УДК 004.4
DOI https://doi.org/10.32782/2663-5941/2024.5.1/29

Мельниченко Д.В.
Державний університет «Житомирська політехніка»

Вакалюк Т.А.
Державний університет «Житомирська політехніка»

Фаррахов О.В.
Центр інформаційно-аналітичного та технічного забезпечення моніторингу об'єктів атомної енергетики Національної академії наук України

Кот Н.С.
Комунальний заклад загальної середньої освіти «Ліцей 5» Хмельницької міської ради

АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ РОБОЧИМ ЧАСОМ

У статті проведено дослідження напрямків використання інформаційних технологій для реалізації вебсистеми управління персоналом, було досліджено усі важливі аспекти початкового планування вебсистеми та участі інформаційних технологій в них. Розробка вебзастосунку для управління робочим часом з елементами системи управління персоналом дозволить автоматизувати рутинні завдання, пов'язані з плануванням та обліком робочого часу, а також з управлінням персоналом. Для виконання поставлених завдань в повній мірі потрібно створити гнучку і здатну до розширення архітектуру проекту, яка забезпечить стабільне і надійне виконання поставлених завдань. Авторами було проведено детальну роботу для постановки завдання розробки. Під час виконання цієї частини роботи було визначено тип вебсистеми, її мету, її масштаби і детальні кроки для досягнення бажаного результату. Проведено важливу роботу із визначення архітектури вебссистеми. При цьому проаналізовано різні можливі варіанти архітектури, наведено їх можливості. Прийнято рішення зупинитися на використанні вебтехнологій для досягнення найкращих результатів. Було обрано використовувати клієнт-серверну архітектуру для розподілення функцій системи. Проведено аналіз монолітної, N-шарової та мікросервісної архітектури для серверної частини вебсервісу. В результаті проведеного дослідження було прийнято рішення використовувати N-шарову архітектуру як рішення збалансоване між технічними перевагами та економією ресурсів. Також було проведено детальний аналіз можливих варіантів клієнтської архітектури, внаслідок чого прийнято рішення використовувати концепцію реактивного програмування, архітектури Flux та методологію Re-ducks для клієнтської бізнес-логіки. Було обрано та обґрунтовано цей вибір інструментальних засобів для реалізації вебсистеми із урахуванням усіх бізнес-вимог проекту та обраних архітектурних рішень.

Ключові слова: *засоби реалізації, застосунок для керування робочим часом, архітектура, N-шарова архітектура, реактивне програмування.*

Постановка проблеми. В сучасному світі наявність програмних продуктів у будь-яких сферах діяльності значно покращує умови роботи та ефективність процесів. Це твердження справедливе і для інструментів, які допомагають компаніям легше виконувати роботу із управління персоналом.

Розробка вебзастосунку для управління робочим часом з елементами системи управління персоналом дозволить автоматизувати рутинні завдання, пов'язані з плануванням та обліком робочого часу, а також з управлінням персоналом. Це включає наявність актуальної інформації про працівників із можливістю їх гнучкого оновлення, відслідковування робочого часу та відвідуваності, управління запитами на відпустки та лікарняні, а також виконання інших адміністративних процедур.

Для виконання поставлених завдань в повній мірі потрібно створити гнучку і здатну до розширення архітектуру проекту, яка забезпечить стабільне і надійне виконання поставлених завдань. Архітектура повинна дозволяти легко додавати нові функціональні можливості без суттєвих змін існуючого коду; забезпечити швидкий і надійний обмін даними між усіма частинами програми. Це включає інтеграцію між серверною і клієнтською

частинами вебзастосунку, а також можливість інтеграції з зовнішніми системами через API.

Аналіз останніх досліджень і публікацій.

Питання керування робочим часом привертає увагу все більше науковців. Зокрема, Євтушенко Г. І., Дерев'янко В. М. у своєму дослідженні аналізують стан управління робочим часом та шляхи підвищення ефективності застосування «Тайм-менеджменту» в організації [1]. Писаревська Г. І. розглядає можливості використання тайм-менеджменту для підвищення ефективності управління персоналом [2].

Крикун О. О., Медяник Ю. Г. дослідили вибір менеджером електронних інструментів для створення ефективної системи тайм-менеджменту [3]. Інша група авторів розглядала в колективній праці технології тайм-менеджменту в управлінській діяльності державних службовців [4]. Примак Т. Ю., Васильчук О. В. розглядали тайм-менеджмент як інструмент підвищення ефективності діяльності туристичного підприємства [5].

Постановка завдання. Метою статті є провести аналіз можливих засобів реалізації вебзастосунку для керування робочим часом.

Виклад основного матеріалу.

1. Визначення архітектури вебзастосунку

При плануванні проекту було вирішено розробити систему управління персоналом за допомогою вебтехнологій. Вибір обумовлений доступністю оптимального інструментарію для розробки у вигляді вебзастосунку, що спрощує використання та підтримку системи. Одночасно потрібно реалізувати можливість одночасного доступу до ресурсу для багатьох користувачів через мережу, що і є основними перевагами вебсередовища, яке надає локальний мережевий доступ та можливість спільної роботи.

Для розробки застосунку було обрано клієнт-серверну архітектуру, що є стратегічним рішенням, орієнтованим на оптимальний розподіл функцій та завдань між клієнтською та серверною частинами системи. Цей вибір дозволяє ефективно управляти обробкою даних, взаємодією з користувачем та забезпечити гнучкість у розробці та масштабуванні системи. Клієнт-серверна архітектура дозволяє розділити відповідальності між фронтендом та бекендом, сприяючи оптимізації роботи застосунку та покращенню користувацького досвіду. Такий підхід також полегшує можливість розширення та підтримки системи в подальшому.

Визначимо *архітектуру* серверної частини вебзастосунку. Серверна частина є ядром системи і буде виконувати увесь основний функціонал системи. Сервер відповідальний за усі обрахунки, прийняття рішень, надання доступу до функцій, збирання та зберігання інформації. Тому критично важливим є вибір архітектури для забезпечення масштабованості, оптимізованості, простоти розробки, надійності, безпеки і швидкості роботи системи та ще й з урахуванням доступних ресурсів.

Розглянемо основні види архітектур та їх специфіку.

Монолітна архітектура – даний вид архітектури є традиційним і стандартним для більшості програмних продуктів, тому більшість програмного коду, який можна знайти в інтернеті (в тому числі і навчальний матеріал), написаний саме у вигляді монолітної архітектури [6]. Цей тип архітектури передбачає написання програмного коду у вигляді об'єднаної автономної одиниці, яка є незалежною від інших програм. Тобто програмний продукт має представляти собою одну велику обчислювальну мережу з однією кодовою базою, яка об'єднує весь функціонал разом в одному місці.

Наведемо основні переваги даної архітектури. Оскільки один виконуваний файл або каталог дозволяє значно спростити розгортання проекту, то це власне полегшує подальшу експлуатацію. Якщо застосунок має одну кодову базу – це полегшує розробку. У централізованій базі коду та сховищі, один API може виконувати ту саму функцію, яку виконують численні API в інших архітектурних шаблонах. Окрім того, end-to-end тестування виконувати значно легше ніж у розподіленій системі. Також централізована кодова база дозволяє простіше відслідковувати запити та знаходити помилки [6]. Проте у даного виду архітектури є й свої недоліки. Зокрема, відсутність великої масштабованості проекту, складність розробки якого буде рости паралельно із розміром застосунку, призведе до неможливості працювати над цим проектом в подальшому. Якщо певна частина коду працює некоректно, це може вплинути на роботу усієї системи, що призведе до великих проблем у роботі з системою. Будь-яка зміна в окремій частині застосунку тягне за собою зміни по всьому проекту, що є великою затратою часу. Також варто відмітити, що будь-яка зміна в проекті потребує повного повторного розгортання проекту.

Враховуючи вимоги до масштабованості у гнучкості власної розробки, було прийнято рішення відмовитися від неї через значні ускладнення при збільшенні розмірів проекту. Крім

того, розробка програмного продукту, в даному випадку, може зайняти значно більше часу і при цьому не дасть відчутних переваг.

Мікросервісна архітектура – це архітектурний шаблон, який ґрунтується на серії служб, які розгортуються незалежно один від одного. Кожна служба має свою власну бізнес-логіку та базу даних з конкретною метою.

Оновлення, тестування, розгортання та масштабування відбуваються всередині кожної служби. Мікросервіси роз'єднують значні бізнес-специфічні частини на окремі, незалежні кодові бази. Мікросервіси не зменшують складність, але вони роблять будь-яку складність видимою та керованою, розділяючи завдання на менші процеси, які працюють незалежно один від одного та сприяють загальному цілому.

Запровадження мікросервісів часто йде через залучення DevOps-спеціалістів, оскільки вони є основою для безперервної доставки, які дозволяють командам швидко адаптуватися до вимог користувачів.

При цьому варто відмітити, що наявна велика гнучкість при розробці із частим розгортанням проекту, можливість робити оновлення частіше і швидше. Також в такому випадку легше налаштовувати масштабованість навантаження на окремі частини продукти; система набагато краще побудована для підтримки і усунення помилок. Окрім того, можна вибрати будь-які технології для реалізації окремих одиниць програми, а також зміни стосуються лише окремого сервісу системи.

Окрім того, варто відмітити, що система має вищу загальну складність і може мати набагато ширше коло використаних технологій, а кожен окремий сервіс має власну інфраструктуру, тому і розгортується окремо, через це також росте кількість ресурсів, що потрібна на утримання такого проекту. Також через те, що технології та кодова база сервісів можуть суттєво відрізнятися, то їх стає доволі важко стандартизувати, окрім того, якщо проект має помилку, то знайти її важче на фоні багатьох функціонуючих сервісів.

Провівши аналіз особливостей мікросервісної архітектури, було прийнято рішення відмовитись від її застосування для проекту. Основна причина відмови це знадто велика кількість ресурсів, що потрібна для організації такої архітектури. Така архітектура буде доцільнішою для проектів, що мають більшу кількість ресурсів.

N-шарова архітектура – це розділення логіки програми на певні рівні. Архітектурний шаблон N-рівня є зрілою архітектурою та просто стосується додатків, які розділяють різні логічні рівні на окремі фізичні рівні [7].

Такий тип аріхтектури часто використовують як збалансований і перевірений архітектурний шаблон, який має багато переваг і націлений на програми що швидко розвиваються. Тому цей шаблон доволі часто використовуються через його адаптивність до різних вимог програмного застосунку.

Традиційна трирівнева програма має рівень презентації, середній рівень логіки і рівень бази даних. Середній ярус необов'язковий. Більш складні програми можуть мати більше трьох рівнів.

N-рівнева архітектура також дуже часто використовуються в багатьох проектах що реалізовані на принципах ООП, адже цей тип архітектури допомагає дотримуватися принципів SOLID, що позитивно сказується на якості та стабільності програми, а також її підтримкою.

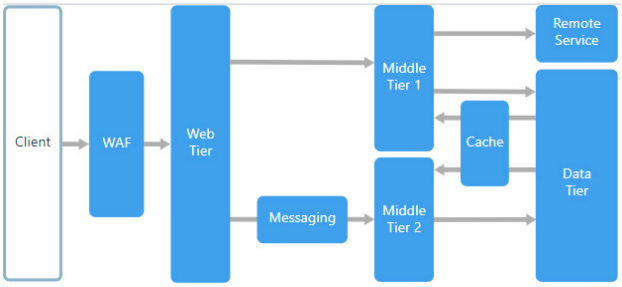

Рис. 1. Схема N-шарової архітектури [7]

При використанні даного типу архітектури помилка або збій в одному з рівнів менш ймовірно вплине на функціональність інших рівнів, а наявність кількох відокремлених компонентів в архітектурі дозволяє легко масштабувати, оновлюючи один або декілька з цих індивідуальних компонентів, окрім того, зміни, внесені в одну секцію, не впливають на інші функції. Також варто відмітити, що кожен рівень працює на власній інфраструктурі або сервері, тому ресурси не діляться.

При цьому зазначимо, що більша кількість рівнів може призвести до збільшення складності системи, а розробка та підтримка багаторівневої архітектури можуть бути дорожчими через її складність в порівнянні з монолітною архітектурою. Також є очевидним те, що чим більше рівнів, тим більше часу потрібно для обробки запитів, що може вплинути на продуктивність, при цьому управління багаторівневою архітектурою може бути складним, особливо при масштабуванні.

Через наявність одночасно масштабованості проекту, зручності розробки, якості, швидкості та надійності за збалансовану кількість ресурсів,

було прийнято рішення застосувати саме цей архітектурний шаблон.

Для серверної частини застосунку була обрана трьохшарова архітектура, яка включає шар даних, шар логіки та шар API. Це рішення дозволяє чітко розділити функціональні компоненти та відповідальності між різними рівнями системи. У шарі даних відбувається управління базою даних та зберіганням інформації, шар логіки відповідає за обробку та виконання бізнес-логіки, тоді як шар API надає інтерфейс взаємодії для клієнтської частинами. Ця архітектурна модель сприяє покращенню читабельності коду, підтримці та розширенню системи, а також спрощує тестування та вдосконалює загальну продуктивність.

2. Визначення структури клієнтської частини вебзастосунку

Для забезпечення масштабованості проекту критично важливим було впровадити зручну клієнтську архітектуру програми. Потрібно було обрати такий тип архітектури, який дозволив би зручно перевикористовувати код і його елементи, а також відокремити бізнес-логіку та графічну компоненти системи.

Реактивне програмування (RP) – це парадигма програмування, у якій основна увага приділяється потокам даних та їхній автоматичній обробці в реальному часі. RP передбачає швидку реакцію програми на зміни, а також використання асинхронних операцій [8].

Основні принципи RP – це створення реактивних функцій, управління подіями та їхнє опрацювання в реальному часі, а також використання патернів проєктування для зручного управління потоками даних.

При реактивному програмуванні легко управляти великою кількістю асинхронних подій, наявна можливість комбінувати і трансформувати потоки даних у гнучкий і декларативний спосіб, при цьому реактивне програмування полегшує роботу з побічними ефектами, такими як мережеві запити. Однак, з іншого боку, використання реактивного програмування може бути надмірним для простих задач, а крута крива навчання для розуміння концепцій і операторів реактивного програмування.

Flux архітектура дозволяє легко міркувати про програму у спосіб, який нагадує функціональне реактивне програмування, або, точніше, програмування потоку даних або програмування на основі потоку, де дані проходять через програму в одному напрямку. Стан програми підтримується лише в сховищах, що дозволяє різним частинам програми залишатися сильно відокремленими. Там, де між магазинами виникають залежності, вони зберігаються в строгій ієрархії з синхронними оновленнями, керованими диспетчером.

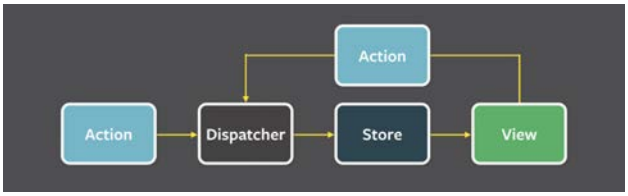

Рис. 2. Схема принципу роботи Flux архітектури [8]

Ідеї Flux архітектури також розвиває методологія Re-ducks, яка полягає у створенні файлової структури, яка є масштабованою та легкою для дотримання. Використання Redux для керування станом призводить до великої кількості дій, які використовуються в кількох компонентах, і структурі стану, яка має кілька рівнів. Як наслідок, ефективний спосіб імпортування функціональності в кілька файлів/компонентів є ключовим для уникнення помилок.

У результаті проведеного аналізу, було обрано за основу ці дві концепції і вирішено модифікувати під потреби проекту.

3. Обґрунтування вибору інструментальних засобів

Для створення архітектури застосунку, що створюється для такої критично важливої сфери як управління персоналом, потрібно обрати технології що зможуть забезпечити ефективність, стабільність та зручність впровадження нового функціоналу.

Вибір бази даних є критично важливим під час планування розробки. Від цього залежить ефективність та швидкодія проекту, його масштабованість та можливість оптимізації в подальшому. Важливо обрати базу даних, яка відповідає конкретним вимогам проекту, забезпечить надійність зберігання та швидкість доступу до інформації. Окрім того, важливо враховувати масштаби та обсяги даних, які будуть оброблятися, а також можливості інтеграції з іншими системами. Всі ці фактори сприятимуть успішній реалізації та подальшому успіху системи управління людськими ресурсами.

Було прийнято рішення використовувати реляційну базу даних SQL і СУБД SQL Server для досягнення оптимальних рішень. Реляційна база даних зможе виокремити чітку структуру і таблиці, які можна зручно використовувати та розширювати. Крім того, важливу роль у виборі зіграла високий рівень сумісності із технологі-

ями, що будуть використані при розробці серверної частини вебсервісу.

Для реалізації серверу вебзастосунку було обрано використання мови програмування C# та фреймворку ASP.NET. Це стратегічне рішення дозволило забезпечити ефективність, надійність та швидкість в розробці серверної частини системи. Мова C# володіє великою популярністю в розробці вебзастосунків, а фреймворк ASP.NET забезпечує високий рівень абстракції та простоту в роботі з вебтехнологіями.

В основі шару даних серверної трьох-шарової архітектури для взаємодії з базою даних використовується Dapper – простий та швидкий ORM (Object-Relational Mapping) для мови програмування C#. Dapper надає простий і дуже ефективний інтерфейс для виконання запитів до бази даних, що сприяє оптимальній роботі з даними та підвищує продуктивність вебзастосунку. Це рішення допомагає забезпечити ефективну та швидку взаємодію з базою даних, роблячи систему більш масштабованою та продуктивною.

Для того, щоб задовільнити додаткові вимоги проекту були використані інші додаткові пакети. Наприклад, був застосований MailKit, для створення і надсилання електронної пошти. Для вирішення проблеми розвитку схеми бази даних для кількох баз даних були задіяні міграції за допомогою FluentMigrator. Quartz.Net було використано для виконання запланованих завдань. А пакет EPPlus надає зручний інструментарій для створення XML (.xml) та EXCEL (.xlsx) файлів.

У процесі розробки серверу веб-сервісу було вирішено використовувати GraphQL як мову запитів до API для забезпечення гнучкості та ефективності. Це технологія спрощує взаємодію і зменшуючи мережевий трафік завдяки можливості отримувати лише необхідні дані. Для продуктів, що націлені на велику масштабованість, GraphQL забезпечує гнучкість відносно структури даних, підтримку кешування для оптимізації швидкості та масштабованості системи, а також гнучкий механізм керування доступом до даних, забезпечуючи безпеку та конфіденційність.

Реалізація GraphQL для серверу веб-сервісу була здійснена за допомогою бібліотеки GraphQL.Net. Цей інструментарій надає потужний фреймворк для впровадження GraphQL в .NET-середовищі, який лежить в основі шару API трьох-шарової архітектури серверу. Використання GraphQL.Net дозволяє ефективно вирішувати завдання з обробки та відповіді на GraphQL-запити, забезпечуючи зручний та ефективний механізм обміну даними між клієнтом та сервером.

Для реалізації клієнтської частини технологій було обрано мову TypeScript як розширену типізовану версію JavaScript та бібліотеку React для впровадження інтерфейсів на основі технології SPA (Single-Page Application). React є гарним прикладом бібліотеки що має в основі концепції компонентно-орієнтованої архітектури відображення графічних елементів.

Для керування внутрішнім станом клієнтського застосунку і реалізацією Flux архітектури використовувалися такі технології як Redux та Redux Toolkit для керування станом програми. Вони запроваджують єдине та централізоване для усього застосунку джерело стану в передбачуваному стані та методи його зміни.

Для реалізації концепції реактивного програмування також використовувалися бібліотеки RxJS та Redux-Observable. Використовування реактивного програмування дозволяє нам створити неперервний, контрольований та асинхронний потік даних в застосунку. Також бібліотеки надають великий функціонал для створення сторонніх ефектів як асинхронні запити на API.

Висновки. Під час дослідження напрямків використання інформаційних технологій для реалізації вебсистеми управління персоналом, було досліджено усі важливі аспекти початкового планування вебсистеми та участі інформаційних технологій в них. Було проведено детальну роботу для постановки завдання розробки. Під час виконання цієї частини роботи було визначено тип вебсистеми, її мету, її масштаби і детальні кроки для досягнення бажаного результату. Проведено критично важливу роботу із визначенням архітектури вебссистеми. Прийнято рішення зупинитися на використанні вебтехнологій для досягнення найкращих результатів. Було обрано використовувати клієнт-серверну архітектуру для розподілення функцій системи. Проведено аналіз монолітної, N-шарової та мікросервісної архітектури для серверної частини вебсервісу. Прийнято рішення використовувати N-шарову архітектуру як рішення збалансоване між технічними перевагами та економією ресурсів.

Було проведено детальний аналіз можливих варіантів клієнтської архітектури. Було прийнято рішення використовувати концепцію реактивного програмування, архітектури Flux та методологію Re-ducks для клієнтської бізнес-логіки та компонентно-орієнтовану архітектуру створення графічних елементів інтерфейсу. Було обрано та обґрунтовано цей вибір інструментальних засобів для реалізації вебсистеми із урахуванням усіх бізнес-вимог проекту та обраних архітектурних рішень.

Список літератури:

1. Євтушенко Г. І., Дерев'янко В. М. Аналіз стану управління робочим часом та шляхи підвищення ефективності застосування "Тайм-менеджменту" в організації. Збірник наукових праць Національного університету державної податкової служби України. 2014.№ 1. С. 88-96. URL: http://nbuv.gov.ua/UJRN/znpnudps_2014_1_12
2. Писаревська Г. І. Використання тайм-менеджменту для підвищення ефективності управління персоналом. Науковий вісник Херсонського державного університету. Сер. : Економічні науки. 2016. Вип. 20(1). С. 148-153. URL: http://nbuv.gov.ua/UJRN/Nvkhdu_en_2016_20%281%29__38
3. Крикун О. О., Медяник Ю. Г. Вибір менеджером електронних інструментів для створення ефективної системи тайм-менеджменту. Проблеми сучасних трансформацій. Серія: економіка та управління, 11, 2024. DOI: 10.54929/2786-5738-2024-11-04-04
4. Технології тайм-менеджменту в управлінській діяльності державних службовців: монографія / Л.Л. Приходченко, Н.В. Пироженко, М.П. Кернова, І.М Синчак ; під заг. ред Л.Л. Приходченко. Одеса: ОРІДУ НАДУ, 2021. 180 с.
5. Примак Т. Ю., Васильчук О. В. Тайм-менеджмент як інструмент підвищення ефективності діяльності туристичного підприємства. Ефективна економіка. 2019. № 12. DOI: 10.32702/2307-2105-2019.12.70
6. Microservices vs. monolithic architecture. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith.
7. N-tier architecture style. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier.
8. Реактивне програмування: що це таке та його особливості. URL: https://foxminded.ua/reaktyvne-prohramuvannia/.
9. Flux In-Depth Overview. URL: https://facebookarchive.github.io/flux/docs/in-depth-overview/.

Melnychenko D.V., Vakaliuk T.A., Farrakhov O.V., Kot N.S. ANALYSIS OF THE TOOLS FOR IMPLEMENTING A WEB APPLICATION FOR WORKING TIME MANAGEMENT

The article studies information technologies for implementing a web-based personnel management system. It examines all essential aspects of the initial planning of the web-based system and the participation of information technologies in it. Developing a web-based application for working time management with elements of a personnel management system will automate routine tasks related to the planning and accounting of working time and personnel management. In order to fulfil the tasks in total, it is necessary to create a flexible and extensible project architecture that will ensure stable and reliable performance of the tasks. The authors carried out detailed work to set the development task. During this part of the work, the type of web system, its purpose, scope and detailed steps to achieve the desired result were determined. Significant work was done to define the architecture of the web system. Various possible architecture options were analysed, and their capabilities were presented. It was decided to use web technologies to achieve the best results. A client-server architecture was chosen to distribute the system's functions. In the article was analysed monolithic, N-layer and microservice architectures for the server side of the web service. As a result of the study, it was decided to use the N-layer architecture as a solution balanced between technical advantages and resource savings. A detailed analysis of possible client architecture options was also carried out, resulting in a decision to use the concept of reactive programming, Flux architecture and the Re-ducks methodology for client business logic. The tools used to implement the web system were selected and justified, considering all the project's business requirements and the selected architectural solutions.

Key words: *implementation tools, working time management application, architecture, N-layer architecture, reactive programming.*