ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2023, № 1

УДК 004.9

# ДОСЛІДЖЕННЯ ІНТЕГРАЦІЇ МЕСЕНДЖЕРІВ ДО ВЕБСЕРВІСІВ

Михальчук Я.О. (ORCID: 0009-0006-2484-0727),
Гришанович Т.О. (ORCID: 0000-0002-3595-6964)
Глинчук Л.Я. (ORCID: 0000-0002-8943-9604)
*Волинський національний університет імені Лесі Українки*

# RESEARCH ON THE INTEGRATION OF MESSENGERS INTO WEBSERVICES

Mykhalchuk Y.O., Hryshanovych T.O., Hlynchuk L.Y.
*Lesya Ukrainka Volyn National University*

**Abstract**. The research contains information on the research of technologies for integrating messengers into web services. Methods of integrating different messengers using chat bots and combining them in one user interface are considered and compared. These methods were used during the development of the software product. The program is a complex development and consists of three main components: a database, a client part, and a server part. The following technical tools were used: TypeScript, React.js, Nest.js, MySQL, Git, GitHub.
**Keywords**: messenger, chat bot, integration, TypeScript, React.Js, Nest.js.

**Вступ**. Огляд досліджень за даною тематикою. Система обміну миттєвими повідомленнями – це програмний застосунок для обміну повідомленнями між комп'ютерами або іншими пристроями, який використовує для цього мережу Інтернет. Для того, щоб користуватися даною системою, необхідна клієнтська програма, яку називають месенджером.

Месенджер – це програмний застосунок, основною метою якого є надання можливості обміну повідомленнями між декількома користувачами в реальному часі. [24]

Месенджери стали масово використовуватися у 1990-х роках. Проте, початок розвитку концепції для спілкування певної кількості користувачів між собою за допомогою даних програмних засобів, які поєднані в уніфіковану систему, припадає на середину 1960-х років, коли була створена система обміну даними в режимі реального часу Compatible Time-Sharing System (CTSS) [3].

У 1970-х роках був створений peer-to-peer протокол. Це комп'ютерна мережа, основа якої полягає на рівноправності користувачів. В даній мережі відсутні виділені сервери, а кожен клієнт може виконувати функції сервера.

Месенджери, як універсальні настільні додатки, беруть початок із 1996 року після появи ICQ компанії Mirabilis. Саме в даному програмному застосунку було додано функціонал для передачі файлів, багатокористувацькі чати та інші опції, які не були доступні в інших месенджерах даного періоду часу. В той же період часу були розроблені такі месенджери, як Yahoo! Messenger та MSN Messenger.

Проте найбільшої популярності месенджери здобули в 2014-2015 рр, адже саме в цей період розпочалось розповсюдження 3G та 4G зв'язку та розвиток таких популярних месенджерів, як Telegram, Viber, та Facebook Messenger.

На сьогодні месенджери є одними з основних засобів спілкування в Інтернеті. Під час дослідження було розглянуто та проаналізовано найпопулярніші з них. На рис. 1. подано рейтинг найпопулярніших месенджерів у світі за 2022 рік [12].

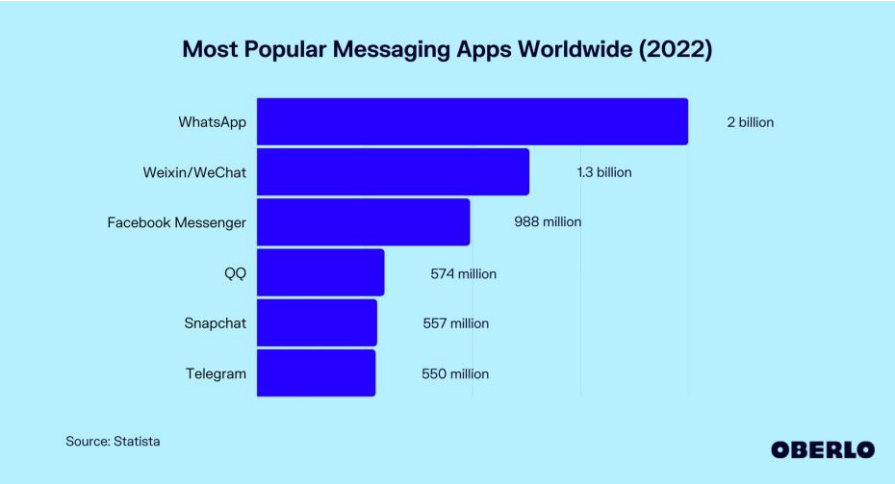


Рис. 1. Рейтинг найпопулярніших месенджерів у світі за 2022 рік [12].

Для України рейтинг месенджерів станом на березень 2023 року подано на рис. 2. [25]:

| | App | Publisher |
|---|---|---|
| 1 | WhatsApp Messenger | WhatsApp LLC |
| 2 | Telegram | Telegram FZ-LLC |
| 3 | Viber - Safe Chats And Calls | Viber Media S.à r.l. |
| 4 | Getcontact | Getverify LDA |
| 5 | Messenger | Meta Platforms, Inc. |

Рис. 2. Рейтинг месенджерів в Україні станом на березень 2023 року [25].

Суспільство активно використовує месенджери як у повсякденному житті, так і для робочих моментів. Комунікація за допомогою месенджерів в переважній більшості випадків витісняє комунікацію за допомогою телефонних дзвінків у різних сферах бізнесу, а особливо у сферах, де потрібно працювати з клієнтами. Даний вид комунікації отримує перевагу в можливості спілкуватися з декількома клієнтами одночасно, проте навіть це не завжди є оптимальним варіантом, оскільки все залежить від кількості клієнтів та кількості месенджерів або акаунтів, за якими закріплений менеджер. Одним з рішень є використання чат-ботів для комунікації та поєднання їх у одному вебінтерфейсі, який буде надавати ідентичний функціонал для комунікації з різних месенджерів. Метою роботи є дослідження методів інтеграції месенджерів у вебсервіси. Для цього слід порівняти можливості кожного із розглянутих месенджерів, які можна інтегрувати, дослідити доцільність використання чат-ботів для комунікації у месенджерах, порівняти особливості інтеграції месенджерів Telegram та Viber, які

входять до трійки напопулярніших в Україні, у вебсервіси; розглянути можливості, які надають вебсервіси для інтеграції з API месенджерів для кожного з месенджерів.

**Методологія дослідження**. Для створення чат-ботів в Telegram, необхідно використовувати BotFather, який є програмним засобом для створення інших ботів в Telegram. Під час створення чат-бота потрібно вказати його ім'я та юзернейм. Якщо валідація цих даних пройшла успішно, тоді BotFather надасть токен, який потрібен для розробки функціоналу бота.

Суть роботи бота з точки зору розробки полягає в:

- отриманні оновлень від користувачів. Оновленнями можна вважати нові повідомлення, відредаговані повідомлення та інші активності користувачів під час комунікації з ботом;
- активностей зі сторони бота. Активностями можуть бути будь-які дії, які запрограмовані для виконання їх ботом, наприклад, відправка повідомлень, відправка медіафайлів, розсилка та інше.
- Існує два способи, які дозволяють отримувати оновлення від користувачів:
- за допомогою webhook-а. Суть даного способу полягає в тому, що оновлення будуть приходити на попередньо задану кінцеву точку на серверній частині додатку для інтеграції бота;
- за допомогою тривалого опитування. Суть даного способу полягає в тому, що сервер додатку час від часу перевіряє, чи є оновлення від користувача. Якщо вони є, тоді сервер їх опрацьовує згідно того, як він запрограмований.

Перед використанням будь-якого з цих способів слід потрібно перевірити бот за допомогою методу getMe(), оскільки якщо бот є неперевіреним, тоді розробка його функціоналу є неможливою.

Для того, щоб здійснювати дії зі сторони бота, достатньо надсилати запит за параметрами на потрібне посилання, обов'язковим для запиту є наявність в URL токена бота. Telegram Bot API надає широкий спектр можливостей для ботів.

У Viber найчастіше інтегрують ботів за допомогою таких технологій, як Node.js та Python.

Для інтеграції бота у Viber потрібно здійснити наступні дії:

- потрібно мати активний акаунт в даному месенджері;
- здійснити вхід у панель адміністратора Viber за номер телефону, до якого прив'язаний акаунт;
- створити бот, заповнивши усі обов'язкові поля у формі;
- після успішного створення потрібно скопіювати унікальний токен бота, який потрібний для надсилання кожного запиту до Viber. Даний токен додається до заголовків запиту за ключем X-Viber-Auth-Token;
- встановити webhook. Для цього потрібно створити кінцеву точку на серверній стороні додатку, яка буде обробляти дані, які повертатиме webhook. URL даної точки повинен бути перевіреним та захищеним за допомогою SSL сертифікату.

Під час обміну повідомленнями існує три основних складових, які беруть участь в даній операції:

1. Public account. Даною складовою виступає бот, який інтегрований в додаток. Дана складова відповідає за надсилання повідомлень до Viber та обробку даних на стороні додатку, до якого інтегрований бот.

2. Viber Client. Це інтерфейс додатку Viber на телефоні або ПК, який відповідає за взаємодію з користувачем.

3. Viber Backend. Це сервер Viber, він виступає посередником між Viber Client і Public account. Основне завдання даної складової – обробляти запити від двох вищевказаних складових та налагоджувати комунікації між ними.

На сьогоднішній день існує багато напрямків в розробці для інтеграції чат-ботів. Найпопулярнішими з них є: мобільна розробка, розробка під персональні комп'ютери та розробка вебдодатків.

Проте, найбільш доцільним для інтеграції варто виділити напрямок розробки вебдодатків, оскільки в нього є декілька суттєвих переваг над іншими. Основними з них є: можливість ідентичного використання як на смартфонах, так і на персональних комп'ютерах через браузер; відсутність проблем з кросплатформенністю при використанні сучасних фреймворків; існування готових бібліотек, які частково полегшують інтеграцію; легкість у запуску.

У випадку інтеграції месенджерів найкраще розробляти додаток, використовуючи підхід SPA – це підхід до веб-розробки, за якого вся програма розміщена на одній сторінці [19].

**Результати дослідження та їхнє обговорення**. Розроблена реалізація інтеграції месенджерів до вебсервісу складається з трьох основних частин: клієнтської, серверної та бази даних.

Для комунікації між клієнтами та сервером було обрано клієнт-серверну архітектуру. Доцільність використання даної архітектури зумовлюється тим, що в даному програмному забезпеченні клієнтська та серверна частина є окремими складовими, і, відповідно, вони будуть взаємодіяти між собою, використовуючи мережу Інтернет. Передбачено, що клієнтів може бути досить багато і всі вони будуть звертатись до одного серверу. Серверна частина даного програмного забезпечення надає доступ до даних за підходом REST (Representational State Transfer) — це архітектурний стиль для забезпечення стандартів між комп'ютерними системами в Інтернеті, полегшуючи тим самим взаємодію між системами.

Взаємодія між клієнтом та сервером відбувається за допомогою HTTP протоколу та вебсокетів. Вебсокети використовуються для взаємодії клієнтської та серверної частин месенджера, а HTTP є основним протоколом для передачі даних між двома частинами додатку [8]. Даний протокол використовує запити для передачі даних.

HTTP-запит – це блок даних, який пересилається між HTTP-додатками. Такі блоки даних починаються з деякої текстової мета-інформації, що описує зміст і значення повідомлення, після чого йдуть додаткові дані.

Для серверу було обрано трирівневу архітектуру. Принцип роботи даної архітектури зображено на рис. 3.

1. Рівень доступу до даних (Data Access Layer) – відбувається отримання, записування та оновлення даних. Провайдером в даному рівні можуть виступати реляційні або нереляційні бази даних, технології, які зв'язують БД з концепціями мов програмування.

2. Рівень бізнес-логіки (Business Layer) - отримання даних з презентаційного рівня, обробка даних та передачі до рівня доступу до даних та навпаки. На даному рівні виконується функціонал для перевірки даних на відповідність до бізнес правил.

3. Презентаційний рівень (Presentation Layer) - отримання даних з нижчих рівні або від користувача системи (клієнта). Дані можуть бути представлені у різних виглядах, наприклад, веб-сторінка, JSON-формат та інші. У контексті даного проєкту дані представляються у JSON-форматі.

Для розробки клієнтської частини було вирішено використати MVVM (Model-View-ViewModel) — це архітектурний шаблон, заснований на MVC і MVP, який

намагається більш чітко відокремити розробку інтерфейсів користувача (UI) від бізнес-логіки та поведінки в програмі [16]. З цією метою багато реалізацій цього шаблону використовують декларативні зв'язки даних, щоб дозволити відокремити роботу над видами від інших рівнів.

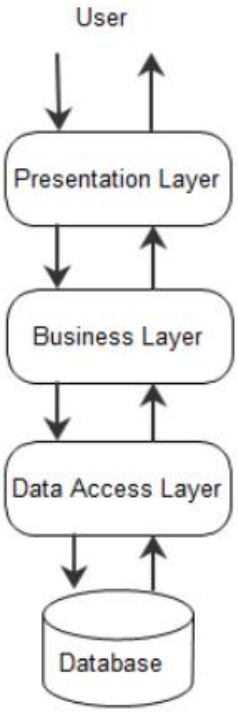


Рис. 3. Принцип роботи трирівневої архітектури

Даний шаблон складається з трьох складових [16]:

- модель (model), яка представляє специфічні для домену дані або інформацію, з якою працюватиме програма;
- вид (view), єдина частина програми, з якою користувачі фактично взаємодіють;
- модель перегляду (viewModel), яку можна вважати спеціалізованим контролером, який діє як перетворювач даних. Він змінює інформацію моделі на інформацію виду, передаючи команди від виду до моделі.

Для зберігання даних було обрано систему керування БД (СКБД) MySQL. Безпосередньо БД було запущено за допомогою технології Docker в інкапсульованому контейнері [5].

На рис. 4 представлено ключові відношень у поданій БД.

Оскільки дане програмне забезпечення є вебдодатком, потрібно було обрати мову програмування, яка підтримується браузером, або яка може бути інтерпретована в таку мову. На основі цього було вирішено обрати TypeScript, як основну мову програмування [7, с.19]. Для розробки клієнтської частини додатку було обрано React.js як основну технологію [2, с.14]. Оскільки React немає якісного вбудованого рішення для управління станом додатку у великих проєктах, було вирішено використати бібліотеку Redux[2, с.190]. Для розробки серверної частини було обрано фреймворк Nest.js, який базується на платформі Node.js [7, с.11]. Для запуску СКБД було вирішено використати Docker, оскільки даний інструмент дозволяє досить швидко розгорнути MySQL в інкапсульованому контейнері, до якого є доступ лише через заданий порт та хост розробником. Це в певному плані дозволяє захистити СКБД від впливу зовнішніх непередбачених факторів. Середовищем для розробки обрано WebStorm.

Оскільки додаток є розділеним на дві частини: клієнтська частина та серверна частина, відповідно програмний код також є розділеним.

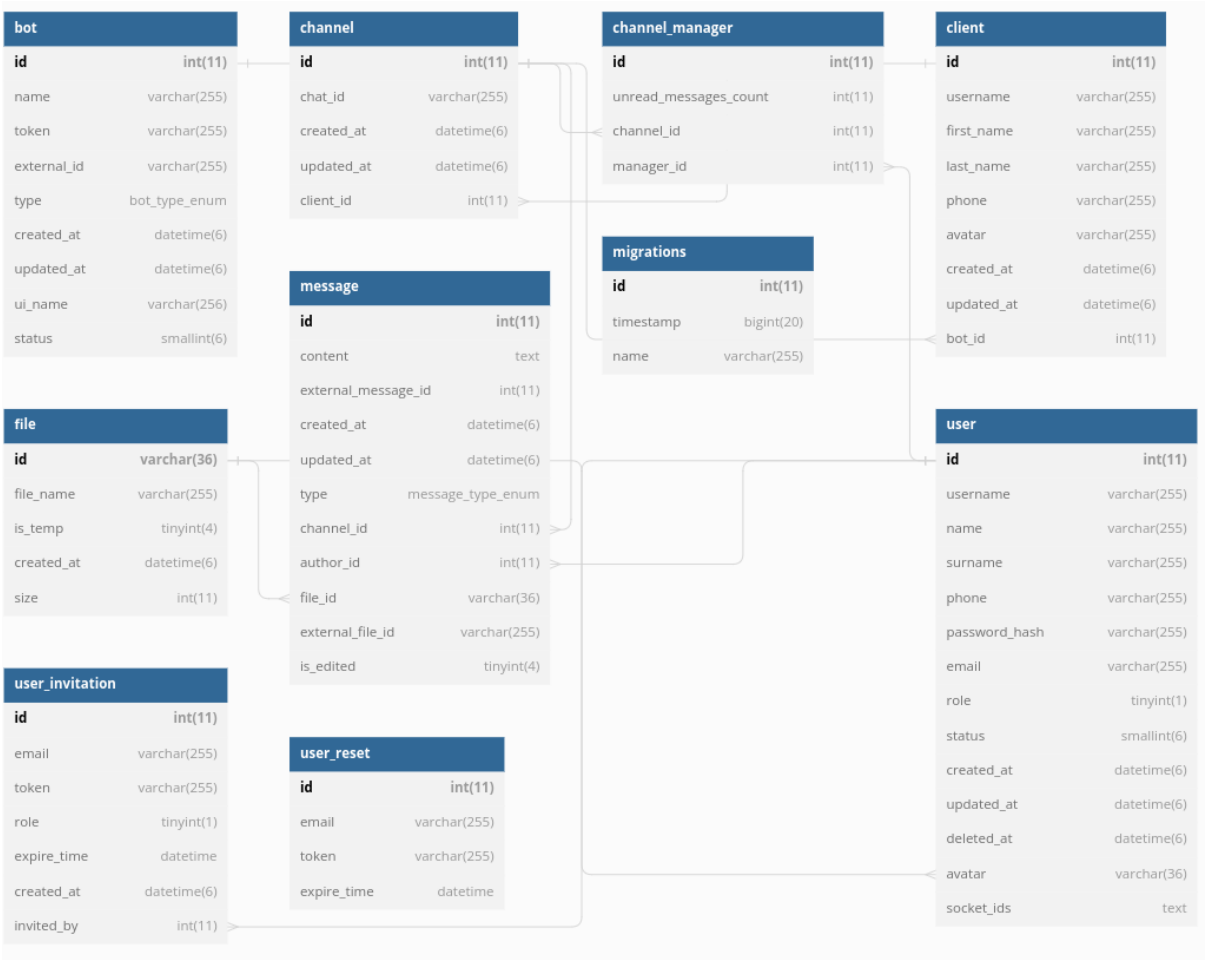


Рис. 4. Загальна схема відношень в базі даних

Клієнтська частина представлена у вигляді SPA (Single Page Application) та розроблена з використанням функціонального стилю програмування. Структура програми поділена ієрархічно, враховуючи роль кожного компонента програми. На рис. 5 наведено файлову структуру програми.

Файли з компонентами та функціями зберігаються в певній папці, в залежності від їх логічного значення для програми. Розглянемо призначення кожної папки.

Api – в даній папці знаходять файли з функціями, які відповідають за роботу з серверною частиною. Варто виділити метод, який створює сервіс для звернення до серверної частини та метод, який використовується для завантаження файлів. Сервіс створюється за допомогою бібліотеки apisauce. Під час створення сервісу потрібно вказати базовий URL на серверну частину, тайм-аут та заголовки. У нашому випадку вказуємо тип контенту для файлів та токен для авторизації. На рис. 6 наведено код даного сервісу.

Метод для завантаження файлів приймає файл як аргумент. Після цього створюється інтерфейс FormData, в котрий додається отриманий файл. Далі за допомогою сервісу надсилається POST-запит для завантаження файлу. Якщо все пройде успішно, отримуємо результат, який повернув сервер, якщо ні – отримаємо помилку від сервера, в якій буде вказана причина. На рис. 7 наведено код даного методу.

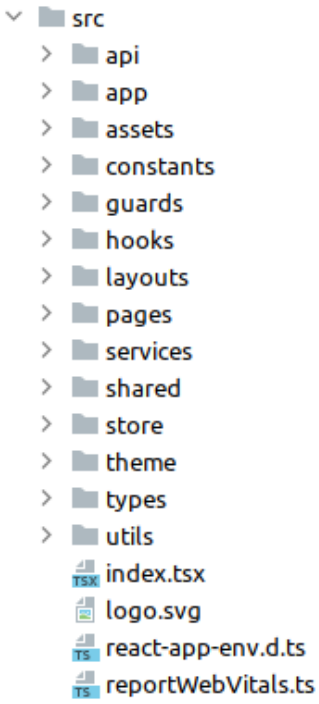


Рис. 5. Файлова структура клієнтської частини

```
import { ApiResponse, create } from 'apisauce';
import { getTokenFromStorage } from 'utils';
import { FileData } from 'types';
import { API_ROUTES } from 'constants/api.routes';
import { CommonResponse } from './types';
import { getGeneralApiProblem } from './api-problem';

const token = getTokenFromStorage();

export const fileService = create( config: {
  baseURL: process.env.REACT_APP_API_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'multipart/form-data',
    Authorization: token ? `Bearer ${token}` : '',
  },
});
```

Рис. 6. Код методу для створення сервісу, який буде звертатись до серверної частини.

App – в ній знаходяться файли з функціоналом для запуску клієнтської частини.

Assets – в даній папці знаходяться картинки, шрифти та інші медіа-файли.

Constants – містить в собі файли з константами.

Guards – містить в собі функціонал для перевірки, чи даний користувач є аутентифікованим. Варто виділити компонент AuthGuard. Суть роботи даного компонента полягає в перевірці, чи авторизований поточний користувач системи при відкритті кожної сторінки. Якщо перевірка успішна – сторінка буде відкрита, в іншому випадку користувачу буде відкрита сторінка логіну. На рис. 8 наведено код даного компонента.

```
export const uploadFile = async (file: File): Promise<FileData | undefined> => {
  try {
    const formData = new FormData();
    formData.append( name: 'file', file);
    const response: ApiResponse<FileData> = await fileService.post(
      API_ROUTES.FILES.UPLOAD,
      formData,
    );
    return response.data;
  } catch (e) {
    throw new Error('Bad data');
  }
};
```

Рис. 7. Код методу для завантаження файлу на серверну частину.

```
import React from 'react';
import { useLocation } from 'react-router-dom';
import { useSelector } from 'react-redux';
import Login from 'pages/Authentication/Login/Login';
import { selectAuthenticationState } from 'store/auth';
import { getTokenFromStorage } from 'utils';

type AuthGuardProps = {
  children: React.ReactNode | React.ReactNode[],
}

export const AuthGuard: React.FC<AuthGuardProps> = ({
  children : ReactElement<any, string | JSXElementConstructor<any>> | ... ,
}): JSX.Element => {
  const isAuthenticated = useSelector(selectAuthenticationState);
  const accessToken = getTokenFromStorage();
  const { pathname } = useLocation();
  if (!isAuthenticated && !accessToken && pathname !== null) {
    return (
      <Login />
    );
  }

  return (
    <>
      {children}
    </>
  );
};
```

Рис. 8. Код компонента AuthGuard.

Hooks – містить в собі хуки, які в свою чергу є одним з концептів розробки додатку за допомогою React.js. Можуть бути використані в будь-якому файлі з кодом в клієнтській частині.

Layouts – містить в собі файли, які є основними макетами для додатку.

Pages – в даній папці зберігаються файли, які використовуються для поділу додатку на сторінки.

Services – містить в собі сервіси, які є проміжним функціоналом між менеджером стану та методами для роботи з серверною частиною.

Shared – містить в собі невеликі презентаційні компоненти, які можуть бути використані в усьому додатку. Одним із таких компонентів є компонент Item, який використовується для генерування таблиць. Даний компонент приймає декілька аргументів: дані, конфігурацію для виведення комірок таблиці, параметри для надання дозволів та інші. Відповідно кожна окрема таблиця буде показана користувачу в залежності від прийнятої конфігурації, даних та параметрів для надання дозволів. На рис. 9 наведено код даного компоненту.

```
export const Item = <T extends { id: number}, >({
  data, onDelete, config, updatingLink, module, action,
}: ItemProps<T>): JSX.Element => {
  const { onItemClick } = useOnItemClick(updatingLink, data.id);
  return (
    <TableRow
      hover
      sx={{ cursor: 'pointer' }}
    >
      {renderCell(config).map((cell :Cell<unknown> ) => (
        <TableCell
          key={uuidv4()}
          onClick={onItemClick}
          align={data[cell.name] ? 'left' : 'center'}
        >
          {cell.render ? cell.render(data, cell) : data[cell.name] || '-'}
        </TableCell>
      ))}
      <TableCell align="right">
        <PermissionWrapper module={module} action={action}>
          <MoreMenu
            itemId={data.id}
            updatingLink={updatingLink}
            onDelete={onDelete}
          />
        </PermissionWrapper>
      </TableCell>
    </TableRow>
  );
```

Рис. 9. Код компонента Item.

Store – в даній папці зберігаються файли з функціоналом для управлінням стану додатку. Вся інформація, яка є загальною по додатку, зберігається в об'єкті, який має назву store. store створюється бібліотекою reduxjs/toolkit. На рис. 10 наведено код методу, який створює даний об'єкт.

Theme – містить в собі всі функції, які потрібні для темізації.

Types – містить в собі всі типи, які потрібні для типізації функції та компонентів додатку.

Utils – в даній папці містяться допоміжні функції, які використовуються в додатку.

```
export const setupStore = () => configureStore( options: {
  reducer: rootReducer,
  middleware: (getDefaultMiddleware :CurriedGetDefaultMiddleware<S> ) => getDefaultMiddleware().concat([
    authAPI.middleware,
    usersAPI.middleware,
    botsAPI.middleware,
    clientsAPI.middleware,
    chatMiddleware,
  ]),
});
```

Рис. 10. Код методу setupStore.

Структура серверної частини побудована з використанням принципів фреймворку NestJS, який використовувався для розробки. Суть даної структури полягає в тому, що кожна папка виступає модулем, який містить в собі контролери, сервіси, сутності та інше. На рис. 11 наведено структуру даної частини додатку.

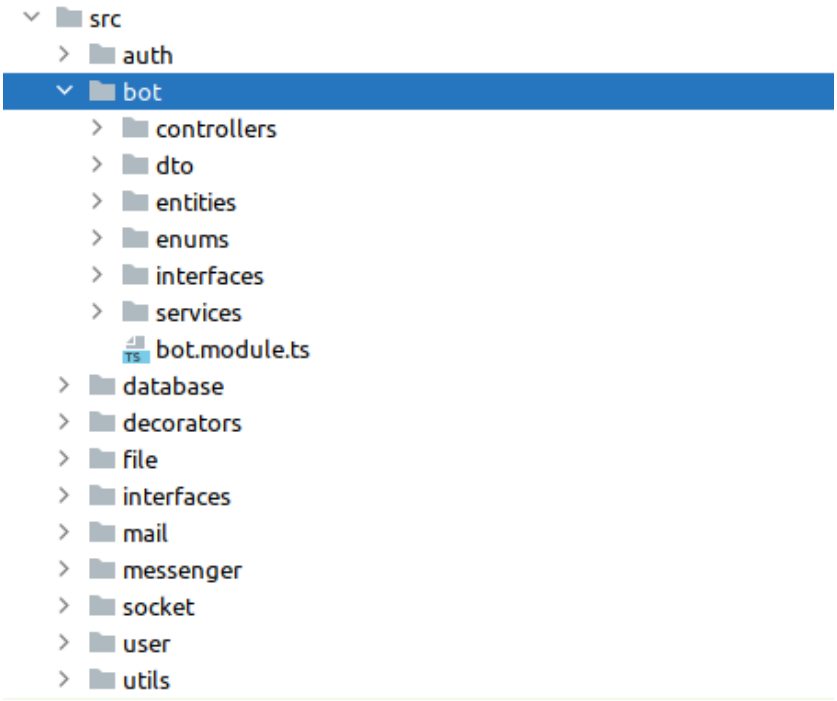


Рис. 11. Структура програмного коду серверної частини.

Основний патерн, який використовується в додатку – це Впровадження Залежностей (Dependency Injection, DI). Це шаблон програмування, у якому залежність передається за допомогою параметрів замість інстанціювання її у функцію чи клас. DI дозволяє створювати ізольовані окремі компоненти в коді програми та полегшує перемикання цих залежностей у майбутньому, коли вимоги зміняться. Передача параметрів як залежності, також дозволяє легко тестувати ці компоненти ізольовано, впроваджуючи їх змішану версію [1]. Даний патерн закладений у філософію використання фреймворку NestJs. Приклад використання даного патерну наведений на рис. 12.

```
export class BotService {
  constructor(
    @InjectRepository(Bot) private botRepository: Repository<Bot>,
    private readonly factoryService: FactoryService,
    private readonly socketService: SocketService,
    private readonly channelService: ChannelService,
    private readonly messageService: MessageService,
  ) {}
```

Рис. 12. Приклад патерну DI у серверній частині додатку.

Основним модулем додатку є модуль, який пов'язаний з інтеграцією чат-ботів. Робота з ботами організована за допомогою патерну – Фабричний метод. Цей патерн визначає інтерфейс створення об'єкта, але дозволяє підкласам обрати клас створюваного екземпляру. Так Фабричний метод делегує операцію створення екземпляра підкласам [9, с.168].

У даному випадку загальним інтерфейсом виступає клас BotService, який опрацьовує всі запити, які приходять з клієнтської частини або від інтегрованих чат-ботів з месенджерів. Розглянемо метод createBot, який потрібен для інтеграції бота в систему. Код даного методу огорнутий в структуру try catch, яка дозволяє повернути користувачу стандартизовану помилку при виникненні неочікуваних помилок зі сторони месенджерів. Як аргументи, ми отримуємо назву бота в системі, унікальний токен, який надається месенджером та тип месенджера. Спочатку проводиться перевірка, чи існує в системі бот з таким типом месенджера. Якщо існує, тоді повертається помилка, оскільки можливо додати лише один бот з одного месенджера. Якщо не існує, тоді використовується клас FactoryService, який обирає екземпляр потрібного чат-бота для подальшої інтеграції. Програмний код методу createBot та FactoryService наведено на рис. 13-14.

```
public async createBot(data: CreateBotDto): Promise<Bot | HttpException> {
  try {
    await this.checkExistingBot(data.type);
    this.selectBot(data.type);
    const botData: IBotData = await this.bot.validateBot(data.token);
    const savedBot = await this.save( data: {
      ...botData,
      uiName: data.uiName,
      status: BotStatusEnum.ACTIVE,
    });
    await this.bot.setWebhook(savedBot.id);
    return savedBot;
  } catch (e) {
    throw new HttpException(
      response: {
        status: HttpStatus.BAD_REQUEST,
        message: e.message,
      },
      HttpStatus.BAD_REQUEST,
    );
  }
}
```

Рис. 13. Програмний код методу createBot.

```
import { Injectable } from '@nestjs/common';
import { TelegramBotService, ViberBotService } from './bots';
import { BotTypeEnum } from '../enums';

@Injectable()
export class FactoryService {
  constructor(
    private readonly telegramBotService: TelegramBotService,
    private readonly viberBotService: ViberBotService,
  ) {}
  public selectBot(type: BotTypeEnum) {
    switch (type) {
      case BotTypeEnum.TELEGRAM:
        return this.telegramBotService;
      case BotTypeEnum.VIBER:
        return this.viberBotService;
      default:
        return this.telegramBotService;
    }
  }
}
```

Рис. 14. Програмний код класу FactoryService

Даний програмний застосунок складається з двох основних режимів роботи: адміністратора та менеджера.

У додатку може бути лише один адміністратор, який буде автоматично створений лише після розгортання додатку. Всі інші користувачі, які були запрошені, матимуть роль менеджера. Додаток складається з чотирьох модулів, які мають різні режими роботи в залежності від ролі користувача: модуль користувачів, модуль клієнтів, модуль ботів, модуль чатів.

Користувачі можуть здійснювати чотири основні дії, які також залежать від ролі користувача: перегляд, створення, редагування, видалення. У залежності від ролі кожна дія може бути доступна або недоступна для користувача як на клієнтській, так і на серверній частині.

**Висновки.** У роботі було досліджено ключові аспекти роботи месенджерів та технології, за допомогою яких месенджери інтегруються у веб-сервіси. Досліджено методи інтеграції різних месенджерів у один користувацький інтерфейс.

Основною технічною складністю було завдання розробити алгоритм для поєднання різних месенджерів для виконання ідентичних операцій, таких як: додавання бота, встановлення вебхука, отримання оновлень, надсилання повідомлень.

Дане питання було вирішене використанням патерну – Фабричний метод, який дозволив безпосередньо перед використанням функціоналу обрати потрібний сервіс, який відповідає за інтеграцію того чи іншого месенджера.

**Бібліографія**

1. Adhikari S. Dependency Injection in Javascript. Medium. URL: https://medium.com/geekculture/dependency-injection-in-javascript-2d2e4ad9df49 (дата звернення: 19.11.2022).
2. Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2017. 350 p.
3. Compatible Time-sharing System: An Overview of CTSS. HitechNectar. URL: https://www.hitechnectar.com/blogs/compatible-time-sharing-system/ (date of access: 27.03.2023)
4. Compatible Time-sharing System: An Overview of CTSS. HitechNectar. URL: https://www.hitechnectar.com/blogs/compatible-time-sharing-system/ (date of access: 27.03.2023)

5. Documentation | NestJS - A progressive Node.js framework. Documentation | NestJS - A progressive No de.js framework. URL: https://docs.nestjs.com/ (дата звернення: 19.11.2022).
6. Docker Documentation. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 19.11.2022).
7. Dr Clifford Cocks CB. Homepage | University of Bristol. URL: http://www.bristol.ac.uk/graduation/honorary-degrees/hondeg08/cocks.html (дата звернення: 19.11.2022).
8. Fenton S. Pro TypeScript: Application-Scale JavaScript Development. Apress, 2014. 248 p.
9. Gourley D., Totty B. HTTP: The Definitive Guide. O'Reilly Media, Inc., 2002. 656 p.
10. Head First. Патерни проектування / Ерік Фрімен, Елізабет Робсон, Кеті Сьерра і Берт Бейтс; пер. з англ. Г. Якубовська. Харків : ВД «Фабула», 2020. 672 с.
11. How to Make a Chatbot in 2022: Benefits, Types, and Flow. Cleveroad Inc. – Web and App development company. URL: https://www.cleveroad.com/blog/how-to-make-a-chatbot/ (дата звернення: 19.11.2022).
12. Irrelevant patents on elliptic-curve cryptography. cr.yp.to. URL: https://cr.yp.to/ecdh/patents.html (дата звернення: 19.11.2022).
13. Lin Y. Most Popular Messaging Apps Worldwide [Jun 2022 Update]. Oberlo | Where Self Made is Made. URL: https://www.oberlo.com/statistics/most-popular-messaging-apps (date of access: 27.03.2023).
14. Massé M. REST API design rulebook. Sebastopol, CA : O'Reilly Media, 2011. 116 с.
15. Mikowski M., Powell J. Single Page Web Applications: JavaScript end-to-end. Shelter Island: Manning Publications Co., 2014. 514 p.
16. Node.js in Action / M. Cantelon та ін. Shelter Island: Manning Publications Co., 2014. 415 p.
17. Osmani A. Learning JavaScript Design Patterns. Beijing : O'Reilly, 2012. 199 p.
18. Reddy A. The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban. Addison-Wesley Professional, 2015. 384 p.
19. Riti P. Pro DevOps with Google Cloud Platform: With Docker, Jenkins, and Kubernetes. Apress, 2018. 296 p.
20. Scott E. SPA Design and Architecture: Understanding Single Page Web Applications. Shelter Island: Manning Publications Co., 2015. 275 p.
21. Scrumban: Essays On Kanban Systems For Lean Software Development. Modus Cooperandi Press, 2009. 180 p.
22. Sommerville I. Software engineering. 8-ме вид. Harlow, England : Addison-Wesley, 2007. 840 p.
23. The Evolution of Instant Messaging. Visual Capitalist. URL: https://www.visualcapitalist.com/evolution-instant-messaging/ (date of access: 23.11.2022).
24. The History Of WhatsApp. Feedough. URL: https://www.feedough.com/history-of-whatsapp/ (дата звернення: 19.11.2022).
25. Telegram Bot API. Telegram APIs. URL: https://core.telegram.org/bots/api (дата звернення: 19.11.2022).
26. Top Apps Ranking. *Similarweb.* URL: https://www.similarweb.com/apps/top/google/store-rank/ua/communication/top-free/ (дата звернення: 23.03.2023).
27. Viber Developers Hub. Viber Developers Hub. URL: https://developers.viber.com/ (дата звернення: 19.11.2022).
28. Wexler J. Get Programming with Node.js. Shelter Island: Manning Publications Co., 2019. 480 p.

**References**

1. Adhikari S. Dependency Injection in Javascript. Medium. URL: https://medium.com/geekculture/dependency-injection-in-javascript-2d2e4ad9df49 (дата звернення: 19.11.2022).
2. Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2017. 350 p.
3. Compatible Time-sharing System: An Overview of CTSS. HitechNectar. URL: https://www.hitechnectar.com/blogs/compatible-time-sharing-system/ (date of access: 27.03.2023)
4. Compatible Time-sharing System: An Overview of CTSS. HitechNectar. URL: https://www.hitechnectar.com/blogs/compatible-time-sharing-system/ (date of access: 27.03.2023)
5. Documentation | NestJS - A progressive Node.js framework. Documentation | NestJS - A progressive No de.js framework. URL: https://docs.nestjs.com/ (дата звернення: 19.11.2022).
6. Docker Documentation. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 19.11.2022).
7. Dr Clifford Cocks CB. Homepage | University of Bristol. URL: http://www.bristol.ac.uk/graduation/honorary-degrees/hondeg08/cocks.html (дата звернення: 19.11.2022).
8. Fenton S. Pro TypeScript: Application-Scale JavaScript Development. Apress, 2014. 248 p.

9. Gourley D., Totty B. HTTP: The Definitive Guide. O'Reilly Media, Inc., 2002. 656 p.
10. Head First. Патерни проектування / Ерік Фрімен, Елізабет Робсон, Кеті Сьерра і Берт Бейтс; пер. з англ. Г. Якубовська. Харків : ВД «Фабула», 2020. 672 с.
11. How to Make a Chatbot in 2022: Benefits, Types, and Flow. Cleveroad Inc. – Web and App development company. URL: https://www.cleveroad.com/blog/how-to-make-a-chatbot/ (дата звернення: 19.11.2022).
12. Irrelevant patents on elliptic-curve cryptography. cr.yp.to. URL: https://cr.yp.to/ecdh/patents.html (дата звернення: 19.11.2022).
13. Lin Y. Most Popular Messaging Apps Worldwide [Jun 2022 Update]. Oberlo | Where Self Made is Made. URL: https://www.oberlo.com/statistics/most-popular-messaging-apps (date of access: 27.03.2023).
14. Massé M. REST API design rulebook. Sebastopol, CA : O'Reilly Media, 2011. 116 с.
15. Mikowski M., Powell J. Single Page Web Applications: JavaScript end-to-end. Shelter Island: Manning Publications Co., 2014. 514 p.
16. Node.js in Action / M. Cantelon та ін. Shelter Island: Manning Publications Co., 2014. 415 p.
17. Osmani A. Learning JavaScript Design Patterns. Beijing : O'Reilly, 2012. 199 p.
18. Reddy A. The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban. Addison-Wesley Professional, 2015. 384 p.
19. Riti P. Pro DevOps with Google Cloud Platform: With Docker, Jenkins, and Kubernetes. Apress, 2018. 296 p.
20. Scott E. SPA Design and Architecture: Understanding Single Page Web Applications. Shelter Island: Manning Publications Co., 2015. 275 p.
21. Scrumban: Essays On Kanban Systems For Lean Software Development. Modus Cooperandi Press, 2009. 180 p.
22. Sommerville I. Software engineering. 8-ме вид. Harlow, England : Addison-Wesley, 2007. 840 p.
23. The Evolution of Instant Messaging. Visual Capitalist. URL: https://www.visualcapitalist.com/evolution-instant-messaging/ (date of access: 23.11.2022).
24. The History Of WhatsApp. Feedough. URL: https://www.feedough.com/history-of-whatsapp/ (дата звернення: 19.11.2022).
25. Telegram Bot API. Telegram APIs. URL: https://core.telegram.org/bots/api (дата звернення: 19.11.2022).
26. Top Apps Ranking. *Similarweb.* URL: https://www.similarweb.com/apps/top/google/store-rank/ua/communication/top-free/ (дата звернення: 23.03.2023).
27. Viber Developers Hub. Viber Developers Hub. URL: https://developers.viber.com/ (дата звернення: 19.11.2022).
28. Wexler J. Get Programming with Node.js. Shelter Island: Manning Publications Co., 2019. 480 p.