AKADEMIA NAUK STOSOWANYCH

W NOWYM SĄCZU

Wydział nauk Inżynieryjnych

Katedra informatyki

Obraz zawierający fioletowy, zrzut ekranu, Magenta, fiołek

Opis wygenerowany automatycznieDOKUMENTACJA PROJEKTOWA

INZYNIERIA OPROGRAMOWANIA

**TIME**

**ORGANIZER**

**Autorzy:**

Nika Lytvynchuk

Kuba Buczek

Klaudia Łyszczarz

Grzegorz Bobak

Gabriela Zborowska

Spis treści

[1. Tytuł 3](#_Toc156432219)

[2. Link do repozytorium: 3](#_Toc156432220)

[3. Opis aplikacji 4](#_Toc156432221)

[4. Określenie wymagań funkcjonalnych i niefunkcjonalnych 5](#_Toc156432222)

[Wymagania funkcjonalne: 5](#_Toc156432223)

[Wymagania niefunkcjonalne: 5](#_Toc156432224)

[5. Konkurencja 7](#_Toc156432225)

[6. Dobór technologii 8](#_Toc156432226)

[7. Planowanie projektu 9](#_Toc156432227)

[8. Diagramy i scenariusze 14](#_Toc156432228)

[8.1. Diagram ERD 14](#_Toc156432229)

[8.2. Diagram Przypadków użycia 15](#_Toc156432230)

[8.3. Scenariusze 16](#_Toc156432231)

[9. Implementacja 25](#_Toc156432232)

[9.1. Front-END 25](#_Toc156432233)

[9.2. Back-END 31](#_Toc156432234)

[10. Podsumowanie prac (Czas) 42](#_Toc156432235)

[11. Testowanie 43](#_Toc156432236)

[12. Podręcznik użytkownika 49](#_Toc156432237)

[13. Podsumowanie projektu. Wnioski 56](#_Toc156432238)

[14. Bibliografia 57](#_Toc156432239)

# Tytuł

Tytuł naszego projektu: TimeOrganizer

# Link do repozytorium:

<https://github.com/nikalitvinchuk/timeorganizer>

# Opis aplikacji

Aplikacja oferuje możliwość rejestracji użytkowników, korzystając z lokalnej bazy danych SQLite do przechowywania i odczytu danych, na przykład w przypadku logowania. Po zalogowaniu tworzone są dane sesji użytkownika, które między innymi, umożliwiają automatyczne wylogowanie użytkownika po upływie określonego czasu nieaktywności.

Po zalogowaniu użytkownika, pierwszym ekranem, który się wyświetla, jest flyout - strona główna **(HomePage)** z najważniejszymi informacjami oraz krótkimi powiadomieniami dotyczącymi zadań i wydarzeń.

Aplikacja zapewnia dostęp do różnych funkcji poprzez menu, które zawiera następujące podstrony:

* **Kalendarz (Calendar)**: Umożliwia podgląd wydarzeń i zadań zaplanowanych na konkretne dni.
* **Lista zadań (To-Do List)**: Pozwala na tworzenie listy zadań do wykonania, uwzględniając ich opis, datę stworzenia, termin, podzadania, procent wykonania i status.
* **Notatki (Notes)**: Umożliwia użytkownikom tworzenie i przechowywanie własnych notatek.
* **Statystyki (Statistics)**: Dostarcza statystyki postępu w realizacji zadań i celów użytkownika.
* **Kontakty (Contacts)**: Zawiera formularz kontaktowy w celu komunikowania się z właścicielami aplikacji, przesłanie zgłoszeń.
* **Instagram**: Pozwala na wyszukiwanie kont związanych z aplikacją w wybranej społeczności.
* **Ustawienia (Settings)**: Umożliwia podgląd aktualnych danych użytkownika oraz zmiany hasła, maila oraz usunięcie własnego konta
* **Wylogowanie (LogOut)**: Pozwala na bezpieczne zakończenie sesji i wylogowanie użytkownika.

# Określenie wymagań funkcjonalnych i niefunkcjonalnych

## Wymagania funkcjonalne:

1. **Rejestracja**: Użytkownicy mogą zarejestrować nowe konta w aplikacji.
2. **Logowanie**: Zarejestrowani użytkownicy mają możliwość zalogowania się do swoich kont.
3. **Menu Aplikacji**: Dostępne jest menu umożliwiające łatwą nawigację między zakładkami i funkcjami.
4. **Kalendarz**: Użytkownicy mogą przeglądać kalendarz, gdzie mamy podgląd do zadań na konkretny dzień.
5. **Zadania**: Możliwość dodawania nowych zadań, uwzględniających opis, termin, procent wykonania, podzadania i status zadania.
6. **Lista Zadań**: Użytkownicy mają dostęp do przeglądania dodanych zadań w formie listy.
7. **Notatki**: Obsługa dodawania, podglądu oraz usunięcia notatek w wygodny sposób.
8. **Statystyki**: Użytkownicy mogą śledzić postęp w realizacji zadań za pomocą statystyk.
9. **Wyszukiwanie Społecznościowe**: Aplikacja umożliwia wyszukiwanie kont związanych z aplikacją w wybranej społeczności. Np. Instagram

## Wymagania niefunkcjonalne:

1. **Połączenie z Bazą Danych**: Aplikacja sprawdza poprawność połączenia z bazą danych SQLite.
2. **Przechowywanie Danych**: Dane są przechowywane trwale i niezmiennie w bazie danych.
3. **Walidacja Danych**: Wprowadzone przez użytkownika dane są weryfikowane i walidowane.
4. **Obsługa Sesji**: Zarządzanie sesją użytkownika umożliwia wyświetlanie spersonalizowanych danych.
5. **Przyjazny Interfejs**: Interfejs użytkownika jest intuicyjny i dostosowany do preferencji użytkowników.
6. **Zgodność z Standardami**: Aplikacja jest zgodna z obowiązującymi standardami i regulacjami.
7. **Zarządzanie Błędami**: W kodzie źródłowym obecne są mechanizmy zarządzania błędami.
8. **Dokumentacja**: Istnieje dokładna dokumentacja dotycząca aplikacji, ułatwiająca jej zrozumienie i rozwijanie.
9. **Testowanie i Jakość Oprogramowania**: Aplikacja jest dokładnie przetestowana względem kodu.
10. **Zarys widoku strony głównej** jest widoczny na rys. nr. 1

**Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie**

# Konkurencja

W otwartym dostępie można znaleźć różnego typu rozwiązania podobnych programów do naszego zarówno w formie aplikacji webowych, mobilnych, jak i desktopowych, które oferują podobne funkcję zarządzania czasem m. in.:

* **Todoist:** Todoist to znana aplikacja do zarządzania zadaniami, która oferuje prosty interfejs umożliwiający tworzenie zadań, przypisywanie priorytetów, terminów, etykiet oraz list. Wniosek: TimeOrganizer może się wyróżniać poprzez rozbudowane funkcje statystyk postępu, sposobu prezentowania danych, automatycznego wylogowania po określonym czasie nieaktywności
* **Microsoft To-Do:** Aplikacja Microsoftu łączy się z ekosystemem Office 365, zapewniając integrację z innymi narzędziami Microsoftu. Wniosek: TimeOrganizer może konkurować poprzez możliwość tworzenia ogólnych statystyk oraz tworzenie notatek niezależnie od dodanych zadań.
* **Evernote:** Skupia się głównie na notatkach, ma funkcję list zadań i przypomnień. Wniosek: TimeOrganizer wyróżnia się poprzez silniejsze skupienie na zarządzaniu zadaniami, wydarzeniami oraz bardziej zaawansowanej organizacji czasu.
* **Trello:** Platforma tablicowa doskonale nadająca się do zarządzania projektami. Wniosek: TimeOrganizer skupia się bardziej na indywidualnym zarządzaniu zadaniami oraz oferuje możliwość integracji z różnymi funkcjami, zapewniając bardziej kompleksowe podejście do osobistego zarządzania czasem.

# Dobór technologii

W naszym projekcie, zdecydowaliśmy się wykorzystać różne technologie, narzędzia oraz podejścia, aby stworzyć aplikację mobilną, która spełni oczekiwania użytkowników i będzie efektywna, m.in.:

1. **.NET MAUI**: Dzięki .NET MAUI możemy rozwijać tworzyć jedną bazową aplikację, która działa na wielu platformach, takich jak Android, iOS, macOS i Windows
2. **SQLite**: Do przechowywania danych używamy lokalnej bazy danych SQLite. Jest to efektywny i wydajny sposób na trwałe przechowywanie informacji użytkowników, takich jak zadania, notatki i statystyki.
3. **Trójmodel ViewModel**: W projekcie stosujemy trójmodel ViewModel (3MVVM) jako architekturę, która pomaga oddzielić logikę biznesową od warstwy prezentacji. Model View View-Model (MVVM) umożliwia skuteczną organizację kodu, ułatwia testowanie i zapewnia elastyczność.
4. **Jira**: W celu zarządzania projektem i śledzenia postępów, używamy narzędzia Jira. To narzędzie do zarządzania projektem, które pomaga w planowaniu, monitorowaniu i raportowaniu postępów w projekcie.
5. **GitHub**: Cały kod źródłowy projektu jest hostowany na platformie GitHub. Ułatwia to zarządzanie kodem, współpracę w zespole i kontrolę wersji.
6. **Brazor** - innowacyjny framework opracowany przez firmę Microsoft, używamy do budowy interfejsów użytkownika (UI) oparty jest na języku C# i .NET, umożliwia m.in. tworzenie dynamicznych aplikacji internetowych. Framework pozwala używać C# zarówno po stronie klienta, jak i serwera.
7. **Narzędzia Testowe**: Wykorzystujemy narzędzia do testowania dostępne w środowisku Visual Studio, takie jak Test Manager, Test Runner. Zapewniają one kompleksowe wsparcie dla testów ręcznych, zautomatyzowanych oraz analizy wyników, umożliwiając skuteczne śledzenie postępów testów i debugowanie błędów.

# Planowanie projektu

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SPRINT 1** | | | | | |
| Podsumowanie | Klucz zgłoszenia | Status | Osoba przypisana | Rozwiązane | Opis |
| Wczytanie, sortowanie, filtrowanie | SCRUM-8 | Gotowe | Grzegorz Bobak | 07/lis/23 17:37 | Wczytanie listy wszystkich zadań z bazy z podziałem na kategorię  Sortowanie zadań wg kategorii WYKONANE/NIEWYKONANE  Sortowanie zadań wg czasu  Sortowanie czasu wg priorytetu |
| Obsługa zdarzenia dodawania zadań do bazy | SCRUM-7 | Gotowe | Grzegorz Bobak | 07/lis/23 17:37 |  |
| Stworzenie widoku strony głównej + podstrony z formularzem | SCRUM-6 | Gotowe | Klaudia Łyszczarz | 02/gru/23 08:03 | Stworzenie widoku strony głównej (front) + podstrony z formularzem do wpisywania danych (dotyczy nowego zadania) |
| Stworzenie mechanizmu logowania i rejestracji | SCRUM-5 | Gotowe | Nika Lytvynchuk | 01/lis/23 07:06 |  |
| Znalezienie, rozwiązania, stworzenie bazy danych SQLite | SCRUM-4 | Gotowe | Jakub Buczek | 27/paź/23 19:12 | Znalezienie, rozwiązania, stworzenie bazy danych SQLite  - tabela użytkownika → login/hasło/mail/nazwa użytkownika,  - zadania → \*nazwa\*/\*czas i data\* (sortowanie)/\*status\* (aktywne, usunięte, zakończone)/\*użytkownik\*/\*data zakończenia\*/\*progres w %\* (później graficznie odwzorować co 25%) → powiązane z tabelą \*użytkownicy\*  - podzadania - powiązane z tabelą \*zadania\* |
| Stworzenie tablicy, dodanie tasków, prowadzenie sprinta | SCRUM-3 | Gotowe | Gabriela Zborowska | 27/paź/23 19:20 | Stworzenie tablicy, dodanie tasków, prowadzenie sprinta |
| Opracowanie dokumentacji | SCRUM-2 | Gotowe | Nika Lytvynchuk | 12/lis/23 22:00 | Opracowanie dokumentacji |
| **SPRINT 2** | | | | | |
| Podsumowanie | Klucz zgłoszenia | Status | Osoba przypisana | Rozwiązane | Opis |
| Setting obsługa | SCRUM-15 | Gotowe | Jakub Buczek | 11/lis/23 19:35 | Setting obsługa |
| Backend podstrony "Notatki" | SCRUM-14 | Gotowe | Jakub Buczek | 11/lis/23 15:51 | Backend podstrony "Notatki" |
| Front do podstrony "Notatki" | SCRUM-13 | Gotowe | Gabriela Zborowska | 15/lis/23 21:24 | Front do podstrony "Notatki" |
| Dodawanie zadań kontynuacja/wczytywanie/filtrowanie/edycja/modyfikacja | SCRUM-12 | Gotowe | Grzegorz Bobak | 09/gru/23 11:26 | # Dodawanie zadań - kontynuacja  # Wczytywanie  # Filtrowanie  # Edycja/modyfikacja |
| ToDo (poprawic, najpierw wyswietlana lista zadan, na dole przycisk dodaj który przenosi na inną podstrone) | SCRUM-11 | Gotowe | Klaudia Łyszczarz | 02/gru/23 08:02 | ToDo (poprawic, najpierw wyswietlana lista zadan, na dole przycisk dodaj który przenosi na inną podstroneZakładke instagram można usunąć i dodac link do contacts. Po kliknięciu przeniesienie na strone. |
| Dokumentacja | SCRUM-10 | W toku | Nika Lytvynchuk |  | Dokumentacja |
| Plan na rozwój | SCRUM-9 | Do zrobienia | Grzegorz Bobak |  | Zrobienie z typu zadania listy wybierlanej  Tak aby później można było np filtrować po danym typie  Uzasadnienie:  W obecnej sytuacji gdy typ jest podawany z ręki grupowanie po typie będzie ciężkie do zrobienia bo każde zadanie może mieć inaczej wpisany typ |
| Wczytanie, sortowanie, filtrowanie | SCRUM-8 | Gotowe | Grzegorz Bobak | 07/lis/23 17:37 | Wczytanie listy wszystkich zadań z bazy z podziałem na kategorię  Sortowanie zadań wg kategorii WYKONANE/NIEWYKONANE  Sortowanie zadań wg czasu  Sortowanie czasu wg priorytetu |
| Obsługa zdarzenia dodawania zadań do bazy | SCRUM-7 | Gotowe | Grzegorz Bobak | 07/lis/23 17:37 | Obsługa zdarzenia dodawania zadań do bazy |
| Stworzenie widoku strony głównej + podstrony z formularzem | SCRUM-6 | Gotowe | Klaudia Łyszczarz | 02/gru/23 08:03 | Stworzenie widoku strony głównej (front) + podstrony z formularzem do wpisywania danych (dotyczy nowego zadania) |
| **SPRINT 3** | | | | | |
| Podsumowanie | Klucz zgłoszenia | Status | Osoba przypisana | Rozwiązane | Opis |
| Hashowanie haseł | SCRUM-27 | Gotowe | Jakub Buczek | 19/lis/23 07:40 | Hashowanie haseł |
| Zmiana struktury strony/podłączenie Blazor Bootstrap | SCRUM-26 | Gotowe | Grzegorz Bobak | 29/lis/23 11:03 | Zmiana struktury strony/podłączenie Blazor Bootstrap |
| Zmiana struktury strony/podłączenie Blazor Bootstrap | SCRUM-25 | Gotowe | Jakub Buczek | 29/lis/23 11:15 | Zmiana struktury strony/podłączenie Blazor Bootstrap |
| Kontynuacja i dopracowanie "Podzadań" | SCRUM-18 | Gotowe | Grzegorz Bobak | 09/gru/23 17:25 | Kontynuacja i dopracowanie "Podzadań" |
| Dokumentacja | SCRUM-10 | W toku | Nika Lytvynchuk |  | Dokumentacja |
| **SPRINT 4** | | | | | |
| Podsumowanie | Klucz zgłoszenia | Status | Osoba przypisana | Rozwiązane | Opis |
| Podsumowanie zadań - backend | SCRUM-28 | W toku | Grzegorz Bobak |  | Podsumowanie zadań - backend |
| Uzupełnienie diagramów przypadków użycia | SCRUM-24 | Gotowe | Jakub Buczek | 07/gru/23 19:30 | Uzupełnienie diagramów przypadków użycia |
| Wygląd podstrony "Contacts" | SCRUM-23 | Gotowe | Klaudia Łyszczarz | 02/gru/23 08:01 | Wygląd podstrony "Contacts" |
| Dopracowanie wyświetlania podstrony "Notatki" | SCRUM-21 | W toku | Gabriela Zborowska |  | Dopracowanie wyświetlania podstrony "Notatki" |
| Edycja "Zadań" | SCRUM-19 | Gotowe | Grzegorz Bobak | 09/gru/23 17:25 | Edycja "Zadań" |
| Kontynuacja i dopracowanie "Podzadań" | SCRUM-18 | Gotowe | Grzegorz Bobak | 09/gru/23 17:25 | Kontynuacja i dopracowanie "Podzadań" |
| Sesja, autowylogowanie | SCRUM-16 | Gotowe | Nika Lytvynchuk | 28/lis/23 22:14 | Sesja, autowylogowanie |
| Dodawanie zadań kontynuacja/wczytywanie/filtrowanie/edycja/modyfikacja | SCRUM-12 | Gotowe | Grzegorz Bobak | 09/gru/23 11:26 | # Dodawanie zadań - kontynuacja  # Wczytywanie  # Filtrowanie  # Edycja/modyfikacja |
| ToDo (poprawic, najpierw wyswietlana lista zadan, na dole przycisk dodaj który przenosi na inną podstrone | SCRUM-11 | Gotowe | Klaudia Łyszczarz | 02/gru/23 08:02 | ToDo (poprawic, najpierw wyswietlana lista zadan, na dole przycisk dodaj który przenosi na inną podstrone)/ Zakładke instagram można usunąć i dodac link do contacts. Po kliknięciu przeniesienie na strone. |
| Dokumentacja | SCRUM-10 | W toku | Nika Lytvynchuk |  | Dokumentacja |
| **SPRINT 5** | | | | | |
| Podsumowanie | Klucz zgłoszenia | Status | Osoba przypisana | Zaktualizowano | Opis |
| Filtrowanie notatek | SCRUM-30 | Do zrobienia | Jakub Buczek | 07/gru/23 19:39 | Filtrowanie notatek |
| Podsumowanie zadań - frontend | SCRUM-29 | Do zrobienia | Klaudia Łyszczarz | 07/gru/23 19:32 | Podsumowanie zadań - frontend |
| Podsumowanie zadań - backend | SCRUM-28 | W toku | Grzegorz Bobak | 16/gru/23 00:10 | Podsumowanie zadań - backend |
| Dodanie kafelek na "Home", poprawa frontu "Home" | SCRUM-22 | Gotowe | Klaudia Łyszczarz | 02/gru/23 08:03 | Dodanie kafelek na "Home", poprawa frontu "Home" |
| Dopracowanie wyświetlania podstrony "Notatki" | SCRUM-21 | W toku | Gabriela Zborowska | 16/gru/23 00:10 | Dopracowanie wyświetlania podstrony "Notatki" |
| Front podstrony "Settings" | SCRUM-20 | Do zrobienia | Gabriela Zborowska | 18/lis/23 09:00 | Front podstrony "Settings" |
| Podstrona "Contacts" | SCRUM-17 | Gotowe | Nika Lytvynchuk | 28/lis/23 22:14 | Podstrona "Contacts" |
| Dokumentacja | SCRUM-10 | W toku | Nika Lytvynchuk | 16/gru/23 00:10 | Dokumentacja |
| Plan na rozwój | SCRUM-9 | Do zrobienia | Grzegorz Bobak | 02/gru/23 08:01 | Zrobienie z typu zadania listy wybieralnej  Tak aby później można było np filtrować po danym typie  Uzasadnienie:  W obecnej sytuacji gdy typ jest podawany z ręki grupowanie po typie będzie ciężkie do zrobienia bo każde zadanie może mieć inaczej wpisany typ |
| Stworzenie widoku strony głównej + podstrony z formularzem | SCRUM-6 | Gotowe | Klaudia Łyszczarz | 02/gru/23 08:03 | Stworzenie widoku strony głównej (front) + podstrony z formularzem do wpisywania danych (dotyczy nowego zadania) |
| **SPRINT 6** | | | | | |
| Podsumowanie | Klucz zgłoszenia | Status | Osoba przypisana | Rozwiązane | Opis |
| Poprawa frontu "Notatek" | SCRUM-35 | Do zrobienia | Gabriela Zborowska |  | Poprawa frontu "Notatek" |
| Widok strony HomePage | SCRUM-34 | Do zrobienia | Nika Lytvynchuk |  | Widok strony HomePage |
| Testy | SCRUM-33 | Do zrobienia | Jakub Buczek |  | Testy |
| Podpięcie bibliotek + ustawienia | SCRUM-32 | Do zrobienia | Klaudia Łyszczarz |  | Podpięcie bibliotek + ustawienia |
| Statystyki wstęp | SCRUM-31 | Do zrobienia | Grzegorz Bobak |  | Statystyki |
| **SPRINT 7** | | | | | |
| Podsumowanie | Klucz zgłoszenia | Status | Osoba przypisana | Rozwiązane | Opis |
| Poprawki w dokumentacji - wszyscy | SCRUM-40 | Do zrobienia | Nika Lytvynchuk |  | Poprawki w dokumentacji - wszyscy |
| Poprawki w dokumentacji - wszyscy | SCRUM-39 | Do zrobienia | Klaudia Łyszczarz |  | Poprawki w dokumentacji - wszyscy |
| Poprawki w dokumentacji - wszyscy | SCRUM-38 | Do zrobienia | Grzegorz Bobak |  | Poprawki w dokumentacji - wszyscy |
| Poprawki w dokumentacji - wszyscy | SCRUM-37 | Do zrobienia | Jakub Buczek |  | Poprawki w dokumentacji - wszyscy |
| Poprawki w dokumentacji - wszyscy | SCRUM-36 | Do zrobienia | Gabriela Zborowska |  | Poprawki w dokumentacji - wszyscy |

# Diagramy i scenariusze

## Diagram ERD

Obraz zawierający tekst, zrzut ekranu, diagram, Równolegle

Opis wygenerowany automatycznie

## Diagram Przypadków użycia

Obraz zawierający krąg, rysowanie, diagram

Opis wygenerowany automatycznie

## Scenariusze

|  |  |
| --- | --- |
| Nazwa | Scenariusz 1 – rejestracja |
| Aktor | Użytkownik |
| Warunki | Użytkownik znajduje się na stronie głównej |
| Opis | Użytkownik może się zarejestrować podając swoje dane, które zostaną zapisane w bazie danych |
| Ścieżka główna | 1. Użytkownik przechodzi na podstronę z rejestracją 2. Użytkownik uzupełnia dane do rejestracji 3. Użytkownik klika przycisk zarejestruj 4. Pojawia się informacja o poprawnie zarejestrowanym użytkowniku |
| Ścieżka alternatywna | 1. Użytkownik przechodzi na podstronę z rejestracją 2. Użytkownik podaje niepoprawne dane do rejestracji 3. Użytkownik klika przycisk zarejestruj 4. Pojawia się komunikat z informacją dot. niepoprawnie wprowadzonych danych |
| Warunki końcowe | Rejestracja użytkownika w systemie |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 2 – logowanie |
| Aktor | Użytkownik |
| Warunki | Użytkownik znajduje się na stronie głównej |
| Opis | Użytkownik może się zalogować jeśli poda poprawne dane, które istnieją w bazie danych |
| Ścieżka główna | 1. Użytkownik przechodzi na podstronę logowania 2. Użytkownik uzupełnia dane do logowania 3. Użytkownik klika przycisk zaloguj 4. Pojawia się informacja o poprawnych danych i pomyślnym zalogowaniu użytkownika |
| Ścieżka alternatywna | 1. Użytkownik przechodzi na podstronę logowania 2. Użytkownik uzupełnia dane do logowania, których nie ma w bazie 3. Użytkownik klika przycisk zaloguj 4. Pojawia się informacja o błednie podanych danych |
| Warunki końcowe | Logowanie na HomePage |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 3 – dodawanie zadania |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce ToDoPage |
| Opis | Użytkownik dodaje zadanie jakie będzie chciał wykonać |
| Ścieżka główna | 1. Użytkownik uzupełnia pola potrzebne do dodania zadania 2. Użytkownik klika przycisk dodaj 3. Pojawia się informacja o poprawnie dodanym zadaniu |
| Ścieżka alternatywna | 1. Użytkownik niepoprawnie uzupełnia pola potrzebne dodania zadania 2. Użytkownik naciska przycisk dodaj 3. Wyświetla się komunikat informujący co jest nie tak |
| Warunki końcowe | Dodanie zadania do bazy |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 4 – wyświetlanie i filtrowanie zadań |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce ToDoPage |
| Opis | Użytkownik wyświetla zadania wszystkie lub według podanych kryteriów |
| Ścieżka główna | 1. Użytkownik klika przycisk filtruj 2. Wyświetla się lista wszystkich zadań jakie były przypisane do użytkownika w aktualnym dniu |
| Ścieżka alternatywna | 1. Użytkownik wpisuje dane do pól filtra 2. Użytkownik naciska przycisk filtruj 3. Wyświetla się lista zadań która mieści się w podanym kryterium |
| Ścieżka alternatywna | 1. Użytkownik wpisuje dane do pól filtra 2. Użytkownik naciska przycisk filtruj 3. Nie wyświetla się żądne zadanie ponieważ nie ma zadania o podanym kryterium |
| Warunki końcowe | Użytkownika wyświetla jedno lub więcej zadań |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 5 – przejście do szczegółów zadania |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce ToDoPage / Warunek główny scenariusza 4 |
| Opis | Użytkownik wyświetla więcej informacji o zadaniu |
| Ścieżka główna | 1. Użytkownik wybiera zadanie z wyświetlonej listy 2. Użytkownik zostaje przeniesiony na widok szczegółów zadania |
| Warunki końcowe | Wyświetlenie szczegółów zadania oraz listy podzadań zadania |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 6 – dodanie pod zadania |
| Aktor | Użytkownik |
| Warunki | Scenariusz 5 |
| Opis | Użytkownik dodaje pod zadnia zadania głównego które będą składały się na całość wykonania |
| Ścieżka główna | 1. Użytkownik uzupełnia pola potrzebne do dodania pod zadania 2. Użytkownik klika przycisk dodaj pod zadanie 3. Pojawia się informacja o poprawnie dodanym pod zadania |
| Ścieżka alternatywna | 1. Użytkownik niepoprawnie uzupełnia pola potrzebne dodania pod zadania 2. Użytkownik naciska przycisk dodaj 3. Wyświetla się komunikat informujący co jest nie tak |
| Warunki końcowe | Dodanie pod zadania jako część zadania głównego |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 7 – oznaczenie zadań jako wykonane |
| Aktor | Użytkownik |
| Warunki | Scenariusz 5 |
| Opis | Użytkownik oznacza zadania jako wykonane |
| Ścieżka główna | 1. Użytkownik zaznacza pod zadanie jako wykonane |
| Ścieżka główna | 1. Użytkownik zaznacza zadanie jako wykonane |
| Ścieżka alternatywna | 1. Użytkownik zaznacza zadanie główne jako wykonane podczas gdy podzadania nie są wykonane 2. Wyświetla się komunikat że najpierw podzadania muszą zostać wykonane |
| Warunki końcowe | Oznaczenie zadania głównego jako wykonane |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 8 – Zmiana hasła |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Settings |
| Opis | Użytkownik zmienia hasło dostępu do aplikacji |
| Ścieżka główna | 1. Użytkownik wybiera przycisk zmień hasło 2. Użytkownik podaje nowe hasło 3. Użytkownik ponownie podaje nowe hasło 4. Użytkownik podaje obecne hasło 5. Użytkownik wybiera przycisk zmiana hasła |
| Ścieżka alternatywna | 6a. Użytkownik podał 2 razy różne nowe hasła  7a. Wyświetla się komunikat że hasła niezgodne  6b. Użytkownik podał błędne obecne hasło  7b. Wyświetla się informacja o błędnym obecnym haśle.  8. Hasło nie zostaje zmienione |
| Warunki końcowe | Hasło zostaje zmienione |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 9 – Zmiana email |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Settings |
| Opis | Użytkownik zmienia email w aplikacji |
| Ścieżka główna | 1. Użytkownik wybiera przycisk zmień email 2. Użytkownik podaje nowy email 3. Użytkownik podaje obecne hasło 4. Użytkownik wybiera przycisk zmiana hasła |
| Ścieżka alternatywna | 5. Użytkownik podał błędne obecne hasło  6. Wyświetla się informacja o błędnym obecnym haśle.  7. Email nie zostaje zmienione |
| Warunki końcowe | Email zostaje zmieniony |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 10 – Zmiana hasła i email |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Settings |
| Opis | Użytkownik zmienia hasło i email dostępu do aplikacji |
| Ścieżka główna | 1. Użytkownik wybiera przycisk zmień email i haslo 2. Użytkownik podaje nowy email 3. Użytkownik podaje nowe hasło 4. Użytkownik ponownie podaje nowe hasło 5. Użytkownik podaje obecne hasło 6. Użytkownik wybiera przycisk zmiana hasła |
| Ścieżka alternatywna | 7a. Użytkownik podał 2 razy różne nowe hasła  8a. Wyświetla się komunikat że hasła niezgodne  7b. Użytkownik podał błędne obecne hasło  8b. Wyświetla się informacja o błędnym obecnym haśle.  9. Hasło nie zostaje zmienione |
| Warunki końcowe | Hasło i email zostaje zmienione |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 11 – Usunięcie konta |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Settings |
| Opis | Użytkownik usuwa swoje konto z aplikacji |
| Ścieżka główna | 1. Użytkownik wybiera przycisk Usuń konto 2. Użytkownik podaje obecne hasło 3. Użytkownik potwierdza usunięcie konta |
| Ścieżka alternatywna | 4. Użytkownik podał błędne obecne hasło  5. Wyświetla się informacja o błędnym obecnym haśle.  9. konto nie zostaje usunięte |
| Warunki końcowe | Konto zostaje usuniete |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 12 – Podstrona kalendarz |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Kalendarz |
| Opis | Użytkownik może podejrzeć zadania przypisane na konkretny dzień |
| Ścieżka główna | 1. Użytkownik wybiera datę 2. Po kliknięciu w konkretny dzień – podgląd aktualnych zadań |
| Ścieżka alternatywna | W przypadku braku zadań – brak wpisów |
| Warunki końcowe | Użytkownik uzyskuje informację na temat zadań przypisanych w konkretnym dniu |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 13 – Podgląd danych w ustawieniach |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Ustawienia |
| Opis | Użytkownik może podejrzeć dane dot. stworzonego konta |
| Ścieżka główna | 1. Użytkownik po zalogowaniu przechodzi na podstronę Setting 2. Użytkownik odczytuje aktualne dane dot. własnego konta |
| Ścieżka alternatywna | Wyświetlenie komunikatu w przypadku błędu związanego z pobraniem danych z bazy danych |
| Warunki końcowe | Użytkownik ma podgląd dot. konta i ustawień |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 14 – Contacts |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Contacts |
| Opis | Użytkownik może wysłać zgłoszenie do moderatorów |
| Ścieżka główna | 1. Użytkownik podaje swój adres mailowy 2. Użytkownik pisze zgłoszenie/prośbę/komentarz 3. Użytkownik klika w przycisk „Wyślij” 4. Użytkownik dostaje informację zwrotną na wcześniej podanego maila |
| Ścieżka alternatywna | 1. Użytkownik podaje błędny adres mailowy lub nie wpisuje danych w polu „komentarz” 2. Użytkownik klika w przycisk „Wyślij” 3. Użytkownik dostaje komunikat o błędnych danych lub pustym polu |
| Warunki końcowe | Użytkownik może się skontaktować z moderatorami |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 15 – Statistics |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany / Znajduje się zakładce Statistics |
| Opis | Użytkownik może zobaczyć swoje postępy |
| Ścieżka główna | 1. Użytkownik przechodzi na podstronę statistics 2. Użytkownik może podejrzeć wykresy oraz posumowanie dot. wykonanych i niewykonanych zadań |
| Ścieżka alternatywna | 1. Użytkownik przechodzi na podstronę statistics 2. Użytkownik jeszcze nie ma dodanych zadań – na stronie znajduje się odpowiedni komunikat |
| Warunki końcowe | Użytkownik wyświetla statystyki dot. zadań |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 16 – Wylogowanie |
| Aktor | Użytkownik |
| Warunki | Użytkownik jest zalogowany |
| Opis | Użytkownik może się wylogować |
| Ścieżka główna | 1. Użytkownik znajdując się na każdej z podstron może się wylogować za pomocą dostępnego menu klikając w przycisk „Logout” 2. Użytkownik przechodzi na MainPage z możliwością ponownego logowania lub rejestracji |
| Ścieżka alternatywna | Brak |
| Warunki końcowe | Użytkownik zostaje wylogowany |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 17 - Przeglądanie notatek |
| Aktor | Użytkownik Aplikacji |
| Warunki początkowe | Użytkownik jest zalogowany do aplikacji i znajduje się na stronie "Notatki" |
| Opis | Użytkownik otwiera podstronę "Notatki".  System wyświetla listę dostępnych notatek.  Użytkownik przegląda notatki. |
| Ścieżka główna | Użytkownik otwiera stronę "Notatki".  System pobiera i wyświetla notatki użytkownika. |
| Ścieżka alternatywna | Jeśli użytkownik nie jest zalogowany, system przekierowuje go do strony logowania. |
| Warunki końcowe | Użytkownik przegląda notatki, system utrzymuje stan strony "Notatki". |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 18 - Dodawanie nowej notatki |
| Aktor | Użytkownik Aplikacji |
| Warunki początkowe | Użytkownik jest zalogowany do aplikacji i znajduje się na stronie "Notatki". |
| Opis | Użytkownik klika przycisk "Dodaj notatkę".  System otwiera formularz do dodawania nowej notatki.  Użytkownik wprowadza treść notatki.  Użytkownik zapisuje nową notatkę.  System dodaje notatkę do listy. |
| Ścieżka główna | Użytkownik otwiera stronę "Notatki".  Użytkownik kliknie przycisk "Dodaj notatkę".  System wyświetla formularz dodawania notatki.  Użytkownik wprowadza treść notatki i zapisuje. |
| Ścieżka alternatywna | Jeśli użytkownik nie wprowadzi treści notatki, system wyświetli komunikat o błędzie. |
| Warunki końcowe | Nowa notatka zostaje dodana do listy, system utrzymuje stan strony "Notatki". |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 19 - Usuwanie notatki |
| Aktor | Użytkownik Aplikacji |
| Warunki początkowe | Użytkownik jest zalogowany do aplikacji i znajduje się na stronie "Notatki" z co najmniej jedną notatką. |
| Opis | Użytkownik wybiera notatkę do usunięcia.  Użytkownik potwierdza chęć usunięcia.  System usuwa notatkę z listy. |
| Ścieżka główna | Użytkownik otwiera stronę "Notatki".  Użytkownik wybiera notatkę do usunięcia.  System wyświetla potwierdzenie usunięcia.  Użytkownik potwierdza i system usuwa notatkę. |
| Ścieżka alternatywna | Jeśli użytkownik anuluje usunięcie, system nie usuwa notatki. |
| Warunki końcowe | Notatka zostaje usunięta, system utrzymuje stan strony "Notatki". |

|  |  |
| --- | --- |
| Nazwa | Scenariusz 2 - Filtrowanie notatek |
| Aktor | Użytkownik Aplikacji |
| Warunki początkowe | Użytkownik jest zalogowany do aplikacji i znajduje się na stronie "Notatki" z co najmniej jedną notatką. |
| Opis | Użytkownik korzysta z dostępnych opcji filtrowania.  System dostosowuje wyświetlane notatki zgodnie z wybranymi filtrami. |
| Ścieżka główna | Użytkownik otwiera stronę "Notatki".  Użytkownik wybiera kategorie lub wprowadza kryteria filtrowania.  System modyfikuje listę notatek zgodnie z wybranymi filtrami. |
| Ścieżka alternatywna | Jeśli nie ma pasujących notatek do wybranych filtrów, system informuje użytkownika o braku wyników. |
| Warunki końcowe | Lista notatek jest dostosowana do wybranych filtrów, system utrzymuje stan strony "Notatki". |

# Implementacja

## Front-END

* Za pomocą funkcji autoryzującej użytkownika tworzymy dwa różne widoki aplikacji dla zalogowanego i niezalogowanego użytkownika.
* Do przejścia pomiędzy widokami i stronami w aplikacji jest stworzona nawigacja której widok się zmienia w zależności od tego czy użytkownik jest zalogowany. W tym celu dołączony jest AuthServiceSetUser:

@inject LoginService login

W przypadku zalogowanego użytkownika mamy dostęp do wszystkich funkcji aplikacji np. zakładka toDo, Contact, Notes i inne. W przypadku niezalogowanego mamy tylko trzy opcje – logowanie, rejestracja i strona główna dla niezalogowanego użytkownika. Nawigację na przykładzie niezalogowanego użytkownika tworzymy za pomocą znaczników NavLink, ustawienia ścieżki do podstron oraz poprzez dodanie odpowiednich ikon odpowiednio wystylowanych:

<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">

<nav class="flex-column">

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="/" **Match**="NavLinkMatch.All">

<span class="oi oi-home" aria-hidden="true" style="color: blueviolet;"></span> Home

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="register">

<span class="oi oi-person" aria-hidden="true" style="color: blueviolet;"></span> Rejestracja

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="login" >

<span class="oi oi-account-login" aria-hidden="true" style="color: blueviolet;"></span> Logowanie

</**NavLink**>

</div>

</nav> </div>

Dla obsługi zakładki wyloguj w przypadku zalogowanego użytkownika dodajemy również funkcję logout:

private void logout()

{

var auth = ServiceProviders.GetRequiredService<AuthServiceSetUser>();

var authenticatedUser = new ClaimsPrincipal(new ClaimsIdentity());

auth.CurrentUser = authenticatedUser;

App.Current.MainPage = new MainPage();

}

Do obsługi przycisku rozwijającego nawigację mamy funkcję ToggleNavMenu:

private bool collapseNavMenu = true;

private string NavMenuCssClass => collapseNavMenu ? "collapse" : null;

private void ToggleNavMenu()

{

collapseNavMenu = !collapseNavMenu;

}

* Dodanie naszego logo:

@inject AuthServiceSetUser service

@inject ToDoService taskService

<div class="top-row ps-3 navbar navbar-dark">

<div class="container-fluid" style="display: flex; justify-content: space-between; align-items: center;">

<div style="display: flex; align-items: center;">

<img src="tologo.PNG" style="width: 50px; height: 50px; margin-left: 10px;" />

<a class="navbar-brand" href=""><p style="margin: 10px; ">Time Organizer</p></a>

</div>

<button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">

<span class="navbar-toggler-icon"></span>

</button>

</div>

* Ikonka z Instagramem z odnośnikiem na stronę instagram:

<div class="container-fluid">

<img src="instagram\_icon.png" @\* style="width: 50px; height: 50px; margin-left: 10px;" \*@ /><a href="https://www.instagram.com/">odwiedź nasz instagram</a> </div>

* Logowanie: do obsługi są dołączamy zależności z AuthServiceSetUser odpowiadającą za autoryzację użytkownika oraz LoginService odpowiadającego za ogólny proces logowania:

@inject LoginService login

@inject AuthServiceSetUser auth

Dołączamy warunek który sprawdzi czy użytkownik jest zalogowany czy nie. Jeśli nie – wyświetlane jest menu do logowania. Do obsługi inputów dodajemy właściwości @bind, natomiast do obsługi butonów – funkcję @onclick”@(() => ………)), która wywołuje komendę logowania zdefiniowaną w LoginService:

@if (!auth.CurrentUser.Identity.IsAuthenticated)

{

<div class="container">

<img src="user2.png" />

<h2> Logowanie </h2>

<p>Podaj login</p>

<input @bind="@login.LoginValue" />

<p>Podaj hasło</p>

<input @bind="@login.PassValue" type="password" />

<button class="btn btn-primary" @onclick="@(() => login.LoginCommand.Execute(true))"> Login </button>

<a href="/Register" class="button">Rejestracja</a>

<a href="/" class="button">WROC</a>

</div>

}

Jeśli tak – wyświetlamy inny div:

else

{

<div>jestes juz zalogowany!!</div>

}

* Podstrona rejestracji stworzona dokładnie na takiej samej zasadzie jak podstrona Login, za wyjątkiem tego, że posiada więcej pól input:

@page "/register"

@using timeorganizer.Services

@inject RegisterService registerservice

@inject AuthServiceSetUser auth

@if (!auth.CurrentUser.Identity.IsAuthenticated)

{

<div class="container">

<h2> Rejestracja</h2>

<p>Podaj Email</p>

<input @bind="@registerservice.Email" />

<p>Podaj Login</p>

<input @bind="@registerservice.Login" />

<p>Podaj hasło</p>

<input @bind="@registerservice.Password" />

<p>Powtórz hasło</p>

<input @bind="@registerservice.PasswordVerification" />

<button class="btn btn-primary" @onclick="@(() => registerservice.RegisterCommand.Execute(true))"> Zapisz</button>

<a href="/Login" class="button">Zaloguj</a>

</div>

}

else

{

<div>jestes juz zalogowany!!</div>

}

* Home – strona prezentująca spersonalizowany dashboard użytkownika, zapewniając szybki dostęp do istotnych informacji i zadań. Zawiera między innymi pasek postępu i aktualne zadania.

<div class="progress" role="progressbar" aria-label="Basic example" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100">

<div class="progress-bar bg-primary" style="width: @Statistic.ComplitedPrecent;">Ukończone zadania: @Statistic.Realized</div>

</div>

* ToDoPage. Zasada tworzenia widoków podstron pozostaje dokładnie taka sama – najpierw wstrzyskujemy serwisy do strony, które są niezbędne do jej obsługi. Później dodajemy pola i funkcję które odpowiadają za pobranie danych i poprawną obsługę programu. Na stronie ToDoPage mamy formukarz wykorzystujący @bind do powiązania pół formularza z odpowiednimi właściwościami w zewnętrznym serwisie obsługującym tę stronę. Dodana jest pętla foreach do iterację przez kolekcję zadań:

@foreach (var task in taskService.showTask.TasksCollection) {

<div class="d-flex flex-row border p-2" style="cursor: pointer;">

<p class="mr-3 mb-0">@task.Name</p>

<p class="mr-3 mb-0">@task.Description</p>

<p class="mr-3 mb-0">@task.Type</p>

<p class="mr-3 mb-0">@task.status</p>

<p class="mr-3 mb-0">@task.Termin</p>

<button @onclick="@(e => RedirectToDetails(@task.Id))">Szczegóły zadania</button>

</div>

}

Poniżej tworzony jest div z formularzem służącym do dodania nowego zadania:

<label class="form-label">Podaj tytuł swojego zadania</label>

<input class="form-control" @bind="@taskService.addTask.Name" placeholder="Np. Nauka Języka Angielskiego" />

<label class="form-label mt-3">Opisz swoje zadania</label>

<input class="form-control" @bind="@taskService.addTask.Description" placeholder="Określ konkrety, które chcesz zrealizować." />

<label class="form-label mt-3">Określ typ zadania</label>

<input class="form-control" @bind="@taskService.addTask.Typ" placeholder="Wprowadź typ zadania: home/work/hobby/other" />

<label class="form-label mt-3">Podaj Termin wykonania zadania</label>

<input class="datepicker" type="date" data-date-format="dd/mm/yyyy" @bind="@taskService.addTask.Termin">

<button class="btn btn-primary mt-3" @onclick="@(() => taskService.addTask.AddTaskCommand.Execute(true))">Utwórz główne zadanie</button>

* Do obsługi podstrony Contact potrzebne są tylko dwa pola i przycisk z obsługą powiązania danych w ContactService:

<h2> Contact Us</h2>

<p>Podaj email na którego chcesz otrzymać odpowiedź</p>

<input @bind="@contactService.EmailValue" placeholder="E-mail" />

<p>Opisz nam problem w aplikacji który napotkałeś</p>

<input @bind="@contactService.ReportValue" placeholder="Report" id="reportbutton" />

<button @onclick="@(() => contactService.MessageCommand.Execute(true))">Wyślij</button>

* Widok kalendarza jest stworzony za pomocą specjalnej pętli, która iteruje przez wiersze kalendarza, zakładając, że kalendarz będzie miał maksymalnie 6 wierszy. W głównej pętli jest umieszczona pętla wewnętrzna iterująca przez dni tygodnia, tworząc komórki w każdym wierszu. Dla każdej komórki kod oblicza datę „day” odnosząc się do pierwszego dnia miesiąca i uwzględniając korektę o jeden dzień (w systemie niedziela jest liczona jako pierwszy dzień). Wewnątrz komórki możemy dostosować odpowiednie klasy, do których można dodać style. Np. dni wolne wyświetlamy na czerwono. W przypadku kliknięcia komórki, wywoływana zostaje funkcja HandleCellClick.

@for (int i = 0; i < 6; i++) {

<tr>

@for (int j = 0; j < 7; j++) {

var firstDayOfMonth = new DateTime(selectedDate.Year, selectedDate.Month, 1);

var offset = (int)firstDayOfMonth.DayOfWeek - 1; // Korekta o jeden dzień

if (offset < 0) offset = 6;

var day = firstDayOfMonth.AddDays(i \* 7 + j - offset);

<td @((day.Month == selectedDate.Month && (j == 5 || j == 6)) ? "weekend" : "")" @onclick="() => HandleCellClick(day)">

@(day.Month == selectedDate.Month ? day.Day.ToString() : "")

</td> }

</tr>

}

* Notatki – wyświetlenie notatek odbywa się po spełnieniu warunku że kolekcja notatek nie jest pusta. Do prezentacji notatek wykorzystujemy tabele, w której definiujemy nagłówki: Tytuł, treść, data utworzenia oraz komórki w której za pomocą pętli foreach iterujemy po stworzonych notatkach i wyświetlamy ich zawartość:

@if (noteService.ShowNotesTitle.NotesColletion != null && noteService.ShowNotesTitle.NotesColletion.Any())

{

<table class="table">

<thead>

<tr>

<th>Tytuł</th>

<th>Treść</th>

<th>Data utworzenia</th>

<th>Akcje</th>

</tr>

</thead>

<tbody>

@foreach (var note in noteService.ShowNotesTitle.NotesColletion)

{

<tr>

<td>@note.Title</td>

<td>@note.Content</td>

<td>@note.Created</td>

<td><button @onclick="@(e=> DeleteNote(@note.Id))">Usuń</button></td>

</tr>

}

</tbody>

</table>

* Ustawienia – widok zawiera interfejs użytkownika, który umożliwia zarządzanie ustawieniami kont użytkownika oraz podgląd aktualnych danych użytkownika. Domyślny widok stanowią aktualne dane użytkownika oraz opcje ustawień dostępnych do edycji użytkownika.

@if(service.option == -1) // DOMYSLNY WIDOK

{

<div id="userinformation">

<h2>KONTO</h2>

<label>Email: <span>@loginService.Email</span> </label>

<label>Login: <span>@loginService.Login</span></label>

<label>Data Utw.: <span>@loginService.DateCreate</span></label>

<h2>USTAWIENIA</h2>

</div>

<button class="button-menu" @onclick="@(e => changeoption(1))">

<span class="oi oi-envelope-closed" aria-hidden="true"></span> Zmień Email

</button>

<button class="button-menu" @onclick="@(e => changeoption(2))">

<span class="oi oi-lock-locked" aria-hidden="true"></span>Zmień hasło

</button>

<button class="button-menu" @onclick="@(e => changeoption(3))">

<span class="oi oi-lock-locked" aria-hidden="true"></span>Zmień Email i Hasło

</button>

<button class="button-menu" @onclick="@(e => changeoption(4))">

<span class="oi oi-trash" aria-hidden="true"></span>Usuń konto

</button>

..//

}

Następnie są definiowane widoki dla poszczególnych opcji wybranych przez użytkownika. Np. w przypadku usunięcia konta – musimy podać hasło w wyświetlonym inpucie i kliknąć przycisk usuń.

@if (service.option == 4) // USUN KONTO

{

<h2>Wprowadź hasło</h2>

<input @bind="service.CurrentPassword" placeholder="Podaj haslo" type="password" />

<button type="button" class="btn btn-success btn-rounded btn-block btn-lg" data-mdb-ripple-color="#ffffff" style="background-color:blueviolet" @onclick="summit">Zatwierdz zmiany</button>

..//

}

* Statystyki – widok zawiera interfejs użytkownika umożliwiający śledzenie postępów użytkownika w wykonywaniu zadań. Umożliwia wyświetlanie statystyk miesiąca oraz tygodni takich jak liczba zadań ukończonych, liczba zadań ukończonych w całości, liczba zadań ukończonych nie w 100%, procent pełnej kompletności zadań.

<h2>Tygodnie roku</h2>

@foreach (var item in summaryService.TasksColWeek){

if ((year as dynamic).Year == (item as dynamic).Year){

<p class="mr-3 mb-0">Tydzień roku: @((item as dynamic).Week)</p>

<p class="mr-3 mb-0">@((item as dynamic).StartDate) - @((item as dynamic).EndDate)</p>

<p class="mr-3 mb-0">Ilość zadań ukończonych: @((item as dynamic).Count)</p>

<p class="mr-3 mb-0">Ilość zadań ukończonych w całości :@((item as dynamic).Count100Realized)</p>

<p class="mr-3 mb-0">Ilość zadań ukończonych nie w 100% :@((item as dynamic).CountNot100)</p>

<p class="mr-3 mb-0">Procent pełnej kompletności zadań:@((item as dynamic).Count100precent)%</p>

}}}

## Back-END

* Połączenie z bazą danych SQLite, polega na implementacji interfejsu IAsyncDiposable przydatnego do obsługi zwalniania zasobów asynchronicznych. Najpierw definiujemy stałą, która przechowuje nazwę bazy danych oraz definiujemy ścieżkę w której będzie przechowywana. Następnie tworzymy połączenie i na końcu zwalniamy zasoby.

public class DatabaseLogin : IAsyncDisposable

private const string DbName = "Timeorgranizer.db3";

private static string DbPath => Path.Combine(FileSystem.AppDataDirectory, DbName);

private SQLiteAsyncConnection \_connection;

private SQLiteAsyncConnection Database =>

(\_connection ??= new SQLiteAsyncConnection(DbPath,

SQLiteOpenFlags.Create | SQLiteOpenFlags.ReadWrite | SQLiteOpenFlags.SharedCache)); // polączenie do bazy

///….

public async ValueTask DisposeAsync() => await \_connection?.CloseAsync(); // zamkniecie polaczenia z baza

}

}

* Przykładowa składnia metod stworzonych w bazie danych:

private async Task CreateTableIfNotExists<TTable>() where TTable : class, new() //służy do tworzenia tabel z klas

{

await Database.CreateTableAsync<TTable>();

}

public async Task<bool> DeleteItemAsync<TTable>(TTable item) where TTable : class, new() //służy do usuwania po obiekcie elementów z bazy danych

{

await CreateTableIfNotExists<TTable>();

return await Database.DeleteAsync(item) > 0;

}

* Do późniejszego przechowywania danych w bazie konieczne są modele baz danych tzn. zasady przechowujące strukturę, typy danych oraz ich nazwy. W ten sposób definiujemy obiekty. W modelu jest możliwość przypisania domyślnej wartości do wybranego pola (np. w przypadku DateCreated):

[Table("Users")]

public class Users

{

[PrimaryKey,AutoIncrement]

public int Id { get; set; }

public string Email { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public bool RememberMe { get; set; } = false;

public string DataCreated { get; set; } = (DateTime.Now).ToLongDateString();

}

* Struktura MainPage w projekcie. Jest to strona stworzona na podstawie znaczników XAML. W naszym przypadku umożliwia integrację i definiuje element BlazorWebView, który umożliwia osadzenie zawartości Blazor wewnątrz widoku. Strona główna jest określona jako index.html. Wewnątrz pliku BlazorWebView definiujemy sekcję RootComponent jako główny komponent, który jest renderowany

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:timeorganizer"

x:Class="timeorganizer.MainPage"

BackgroundColor="{DynamicResource PageBackgroundColor}">

<BlazorWebView x:Name="blazorWebView" HostPage="wwwroot/index.html">

<BlazorWebView.RootComponents>

<RootComponent Selector="#app" ComponentType="{x:Type local:Main}" />

</BlazorWebView.RootComponents>

</BlazorWebView>

</ContentPage>

... //

* Obsługa metod i funkcji MainPage.Xaml.cs na przykładzie domyślnego konstruktora

namespace timeorganizer

{

public partial class MainPage : ContentPage

{

public MainPage()

{

InitializeComponent();

}

}

}

* Obsługa logowania na stronie odbywa się poprzez pobranie listy użytkowników z bazy danych za pomocą metody GetFilteredAsync oraz wybraniu użytkownika, którego login i hasło podane w polach będą się zgadzać z danymi w bazie danych. Za obsługę walidacji danych odpowiada funkcja IsValidEntry i wyświetla komunikat dot. odpowiedniego pola w którym są błędnie podane dane. W przypadku spełnienia warunków użytkownik dostaje powiadomienie o pomyślnym zalogowaniu i zostaje przekierowany na podstronę index.html .

private async void LoginUser(object obj)

{

if (!isValidEntry())

{

//w przypadku braku poprawnych danych - alert

return;

}

var users = await \_context.GetFileteredAsync<Users>(u => u.Login == LoginValue); //pobrana lista użytkowników zgodnych z LoginValue

var user = users.FirstOrDefault(); //pobieramy pierwszy element który pasuje

if (user != null && user.Password == PassValue)

{

await Application.Current.MainPage.DisplayAlert("Sukces", "Zalogowano pomyślnie", "OK");

App.Current.MainPage = new LoggedMainPage(); // zmiana domyslnego widoku na widok flyout

}

else

{

//komunikat o błędnych danych logowania

await Application.Current.MainPage.DisplayAlert("Błąd", "Niepoprawny login lub hasło", "OK");

}

}

private bool isValidEntry() //obsługa w przypadku braku podanych danych

{

... }}

* Do obsługi sesji jest stworzony model UserSessions

using SQLite;

namespace timeorganizer.DatabaseModels

{

[Table("UserSessions")]

public class UserSessions

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public int UserId { get; set; }

public string Token { get; set; }

public DateTime DateCreated { get; set; }

public DateTime ExpirationDate { get; set; }

}

}

Oraz serwis ActivityService, w którym mamy stworzone dwie metody: Jedna - do przedłużenia czasu ważności expirionDate w tabeli bazy danych po dokonaniu jakichkolwiek czynności w aplikacji oraz druga – do sprawdzania aktualnego czasu oraz czasu wylogowania, i w przypadku gdy ten warunek jest spełniony – użytkownik zostaje wylogowany. Funkcjami pomocniczymi w tym przypadku są metody getId i getToken do pobrania danych aktualnie zalogowanego użytkownika.

Funkcja przedłużająca sesję (ChangeExpirationDateCommand) uruchamia timer, który cyklicznie wywołuje funkcję CheckLastActivityVsExpirationDate() służącą do sprawdzenia expirationDate oraz autowylogowania w przypadku spełnienia warunku.

//Funkcja przedłużająca sesję

public async Task ChangeExpirationDateCommand()

{

timer.Elapsed += async (sender, e) => await CheckLastActivityVsExpirationDate();

timer.Enabled = true;

int userId = await Getid();

string userToken = await GetToken();

if (userId != 0)

{

var sessions = await \_context.GetFileteredAsync<UserSessions>(t => t.UserId == userId);

foreach (var session in sessions)

{

if (session.Token == userToken)

{

Debug.WriteLine("Pobrano sesję użytkownika: " + userId);

session.ExpirationDate = DateTime.Now.AddMinutes(3);

LastActivity = DateTime.Now;

await \_context.UpdateItemAsync<UserSessions>(session);

return;

}

}

return;

}

else

{

return;

}

}

//Funkcja sprawdzająca czy sesja nie wygasła

public async Task CheckLastActivityVsExpirationDate()

{

../

if (session != null)

{

if (session.Token == userToken)

{

DateTime timeNow = DateTime.Now;

if (timeNow >= session.ExpirationDate)

{

SecureStorage.Default.Remove("token");

timer.Enabled = false;

timer.Stop();

Device.BeginInvokeOnMainThread(() =>

{

App.Current.MainPage.DisplayAlert("Nastąpiło wylogowanie", "Twoja sesja wygasła. Zaloguj się ponownie.", "Ok");

App.Current.MainPage = new MainPage();

});

return;

}

return;

}

return;

}

}

* Rejestracja jest obsługiwana za pomocą RegisterService, który najpierw pobiera dane wprowadzone przez użytkownika w każdym z inputów oraz uruchamia metodę CreateUser po kliknięciu w przycisk „Zarejestruj się”. Metoda CreateUser sprawdza poprawność wprowadzonych danych za pomocą metody isValidInput, jeśli dane są poprawne za pomocą wykorzystania metody z DateBaseLogin o nazwie AddItemAsync program dodaje użytkownika do bazy.

private async void CreateUser()

{

if (!isValidInput()) //metoda do weryfikacji poprawności danych {

//wyswietli komunikat w zalezności od wystepującego błedu w podaniu danych

return;

}

var allUser = await \_context.GetFileteredAsync<Users>(user => user.Login == Login || user.Email == Email); //pobieramy wszystkie rekordy z users

..//

Users User = new()

{

Email = \_email,

Password = Helpers.Passwordhash.HashPassword(\_password),

Id = Id,

Login = \_login,

DataCreated = (DateTime.Now).ToLongDateString(),

RememberMe = false

};

await ExecuteAsync(async () =>

{

if (User.Id == 0)

{

await \_context.AddItemAsync<Users>(User);

await Application.Current.MainPage.DisplayAlert("Sukces", "Nowy użytkownik został dodany do bazy", "OK");

}

else

{ }

});

}

* Obsługa dodawania zadań odbywa się za pomocą serwisu AddTaskExtension. Stworzona klasa definiuje komendę AddTaskCommand, która jest wywoływana gdy użytkownik kliknie w przycisk dodaj zadanie. Po walidacji danych Metoda tworzy obiekt zadania Task oraz SubTask oraz dodaje je do bazy.

await \_context.AddItemAsync<Tasks>(Task);

TaskComponents SubTask = new()

* Obsługa sortowania zadań realizowana za pomocą FilterExtension. Najpierw klasa definiuje różne właściwości oraz kolekcje , które przechowują informację o filtrach do zastosowania podczas wyszukiwania zadań oraz kolekcję wyników. Komenda ShowTasks korzysta z metody FilterTasks i odpowiada za filtrowanie zadań. Uwzględniona jest walidacja danych po wywołaniu metody. Działanie metody opiera się na pobraniu danych spełniających określone kryteria oraz aktualizacji kolekcji TaskCollection i wyświetleniu wyników.

public ObservableCollection<Tasks> TasksCollection { get => \_filtered; set => \_filtered = value; } //kolekcja

public async void FilterTasks()

{

var activityservice = new ActivityService(); //inicjalizacja do późniejszego wywołania ChangeExpirationDate

if (\_userId == 0) { \_userId = await Getid(); }

await ExecuteAsync(async () =>

{

OnPropertyChanged(nameof(TasksCollection));

var filters = new Dictionary<string, object>

{

{ "Userid", \_userId }

};

if (Id !=0) filters.Add("Id", Id);

//…

TasksCollection = new ObservableCollection<Tasks>(await \_context.GetFileteredAsync<Tasks> (\_context.CreatePredicateToFiltred<Tasks>(filters)));

await Task.Delay(1000);

filters.Clear();

OnPropertyChanged(nameof(TasksCollection));

});

}

* Obsługa statusu realizacji danego zadania głównego odbywa się poprzez przeliczanie ile podzadań należy do danego zadania i ile z nich zostało ukończone.

public async Task UpdateRealized(int \_id) {

var activityservice = new ActivityService();

await ExecuteAsync(async () => {

var TCFin = await \_context.GetFileteredAsync<TaskComponents>(e => e.Status== "Ukończono" && e.TaskId==\_id);

var TCAll = await \_context.GetFileteredAsync<TaskComponents>(e => e.Status != "Rem" && e.TaskId == \_id);

Tasks Task = await \_context.GetItemByKeyAsync<Tasks>(\_id);

if (TCFin.Count()!=0)

Task.RealizedPercent = (int)(((double)TCFin.Count() / TCAll.Count()) \* 100);

else

Task.RealizedPercent = 0;

await \_context.UpdateItemAsync(Task);

});

await activityservice.ChangeExpirationDateCommand();

}

* Podczas oznaczania zadania głównego jako ukończone odbywa się sprawdzenie czy zadanie posiadało jakieś podzadania I jeśli nie tu nastąpi ukończenie zadania w 100%

public async Task CheckFinComp(Tasks task) {

var activityservice = new ActivityService();

await ExecuteAsync(async () => {

var row= await \_context.GetFileteredAsync<TaskComponents>(e => e.Status != "Rem" && e.TaskId == task.Id);

bool confirmation = await App.Current.MainPage.DisplayAlert("Potwierdzenie", "Ukończenie tego zadania będzię się wiązało z brakiem możliwości pozostałych podzadań", "Ok", "Anuluj");

if (confirmation) {

if (!row.Any()) task.RealizedPercent = 100;

task.Status = "Ukończono";

task.UkonczonoDateTime = DateTime.Now.Date;

await \_context.UpdateItemAsync(task);

await App.Current.MainPage.DisplayAlert("Sukcess", "Zadanie oznaczone jako ukończone", "Ok");

}

});

await activityservice.ChangeExpirationDateCommand();

}

* Obsługa strony kontakt jest zdefiniowana w ContactService. Po walidacji i poprawnie wpisanych danych funkcja SendMessage zapisuje do bazy dane użytkownika oraz treść zgłoszenia.

private async void SendMessage(object obj)

{

if (!isValidInput())

{

//wyswietli komunikat w zalezności od wystepującego błedu w podaniu danych

return;

}

if (\_userId == 0) \_userId = await Getid();

try

{

Message newMessage = new()

{

Email = \_emailValue,

UserId = \_userId,

MessageText = \_reportValue,

DateCreate = DateTime.Now.ToString(),

Status = "new"

};

Application.Current.MainPage.DisplayAlert("Thank You", "Report was sent", "OK");

}

* Autoryzacja: do obsługi autoryzacji wykorzystywany jest obiekt AuthenticationStateProvider wraz z AuthenticationState zawierającym informacje o autoryzowanym użytkowniku.

Autoryzacja odbywa się na poziomie dostępu do poszczególnych stron parametrem auth.CurrentUser.Identity.IsAuthenticated dostarczonym przez obiekt typu AuthenticationState. Poniżej implementacja autoryzowania użytkownika:

public class AuthServiceSetUser

{

public event Action<ClaimsPrincipal>? UserChanged;

private ClaimsPrincipal? currentUser;

public ClaimsPrincipal CurrentUser

{

get { return currentUser ?? new ClaimsPrincipal(new ClaimsIdentity()); }

set

{

currentUser = value;

if (UserChanged is not null)

{

UserChanged(currentUser);

}

}

}

Wywoływana jest podczas logowania.

var auth = ServiceProviders.GetRequiredService<AuthServiceSetUser>();

var authenticatedUser = new ClaimsPrincipal(new ClaimsIdentity(new Claim[] {

new Claim(ClaimTypes.Role, "Administrator"),

new Claim(ClaimTypes.NameIdentifier, LoginValue)}, "Basic"));

auth.CurrentUser = authenticatedUser; // przypisanie stworzonej torzstamości do zmiennej w funkcji aby była dostępna globalnie.

Aby wyczyścić autoryzacje użytkownika korzystamy z przypisania pustego obiektu do currentUser:

var authenticatedUser = new ClaimsPrincipal(new ClaimsIdentity());

* Obsługa aktualizacji danych na przykładzie strony Setting. Podstrona jest obsługiwana na podstawie SettingService, zawiera walidację danych oraz służy do aktualizacji już zawartych w bazie danych. Przykład aktualizacji email i hasła w aplikacji wygląda w ten sposób:

private async Task ChangeAllCommand\_execute()

{

await ExecuteAsync(async () =>

{

if (\_id == 0) \_id = await Getid();

Users user = new Users();

user = await \_context.GetItemByKeyAsync<Users>(\_id);

if (await validatepassword() && await validateEmail())

{

user.Password = \_password;

user.Email = \_email;

user.DataModified = (DateTime.Now).ToLongDateString();

await \_context.UpdateItemAsync(user);

..//

await App.Current.MainPage.DisplayAlert("Success", "Dane zostały poprawnie zmienione", "Ok");

option = -1;

\_email = null;

\_password = null;

\_passwordconfirm = null;

\_currentpassword = null;

}

else

{

await App.Current.MainPage.DisplayAlert("Błąd", "Niepoprawy format email", "Ok");

}

}

Pomimo tego mamy możliwość usunięcia własnego konta, którą realizuje funkcja DeleteAccountCommand():

private async Task DeleteAccountCommand()

{

bool answer = await App.Current.MainPage.DisplayAlert("Usuwanie Konta", "Czy chcesz usunąć swoje konto?", "Tak", "Nie");

if (answer)

{

await ExecuteAsync(async () =>

{

if (\_id == 0) \_id = await Getid();

Users user = new Users();

user = await \_context.GetItemByKeyAsync<Users>(\_id);

if (\_currentpassword == user.Password)

{

await \_context.DeleteItemAsync<Users>(user);

}

SecureStorage.RemoveAll();

option = -1;

App.Current.MainPage = new MainPage();

\_currentpassword = null;

})}}

* Obsługa Kalendarza: po odczytaniu przekazanej zmiennej wybranej daty, wywoływana zostaje funkcja ReadTask, która tworzy filtr do zadań w zależności od użytkownika i terminu. Za pomocą metody GetFilterdeAsync odczytujemy dane spełniające kryteria i aktualizujemy kolekcję.

//zmienna z aktualną datą

private DateTime \_dateValue;

public DateTime DateValue

{

get => \_dateValue;

set

{

\_dateValue = value;

ReadTasks();

}

}

//nowa kolekcja obiektów

private ObservableCollection<Tasks> \_collection = new();

public ObservableCollection<Tasks> TasksCollection

{

get => \_collection;

set => SetProperty(ref \_collection, value);

}

..//

//metoda + dodanie filtrów + aktualizacja kolekcji na podstawie filtrów

public async Task ReadTasks()

{

\_userId = await Getid();

var filters1 = new Dictionary<object, string>

{

{ \_userId,"Equal" }

};

var filters = new Dictionary<string, object>

{

{"Userid",\_userId }

};

if (DateValue != default)

{

filters.Add("Termin", DateValue.ToString("dd.MM.yyyy"));

filters1.Add(DateValue.ToString("dd.MM.yyyy"), "Equal");

}

\_collection = new ObservableCollection<Tasks>(await \_context.GetFileteredAsync<Tasks>(\_context.CreatePredicateToFiltred<Tasks>(filters, filters1)));

filters.Clear(); //czyszczenia filtrów w celu poprawnego działania funkcji }

* Obsługa notatek: za pomocą asynchronicznej metody AddNote w serwisie AddNotesExtension.cs tworzymy nowy obiekt Notes, na podstawie wprowadzonych w polach danych, które później są sprawdzane, jeśli warunki są spełnione – notatka zostaje dodana do bazy.

Notes note = new()

{

Title = \_title,

Content = \_content,

Created = DateTime.Now.ToLongDateString(),

UserId = \_userId,

LastUpdated = null

};

await \_context.AddItemAsync<Notes>(note);

await App.Current.MainPage.DisplayAlert("Succes", "Dodano notatkę do bazy", "Ok");

Do wyświetlenia listy notatek jest wykorzystywana funkcja GetNotes();

Za pomocą asynchronicznej metody DeleteNote usuwamy notatkę na podstawie przekazanego identyfikatora.

private async void DeleteNote(int id)

{

try

{

await context.DeleteItemByKeyAsync<Notes>(id);

// Aktualizacja listy notatek

await noteService.ShowNotesTitle.GetNotes();

StateHasChanged();// Pobiera notatki i aktualizuje NotesList

}

catch (Exception ex)

{

// Obsługa błędu

Console.WriteLine($"Błąd podczas usuwania notatki: {ex.Message}");

}

}

* Obsługa statystyk opiera się na zbieraniu danych o zadaniach użytkownika i obliczeniu różnych wskaźników statystycznych za pomocą metody GetUserRealizedTasks, która wykorzystuje kolekcję TaskList , oblicza łączną liczbę zadań, liczbę zrealizowanych zadań oraz liczbę zadań w trakcie realizacji.

public async Task GetUserRealizedTasks()

{

int \_userid = await Getid();

if (\_userid == 0) { return; }

await ExecuteAsync(async () =>

{

TaskList = new ObservableCollection<Tasks>(await \_context.GetFileteredAsync<Tasks>(t => t.UserId == \_userid));

int \_all = TaskList.Where(t => t.Status != "Rem").Count();

int \_realized = TaskList.Where(t => t.Status == "Ukończono").Count();

int \_inprogress = TaskList.Where(t => t.RealizedPercent > 0 && t.RealizedPercent < 100 && t.Status != "Rem").Count();

Total = \_all;

var tmp = (double)\_realized / \_all \* 100;

ComplitedPrecent = Convert.ToInt32(tmp).ToString() + '%';

Inprogress = \_inprogress;

Realized = \_realized;

Active = \_all - \_realized - \_inprogress;

});

return;

}

# Podsumowanie prac (Czas)

**Obraz zawierający tekst, zrzut ekranu, Równolegle, numer

Opis wygenerowany automatycznie**

# Testowanie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, komputer

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, komputer

Opis wygenerowany automatycznieSystematycznie przeprowadzaliśmy testy na obecność błędów i obsługę wyjątków. Aktywnie poszukiwaliśmy potencjalnych problemów, używając różnych danych wejściowych i scenariuszy użytkowania.

W celu zapewnienia poprawności działania poszczególnych jednostek kodu, zastosowaliśmy testy jednostkowe. Skupiliśmy się na testowaniu funkcji, klas i metod oddzielnie.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, multimedia

Opis wygenerowany automatycznie

Przykład testowania poprawności działania funkcji na podstawie komunikatów w oknie danych wyjściowych i poleceń systemowych. Przeprowadziliśmy testy używając komend systemowych do monitorowania i zarządzania zasobami aplikacji oraz zmiennymi przechowywanymi w pamięci.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Sprawdziliśmy, czy poszczególne komponenty aplikacji współpracują ze sobą poprawnie. Wykonywaliśmy testy integracyjne, które obejmowały interakcje między różnymi modułami i sprawdzanie, czy komunikacja między nimi przebiega bez problemów.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Skoncentrowaliśmy się na testowaniu funkcjonalności dostarczanych przez aplikację. Każda funkcja została dokładnie przetestowana pod kątem zgodności z założeniami biznesowymi i oczekiwaniami użytkownika.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

Systematycznie analizowaliśmy kod źródłowy pod kątem zgodności z najlepszymi praktykami programistycznymi, czytelności, efektywności i bezpieczeństwa. Podsumowując, kompleksowe podejście do testowania obejmujące różne rodzaje testów oraz analizę kodu źródłowego przyczyniło się do stworzenia stabilnej, bezpiecznej i efektywnej aplikacji.

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

# Podręcznik użytkownika

**Logowanie + Menu**

Jeśli korzystamy z aplikacji po raz pierwszy – należy się zarejestrować, na stronę rejestracji zostaniemy przekierowani po kliknięciu w przycisk SIGN UP (rejestracja) na stronie głównej. Po wpisaniu niezbędnych danych i utworzeniu konta – możemy się zalogować po kliknięciu w przycisk – LOGIN (logowanie) na tej samej stronie głównej. Po zalogowaniu pojawi się strona home. Po kliknięciu w przycisk w górnym prawym rogu pojawi się menu z dostępnymi opcjami dla użytkownika.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, komputer

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznie

**Zadania**

Po przejściu na podstronę Zadania – w prawym górnym rogu pojawi się przycisk, który służy do dodawania nowych zadań (wymagane pola to: tytuł, opis, typ zadania). Użytkownik może zdefiniować własne typy zadań w sekcji filtrowanie (przycisk: dodaj własny typ zadań). Po wpisaniu wszystkich danych dodajemy zadanie poprzez przycisk – Utwórz zadanie.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Każde zassanie można przeglądać, edytować, zmienić status poprzez przycisk ukończ zadanie. Wszystkie te opcje są dostępne po kliknięciu w przycisk Szczegóły zadania pod każdym z utworzonych zadań.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

**Kontakt**

Po przejściu na podstronę kontakt każdy może zgłosić problem i przesłać go do moderatorów wpisując w pola informacje, których dotyczy zgłoszenie oraz podając maila do kontaktu. Także po kliknięciu w ikonkę instagramu można przejść na naszą stronę z aktualnymi informacjami.

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

**Kalendarz**

Po przejściu na podstronę kalendarz możemy prześledzić zadania przypisane do konkretnej daty poprzez kliknięcie w wybrany dzień. Na górze mamy przyciski Prev Month i Next Month do zmiany bieżącego miesiąca.

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

**Ustawienia**

Podstrona umożliwia podgląd wybranych danych użytkownika oraz ich zmianę. W przypadku wybrania poszczególnych opcji musimy podać hasło do zatwierdzenia i np. nowy mail. Po wpisaniu i poprawnej walidacji danych – zostają one zmienione, a informację o koncie zaktualizowane.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

**Statystyki**

Podstrona przedstawia dane dot. zrealizowanych zadań i niektórych statystyk uwzględniając całość zadań, ukończonych zadań i zadań w trakcie realizacji.

**i**Obraz zawierający tekst, zrzut ekranu, Czcionka, Strona internetowa

Opis wygenerowany automatycznie

# Podsumowanie projektu. Wnioski

* Stworzona aplikacja nie tylko spełnia bazowe wymagania, ale dostarcza użytkownikowi kompleksowe narzędzie do zarządzania czasem, zadaniami, statystykami, ustawieniami i notatkami.
* System obsługi błędów w aplikacji został dobrze zaimplementowany, dostarczając użytkownikowi wystarczające informacje zwrotne. Kompletna informacja zwrotna stanowi kluczową cechę w przypadku napotkania problemów.
* W kontekście dużej ilości przetwarzanych danych, kluczowym aspektem projektu jest skrupulatna walidacja danych. Ta procedura została uwzględniona w kodzie źródłowym, co wpływa na integralność i bezpieczeństwo danych.
* Zastosowanie technologii Brazor pozwoliło na stworzenie intuicyjnego interfejsu użytkownika, co wpływa pozytywnie na doświadczenie użytkownika.
* Wprowadzenie funkcji interakcji społecznościowej oraz możliwość bezpośredniego kontaktu z moderatorami pozytywnie wpływają na relacje z użytkownikami i budują zaufanie do aplikacji.
* Wybór lokalnej bazy danych SQLite jest odpowiada skali projektu, co sprzyja efektywnemu przechowywaniu danych.
* Wykorzystanie Dependency Injection do utrzymania zależności między klasami jest dobrym praktyką, zwłaszcza w projektach o rozbudowanej strukturze.
* Sumaryczny czas poświęcony na projekt, wynoszący 278 godzin, jest znaczący i świadczy o zaangażowaniu zespołu w proces tworzenia aplikacji.
* Częste sprinty, ciągła komunikacja zespołu oraz analiza problemów i możliwości są kluczowymi aspektami, które wpływają na efektywność pracy i postęp projektu.
* Stworzenie projektu przyczyniło się do doskonalenia umiejętności programistycznych każdego z członków zespołu, co stanowi wartość dodaną dla każdego indywidualnie.

# Bibliografia

* <https://learn.microsoft.com/pl-pl/dotnet/maui/what-is-maui>
* <https://www.youtube.com/watch?v=LrZwd-f0M4I>
* <https://www.infoq.com/maui/>
* <https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor>
* <https://lovepik.com/image-401579738/ui-design-purple-gradient-style-app-landing-page.html>
* <https://www.sqlite.org/docs.html>
* <https://learn.microsoft.com/en-us/samples/dotnet/maui-samples/database-sqlite/>
* <https://learn.microsoft.com/en-us/dotnet/maui/data-cloud/database-sqlite?view=net-maui-8.0>