

Projekt zaliczeniowy



Programowanie obiektowe
Rok akademicki 2025/2026

Autorzy:

Marcin Różalski
Viktoria Toman
Natalia Grabowska

System badania skłonności finansowych pod wpływem ryzyka

Nasz projekt to aplikacja okienkowa, która ma za zadanie przeprowadzić interaktywne badanie z zakresu ekonomii behawioralnej. Program czyni to poprzez zebranie danych uczestników, następnie zadanie pytań pod presją czasu. Głównym zadaniem aplikacji jest symulacja wyborów finansowych, w których uczestnik musi zdecydować między pewnym zyskiem/stratą a opcjami ryzykownymi.

Podział ról

Marcin Różalski: Implementacja kodu i logiki działania aplikacji.
Viktoria Toman: Interfejs graficzny
Natalia Grabowska: Dokumentacja

Analiza klas

1. Klasy z folderu *MainWindow*

Ta część odpowiada za komunikację z użytkownikiem, inicjalizację aplikacji oraz zbieranie wstępnych danych niezbędnych do przeprowadzenia eksperymentu.

1.1 Klasa *App*

Opis: Jest to klasa bazowa aplikacji, dziedzicząca po bazowej klasie *Application*. Jest automatycznie generowana w projektach WPF. W tej klasie nie ma pól ani metod, pełni ona rolę strukturalną i konfiguracyjną.

Rola w projekcie: Odpowiada za ogólny cykl życia aplikacji, czyli uruchomienie, zamknięcie czy ładowanie zasobów.

Modyfikatory dostępu: Klasa *App* została oznaczona jako *public*, ponieważ framework WPF musi mieć możliwość jej utworzenia i zarządzania nią w trakcie uruchamiania programu. Ma ona modyfikator typu *partial*, ponieważ składa się ona zarówno z kodu napisanego przez autora (*App.xaml.cs*), jak i kodu wygenerowanego na podstawie *App.xaml*.

1.2 Klasa *MainWindow*

Opis: Klasa ta jest głównym oknem startowym aplikacji dziedziczącym po *Window*, w którym użytkownik wprowadza swoje dane takie jak: wiek, płeć, poziom stresu, kierunek, który studiuje oraz poziom zamożności.

Rola w projekcie: Klasa *MainWindow* jest bardzo istotna w projekcie, ponieważ to ona nawiązuje pierwszą interakcję z uczestnikiem. Co ważniejsze, zbiera ona dane użytkownika, inicjuje obiekt klasy *Participant*, a po poprawnym podaniu informacji przez osobę badaną zamyka okno startowe i przechodzi do kolejnego okna, czyli *QuestionWindow*. Dzięki temu aplikacja ma strukturę etapową.

Konstruktory, Metody i Modyfikatory dostępu: Klasa jest *publiczna*, ponieważ technologia WPF musi mieć możliwość utworzenia i wyświetlenia okna z poziomu aplikacji. Jest też *partial* zgodnie z mechanizmem kompilacji.

Konstruktor *MainWindow* zawiera metodę *InitializeComponent*, generowaną przez WPF na podstawie pliku *xaml*. Odpowiada ona za załadowanie elementów jak *textboxy* lub przyciski.

Konstruktor *MainWindow* musi być *publiczny*, ponieważ okno jest tworzone przez mechanizm WPF podczas uruchamiania aplikacji. Gdyby konstruktor miał modyfikator *private* lub *protected*, framework nie miałby możliwości utworzenia instancji okna, co spowodowałoby błąd uruchomienia aplikacji.

Metoda obsługi zdarzenia *StartClick* uruchamia się poprzez kliknięcie przycisku "start" w UI. Jest ona *prywatna*, ponieważ jest to funkcja obsługująca zdarzenie interfejsu użytkownika i nie powinna być wywoływana spoza klasy *MainWindow*.

Zastosowaliśmy również blok obsługi wyjątków (try/catch), aby wyłapać potencjalne błędy użytkownika jak pozostawienie pustych pól - w takim przypadku użytkownik dostaje powiadomienie o błędzie.

2. Klasy z folderu projekt_C#

Ta część skupia się na logice działania aplikacji oraz strukturach danych wykorzystywane w eksperymencie. Odpowiada za przetwarzanie odpowiedzi użytkownika, naliczanie wyniku oraz zapis rezultatów.

2.1 Klasa Participant i ParticipantException

Opis: Klasa Participant reprezentuje uczestnika badania - przechowuje wszystkie podane wcześniej dane oraz wynik eksperymentu. Można określić ją centralną klasą modelu danych. ParticipantException jest klasą wyjątku dziedziczącą po Exception stworzoną na wypadek błędów walidacji.

Rola w projekcie: Zadaniem klasy Participant jest przechowanie danych użytkownika oraz jego identyfikatora Uuid, za to własna klasa ParticipantException ma za zadanie poprawiać czytelność kodu oraz sygnalizować ewentualny błąd uczestnika podczas wprowadzania danych.

Konstruktory i Modyfikatory dostępu: Pole `_age` jest prywatne, aby uniemożliwić bezpośrednią zmianę wieku bez kontroli poprawności danych, co zapewnia enkapsulację. Dostęp do wieku odbywa się przez właściwość `Age`, która posiada walidację, w przypadku wybrania wartości spoza zakresu 18-120 wyrzucany jest własny wyjątek `ParticipantException`. Pozostałe właściwości (`Gender`, `Kierunek`, `Stres`, `Zamożność`) są publiczne, ponieważ muszą być łatwo ustawiane z poziomu interfejsu użytkownika. Właściwość `Capital` przechowuje aktualny kapitał uczestnika i jest inicjalizowana domyślnie wartością 1000, co stanowi punkt startowy eksperymentu. Dodatkowo uczestnik otrzymuje unikalny identyfikator `Uuid` typu `Guid`, który jest ustawiany automatycznie przy tworzeniu obiektu i ma tylko getter, aby nie można było go zmieniać w trakcie działania programu. Modyfikator zarówno w konstruktorze jak i klasie `ParticipantException` jest `public`, aby można było tworzyć wyjątek z dowolnego miejsca w aplikacji oraz aby inne klasy mogły go przechwytywać i odpowiednio obsłużyć.

2.2 Klasa Question

Opis: Jest to klasa abstrakcyjna, która stanowi szablon dla pytań w badaniu.

Rola w projekcie: Definiuje wspólny interfejs dla każdego pytania, który zawiera `id`, `tekst`, `limit czasu` oraz `opcje odpowiedzi`.

Konstruktory, Metody i Modyfikatory dostępu: Klasa `Question` zawiera publiczne właściwości `Id`, `Text` i `TimeLimit`, które opisują odpowiednio identyfikator pytania, jego treść oraz limit czasu na odpowiedź. Ponadto posiada właściwość `Options`, czyli listę dostępnych odpowiedzi. Modyfikator konstruktora jest `protected`, ponieważ nie tworzymy jej bezpośrednio, tylko korzystają z niej klasy dziedziczące np. `RiskQuestion`.

Metoda wirtualna `GetQuestionType` zwraca nazwę typu pytania. Domyślnie zwracana jest wartość „Podstawowe pytanie”, natomiast w klasach dziedziczących metoda zostanie ona nadpisana, aby zwracać bardziej konkretny typ.

Metoda abstrakcyjna `ApplyEffect`, która wymusza na klasach pochodnych zdefiniowanie własnej logiki działania pytania. W praktyce oznacza to, że po udzieleniu odpowiedzi program wykonuje odpowiednie działania takie jak zmiana kapitału uczestnika na podstawie wybranej odpowiedzi.

2.3 Klasa *QuestionOption*

Opis: Jest to model danych dla pojedynczej opcji odpowiedzi.

Rola w projekcie: Przechowuje treść odpowiedzi wyświetlaną użytkownikowi oraz wartość `CapitalChange`, która określa, jak zmieni się kapitał uczestnika po wybraniu danej opcji.

Modyfikatory dostępu: Właściwości są publiczne, ponieważ interfejs użytkownika musi być w stanie je wyświetlić, a inne klasy odczytać.

2.4 Klasa *RiskQuestion*

Opis: Jest to klasa dziedzicząca po `Question`, która reprezentuje pytanie dotyczące ryzyka finansowego.

Rola w projekcie: Klasa `RiskQuestion` implementuje konkretne pytanie i na podstawie wybranej odpowiedzi oblicza zmianę kapitału.

Konstruktory, Metody i Modyfikatory dostępu: Konstruktor klasy `RiskQuestion` musi być publiczny, ponieważ pytania są tworzone w innych częściach programu. Metoda `GetQuestionType` została nadpisana, aby zwracać konkretny typ pytania o ryzyko, który jest później pokazywany użytkownikowi. Nadpisana została również metoda `ApplyEffect`, która za zadanie ma odnaleźć wybraną opcję odpowiedzi i na jej podstawie aktualizować kapitał uczestnika.

2.5 Klasa *Storage*

Opis: Ta klasa służy do zapisu danych do pliku w formacie JSON.

Rola w projekcie: Dzięki klasie `Storage` przechowujemy dane również poza aplikacją.

Modyfikatory: Klasa jest statyczna, ponieważ służy jedynie jako narzędzie zapisu, a metoda `Save` jest publiczna, bo może być wywoływana z różnych części programu.

3. Klasy folderu *QuestionWindow*

Ta część zawiera część aplikacji odpowiedzialną za wyświetlanie kolejnych pytań, kontrolę czasu odpowiedzi oraz podsumowanie i zapis wyniku po zakończeniu badania.

3.1 Klasa App

Wszystko analogicznie jak w przypadku klasy App w folderze MainWindow.

3.2 Klasa QuestionWindow

UWAGA: plik nazywa się MainWindow.xaml.cs, ale klasa w środku to QuestionWindow.

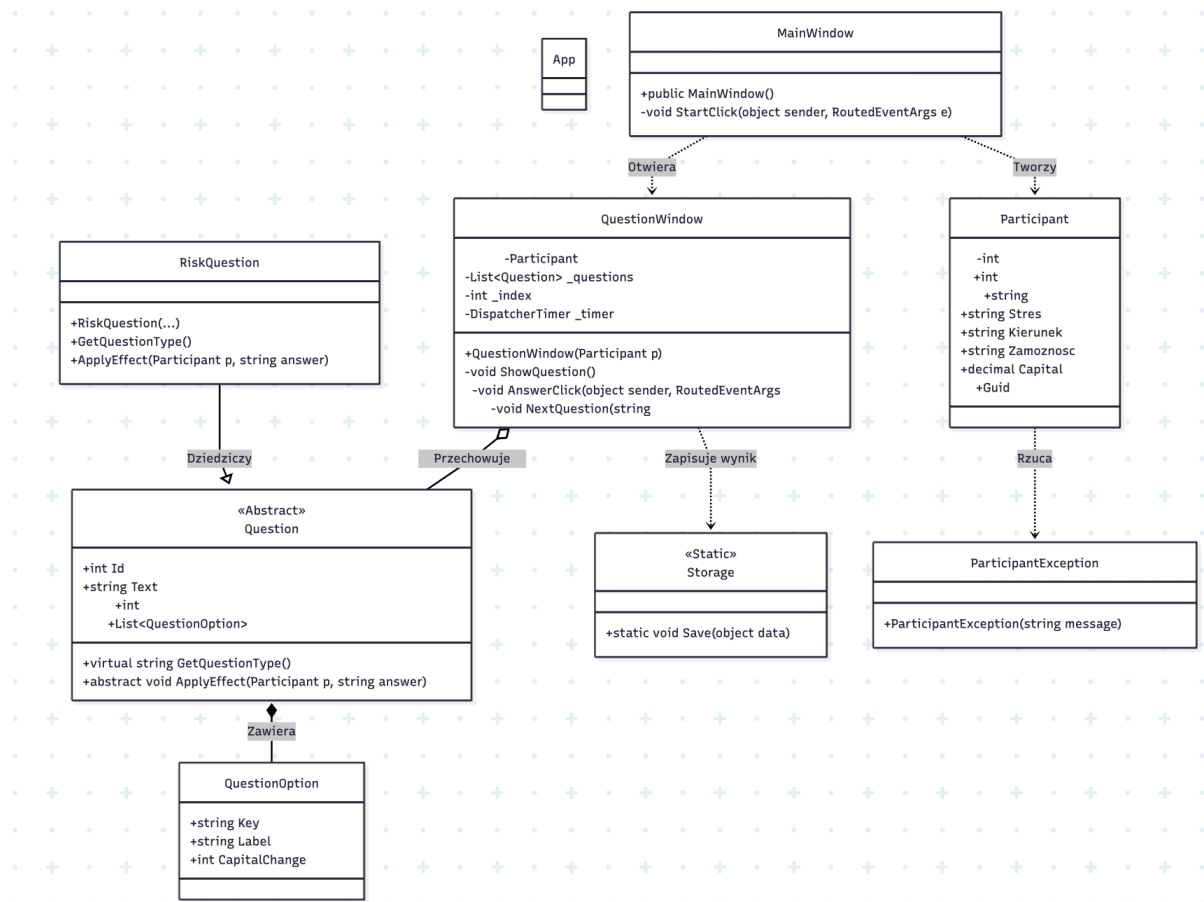
Opis: Klasa ta jest oknem aplikacji dziedziczącym po Window, w której dla użytkownika wyświetlane są pytania zamknięte.

Rola w projekcie: Klasa ta ponownie nawiązuje interakcje z uczestnikiem badania poprzez zadawanie mu pytań w odpowiedniej kolejności, ale również odlicza czas, zmienia wartość kapitału w zależności od wyboru oraz kończy cały eksperyment zapisując wynik do pliku JSON.

Konstruktory, Metody i Modyfikatory dostępu: Klasa QuestionWindow została oznaczona jako public i partial z tych samych powodów co klasa MainWindow. Metoda ShowQuestion jest oznaczona jako private, ponieważ powinna być wykorzystywana tylko w tym oknie. Ma ona za zadanie sprawdzać czy uczestnik badania odpowiedział na wszystkie pytania - jeśli tak, wtedy wynik zostaje zapisany do pliku, eksperyment kończy się, a aplikacja zostaje zamknięta. Jeśli jednak badany nie odpowiedział na wszystkie pytania, metoda pobiera następne pytanie, wybiera jego typ, wyświetla treść oraz ustawia stoper na czas jego odpowiedzi. Metoda AnswerClick również jest prywatna, ponieważ odzwierciedla ona jedynie reakcję na kliknięcie odpowiedniego przycisku w interfejsie użytkownika. Następnie pobiera ona wartość z właściwości Tag danej odpowiedzi, a następnie przekazuje ją do metody NextQuestion.

Metoda NextQuestion również jest prywatna, ponieważ działa ona tylko wewnątrz przebiegu badania. Odpowiada ona za sprawdzenie przypadku, gdy uczestnik nie zmieścił się w czasie i ponosi karę równą 100 jednostek kapitału. Jeśli uczestnik wybrał którąś z opcji wtedy wywoływana jest logika aktualnego pytania poprzez metodę ApplyEffect, która na podstawie odpowiedzi modyfikuje kapitał uczestnika. Następnie zwiększany jest indeks, aby przejść do kolejnego pytania, a na końcu ponownie wywoływana jest metoda ShowQuestion, która wyświetla następne pytanie lub kończy eksperyment.

Diagram klas



Obraz 1 - Diagram klas

Opis funkcjonalności

Rozpoczęcie eksperymentu i identyfikacja uczestnika

Opis: Pierwszym etapem działania aplikacji jest zebranie danych od użytkownika, a realizowane to w oknie startowym, gdzie uczestnik wypełnia formularz zawierający pola takie jak wiek, płeć, poziom stresu, kierunek studiów oraz ocena zamożności. System dba o poprawność wprowadzonych danych i w przypadku błędu informuje użytkownika odpowiednim komunikatem. Po poprawnym uzupełnieniu formularza tworzona jest baza klasy Participant, która otrzymuje identyfikator, a aplikacja przechodzi do głównej fazy eksperymentu.

Klasy realizujące: MainWindow odpowiada za interfejs formularza, pobieranie danych z pól tekstowych, obsługę błędów oraz utworzenie obiektu uczestnika i przejście do okna z pytaniami.

Participant przechowuje dane uczestnika.

ParticipantException przeprowadza walidację wieku

Wiek:

25

Płeć (M/K):

M

Kierunek:

Informatyka

Jak w skali 1-6 radzisz sobie ze stresem:

0

Jak w skali 1-4 poradziłbys sobie obecnie z nagłym wydatkiem 2000zł?:

0

Zacznij Badanie

Obraz 2 - Okno startowe aplikacji - formularz wprowadzania danych uczestników.

Przeprowadzenie eksperymentu w formie serii pytań

Opis: Po rozpoczęciu badania aplikacja przechodzi do okna z pytaniami, w którym uczestnik musi podjąć pewne decyzje finansowe. Pytania są wyświetlane w ustalonej kolejności, a użytkownik wybiera jedną z dwóch opcji odpowiedzi.

Klasy realizujące: QuestionWindow odpowiada za wyświetlanie kolejnych pytań oraz kontrolowanie kolejności.

Question definiuje wspólną strukturę pytań.

RiskQuestion reprezentuje konkretny typ pytania.

Pytanie o ryzyko

Wolisz na 80% zyskać 700zł czy na 100% zyskać 500zł?

na 80% zyskać 700

na 100% zyskać 500

Czas: 9s | Kapitał: 1000

Obraz 3 - Okno przykładowego pytania o ryzyko z aktualnym kapitałem oraz limitem czasu.

Obliczanie wyniku na podstawie odpowiedzi uczestnika

Opis: Podczas eksperymentu odpowiedzi uczestnika wpływają na jego aktualny wynik w postaci kapitału. Każda opcja odpowiedzi ma przypisaną wartość zmiany kapitału. Po wybraniu odpowiedzi program na bieżąco aktualizuje stan konta uczestnika.

Klasy realizujące: RiskQuestion zawiera metodę zmiany kapitału ApplyEffect na podstawie wybranej odpowiedzi.

QuestionOption przechowuje wartość CapitalChange.

Participant zawiera aktualny kapitał uczestnika.

Monitorowanie czasu uczestnika

Opis: Dla każdego pytania inicjowany jest licznik sekund. Jeśli zmienna _seconds spadnie do zera, jest wywoływana metoda NextQuestion, co skutkuje odjęciem 100 jednostek z kapitału uczestnika.

Klasy realizujące: QuestionWindow odlicza czas za pomocą DispatcherTime.

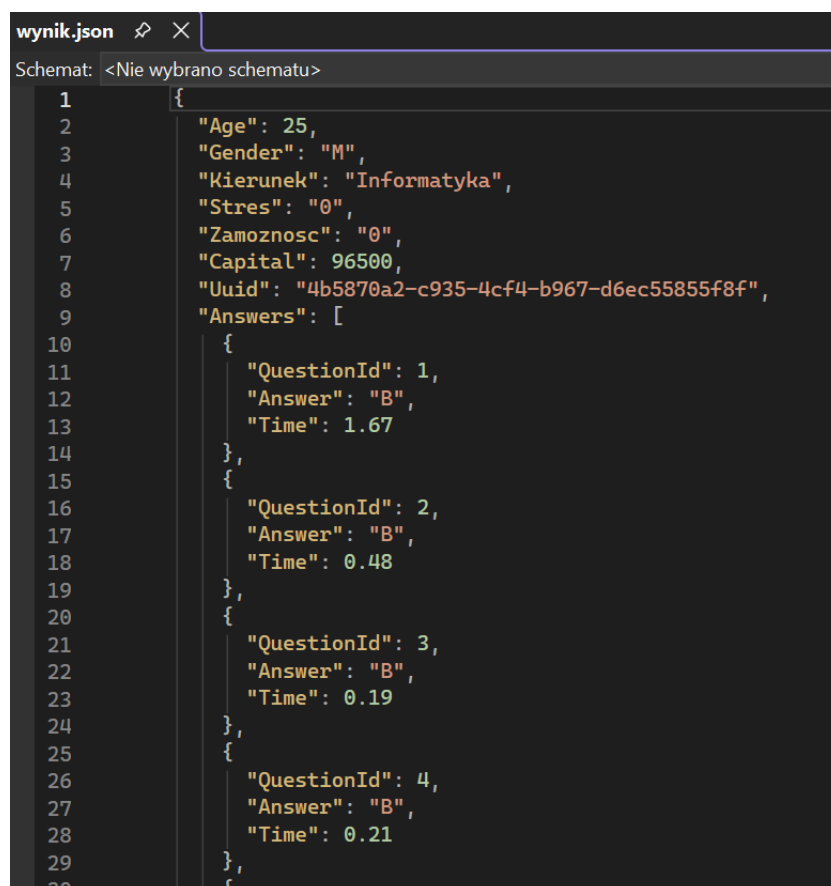
Zakończenie eksperymentu i zapis wyników

Opis: Po udzieleniu odpowiedzi na pytania aplikacja kończy badanie i zapisuje dane uczestników do pliku JSON, a na koniec wyświetla uczestnikowi finalne wyniki.

Klasy realizujące: Storage serializuje dane i zapisuje wyniki do pliku wynik.json.

QuestionWindow rozpoznaje koniec eksperymentu i rozpoczyna zapis danych.

Participant - to dane z tej klasy zostają zapisane.



```
wynik.json  ✎ ✕
Schemat: <Nie wybrano schematu>
1  {
2    "Age": 25,
3    "Gender": "M",
4    "Kierunek": "Informatyka",
5    "Stres": "0",
6    "Zamoznosc": "0",
7    "Capital": 96500,
8    "Uuid": "4b5870a2-c935-4cf4-b967-d6ec55855f8f",
9    "Answers": [
10   {
11     "QuestionId": 1,
12     "Answer": "B",
13     "Time": 1.67
14   },
15   {
16     "QuestionId": 2,
17     "Answer": "B",
18     "Time": 0.48
19   },
20   {
21     "QuestionId": 3,
22     "Answer": "B",
23     "Time": 0.19
24   },
25   {
26     "QuestionId": 4,
27     "Answer": "B",
28     "Time": 0.21
29   },
30 ]
}
```

Obraz 4 - Przykładowa zawartość pliku wynik.json.