.tap('AfterEmitPlugin', () => {
        if (electronProcess) {
          electronProcess.kill();
        }

        electronProcess = spawn('npm', ['start'], {
          shell: true,
          env: process.env,
          stdio: 'inherit',
        })
          .on('close', code => process.exit(code))
          .on('error', spawnError => console.error(spawnError));
      });
    },
  });
}

module.exports = [mainConfig, preloadConfig];
```

Лістинг файлу webpack.config.renderer.js

```
/* eslint-disable */
const { getConfig, applyEntries, getBaseConfig } = require('./webpack.config.base');
const { join } = require('path');
/* eslint-enable */

const PORT = 4444;

const appConfig = getConfig(getBaseConfig('app'), {
  target: 'electron-renderer',

  devServer: {
    contentBase: join(__dirname, 'build'),
    port: PORT,
    hot: true,
    inline: true,
    disableHostCheck: true,
  },
});

applyEntries('app', appConfig, [
  'app',
  'permissions',
  'auth',
  'form-fill',
  'credentials',
  'find',
  'menu',
  'search',
  'preview',
]);

module.exports = appConfig;
```

Лістинг файлу webpack.config.web.js

```
/* eslint-disable */
const { getConfig, applyEntries, getBaseConfig } = require('./webpack.config.base');
const { join } = require('path');
/* eslint-enable */

const PORT = 4445;

const webConfig = getConfig(getBaseConfig('web'), {
  target: 'web',

  devServer: {
    contentBase: join(__dirname, 'build'),
    port: PORT,
    hot: true,
    inline: true,
    disableHostCheck: true,
  },

  output: {
    libraryTarget: 'var',
  },
});

applyEntries('web', webConfig, ['settings', 'history', 'newtab']);

module.exports = webConfig;
```

Лістинг файлу extensions.js

```
const { existsSync, createWriteStream, unlinkSync } = require('fs');
const mkdirp = require('mkdirp');
const axios = require('axios');
const { promisify } = require('util');
const extractZip = require('extract-zip');
const { resolve } = require('path');

const extract = promisify(extractZip);

const darkreaderPath = resolve(
  __dirname,
  '../build/extensions/wexond-darkreader',
);

mkdirp(darkreaderPath, async err => {
  if (err) return console.error(err);

  if (!existsSync(resolve(darkreaderPath, 'manifest.json'))) {
    try {
      const asset = 'https://wexond.net/build.zip';

      console.log('Downloading build.zip...');
      const res2 = await axios({
        url: asset,
        method: 'GET',
        responseType: 'stream',
      });

      const zipPath = resolve(__dirname, '../build/extensions/build.zip');
      const stream = createWriteStream(zipPath);

      res2.data.pipe(stream);

      stream.on('close', async () => {
        await extract(zipPath, { dir: darkreaderPath });
        unlinkSync(zipPath);
      });
    } catch (e) {
      console.error(e);
      process.exit(1);
    }
  }
});
```

Лістинг файлу electron-builder.json

```
"appId": "org.wexond.wexond",
"productName": "Wexond",
"nsis": {
  "include": "static/installer.nsh"
},
"asar": true,
"directories": {
  "output": "dist",
  "buildResources": "static/app-icons"
},
"files": [
  "build/**/*",
  "package.json",
  "static/**/*"
],
"publish": "github",
"linux": {
  "category": "Network",
  "target": [
    {
      "target": "AppImage",
      "arch": [
        "ia32",
        "x64"
      ]
    },
    {
      "target": "deb",
      "arch": [
        "ia32",
        "x64"
      ]
    }
  ]
},
"win": {
  "target": [
    {
      "target": "nsis",
      "arch": [
        "x64",
        "ia32"
      ]
    },
    {
      "target": "zip",
      "arch": [
        "x64",
        "ia32"
```

```
      ]
     }
    ]
   },
   "mac": {
    "category": "public.app-category.navigation"
   },
   "fileAssociations": [
    {
     "name": "Document",
     "description": "Wexond",
     "role": "Viewer",
     "ext": "html"
    }
   ]
}
```