

# **Zaawansowane metody programowania**

---

## **laboratorium**



Karpiński Maciej  
Kuczma Łukasz  
Zawada Marcin

Prowadzący mgr inż. Marcin Tracz

# Spis treści

<b>1</b>	<b>Opis funkcjonalny systemu</b>	<b>3</b>
<b>2</b>	<b>Opis technologiczny</b>	<b>4</b>
2.1	Aplikacja desktopowa . . . . .	4
2.1.1	Cargo . . . . .	4
2.1.2	chrono . . . . .	4
2.1.3	Cloud Firestore . . . . .	4
2.1.4	Firebase . . . . .	4
2.1.5	Firebase Authentication . . . . .	4
2.1.6	firestore-db-and-auth . . . . .	5
2.1.7	gdk-pixbuf . . . . .	5
2.1.8	gio . . . . .	5
2.1.9	Glade . . . . .	5
2.1.10	glib . . . . .	5
2.1.11	gtk . . . . .	5
2.1.12	GTK+ 3 . . . . .	5
2.1.13	oauth2 . . . . .	5
2.1.14	rand . . . . .	5
2.1.15	request . . . . .	6
2.1.16	Rust . . . . .	6
2.1.17	serde . . . . .	6
2.1.18	serde_json . . . . .	6
2.1.19	url . . . . .	6
2.1.20	Visual studio code . . . . .	6
2.1.21	webkit2gtk . . . . .	6
2.2	Aplikacja mobilna . . . . .	6
2.3	Aplikacja webowa . . . . .	6
<b>3</b>	<b>Wzorce projektowe</b>	<b>7</b>
3.1	Aplikacja desktopowa . . . . .	7
3.2	Aplikacja mobilna . . . . .	7
3.3	Aplikacja webowa . . . . .	7
<b>4</b>	<b>Instrukcja uruchamiania testów i systemu</b>	<b>8</b>
4.1	Aplikacja desktopowa . . . . .	8
4.2	Aplikacja mobilna . . . . .	8
4.3	Aplikacja webowa . . . . .	8
<b>5</b>	<b>Wnioski projektowe</b>	<b>9</b>
<b>6</b>	<b>Spis rysunków</b>	<b>10</b>

# **1 Opis funkcjonalny systemu**

## 2 Opis technologiczny

### 2.1 Aplikacja desktopowa

Aplikacja desktopowa została stworzona przy wykorzystaniu następujących technologii:

#### 2.1.1 Cargo

Cargo to menadżer pakietów i system kompilowania języka Rust. Cargo pobiera zależności, kompiluje je, tworzy pakiety do dystrybucji i przesyła je do [crates.io](https://crates.io), rejestru pakietów społeczności Rust. Cargo można rozbudować o dodatkowe możliwości poprzez instalację dodatkowych pakietów np. watch lub clippy. Większość użytkowników używa tego narzędzia do zarządzania swoimi projektami.

#### 2.1.2 chrono

Pakiet chrono jest biblioteką dat i godzin dla Rust. Chrono ściśle przestrzega normy ISO 8601, domyślnie rozpoznaje strefę czasową, posiada oddzielne typy, które nie posiadają strefy czasowej.

#### 2.1.3 Cloud Firestore

Cloud Firestore to elastyczna, skalowalna baza danych do tworzenia aplikacji mobilnych, internetowych i serwerowych z Firebase i Google Cloud. Podobnie jak Baza danych czasu rzeczywistego Firebase, zapewnia synchronizację danych między aplikacjami klientami za pośrednictwem odbiorników w czasie rzeczywistym i oferuje obsługę offline dla urządzeń przenośnych i internetowych, dzięki czemu możesz tworzyć responsywne aplikacje, które działają niezależnie od opóźnień w sieci lub łączności z Internetem.

#### 2.1.4 Firebase

Firebase to platforma opracowana przez Google do tworzenia aplikacji mobilnych i internetowych. Platforma Firebase obejmuje 18 produktów podzielonych na trzy grupy:

- Develop,
- Quality,
- Grow,

Aplikacja desktopowa wykorzystuje Cloud Firestore i Firebase Authentication.

#### 2.1.5 Firebase Authentication

Firebase Authentication zapewnia usługę uwierzytelniania dla backendu, łatwe w użyciu pakiety SDK i gotowe biblioteki interfejsu użytkownika do uwierzytelniania użytkowników w aplikacjach. Obsługuje uwierzytelnianie za pomocą haseł, numerów telefonów, popularnych dostawców tożsamości federacyjnych, takich jak Google, Facebook i Twitter. Firebase Authentication ściśle integruje się z innymi usługami Firebase i wykorzystuje standardy branżowe, takie jak OAuth 2.0 i OpenID Connect.

### **2.1.6 firestore-db-and-auth**

Pakiet umożliwiający łatwy dostęp do bazy danych Cloud Firestore za pośrednictwem konta usługi lub poświadczeń OAuth Firebase Authentication.

### **2.1.7 gdk-pixbuf**

Pakiet umożliwia wykorzystanie biblioteki Gdk-Pixbuf napisanej w C dla języka Rust.

### **2.1.8 gio**

Pakiet umożliwia wykorzystanie biblioteki GIO napisanej w C dla języka Rust.

### **2.1.9 Glade**

Glade jest aplikacją do wizualnego projektowania graficznego interfejsu użytkownika dla programów GTK+/GNOME. Projektowany interfejs jest zapisywany jako plik XML. Pliki w formacie GtkBuilder i Libglade mogą być ładowane przez odpowiednie biblioteki GTK+ lub Libglade.

### **2.1.10 glib**

Pakiet umożliwia wykorzystanie biblioteki GLib napisanej w C dla języka Rust.

### **2.1.11 gtk**

Pakiet umożliwia wykorzystanie bibliotek GDK 3, GTK+ 3 i Cairo napisanych w C dla języka Rust.

### **2.1.12 GTK+ 3**

Biblioteka służąca do tworzenia interfejsu graficznego do programów komputerowych. Pierwotnie stworzona na potrzeby programu GIMP, stąd też nazwa, pochodząca od ang. The GIMP Toolkit. Znak + pojawił się w nazwie, gdy autorzy dodali do oryginalnego GTK możliwość programowania obiektowego. GTK+ została napisana w C, aczkolwiek jest zaprojektowana obiektowo, w oparciu o implementację obiektowości dla C – GObject. Z biblioteki GTK+ można korzystać przy pomocy większości języków programowania. Biblioteka ta jest podstawą dla środowisk graficznych GNOME i Xfce. Na platformie uniksowej sama wykorzystuje bibliotekę GDK (odpowiedzialną za rysowanie obiektów) oraz GLib, zawierającą specjalne typy danych. Dzięki takiemu odseparowaniu GTK+ od systemu graficznego (w przypadku Uniksa jest to przeważnie X Window System) biblioteką bezpośrednio odpowiedzialną za interakcję z systemem graficznym, możliwe było łatwe przeportowanie GTK+ na inne niż uniksowe architektury (np.: Microsoft Windows oraz linuksowy DirectFB).

### **2.1.13 oauth2**

Pakiet implementujący protokół OAuth2 (RFC 6749).

### **2.1.14 rand**

Biblioteka Rust przeznaczona do generowania liczb losowych;

### **2.1.15 reqwest**

Pakiet reqwest ergonomiczny klient HTTP/HTTPS dla języka Rust.

### **2.1.16 Rust**

Kompilowany język programowania ogólnego przeznaczenia rozwijany przez Fundację Mozilla. Stworzony z myślą, aby był „bezpieczny, współbieżny i praktyczny”. Język zaprojektował Graydon Hoare w 2006 roku, w 2009 projekt został przyjęty pod skrzydła Mozilla Foundation. W 2010 Mozilla upubliczniła informację o języku. W 2011 roku kompilator języka, znany jako rustc, został z powodzeniem skompilowany przez samego siebie. Pierwsza numerowana wersja alfa została wydana w 2012 roku. 15 maja 2015 ukazała się wersja 1.0. Rust wykorzystuje Cargo jako menadżer pakietów. Wiele organizacji wykorzystuje ten język programowania w zastosowaniach produkcyjnych. Obecnie dwoma największymi otwartymi projektami korzystającymi z języka Rust są: Servo oraz kompilator Rusta.

### **2.1.17 serde**

Serde to framework do wydajnej i ogólnej serializacji i deserializacji struktur danych Rust.

### **2.1.18 serde\_json**

Serde\_json jest rozszerzeniem frameworka o możliwość serializacji i deserializacji struktur danych do/z formatu JSON.

### **2.1.19 url**

Biblioteka URL dla Rust, oparta na standardzie URL WHATWG.

### **2.1.20 Visual studio code**

Visual Studio Code – darmowy edytor kodu źródłowego z kolorowaniem składni dla wielu języków, stworzony przez Microsoft, o otwartym kodzie źródłowym. Oprogramowanie ma wsparcie dla debugowania kodu, zarządzania wersjami kodu źródłowego za pośrednictwem systemu kontroli wersji Git, automatycznego uzupełniania kodu IntelliSense, zarządzania wycinkami kodu oraz jego refaktoryzacji. Funkcjonalność aplikacji można rozbudować za pomocą rozszerzeń instalowanych z dedykowanego repozytorium rozszerzeń. Według badania przeprowadzonego przez serwis StackOverflow w 2018 roku, Visual Studio Code zostało ogłoszone najpopularniejszym narzędziem służącym wytwarzaniu oprogramowania, za którym na drugim miejscu znajduje się produkt tego samego twórcy, Microsoft Visual Studio. Oprogramowanie zostało stworzone w oparciu o framework Electron.

### **2.1.21 webkit2gtk**

Pakiet umożliwia wykorzystanie biblioteki webkit2gtk napisanej w C dla języka Rust.

## **2.2 Aplikacja mobilna**

## **2.3 Aplikacja webowa**

## 3 Wzorce projektowe

### 3.1 Aplikacja desktopowa

Pakiet bibliotek gtk-rs wykorzystuje następujące wzorce projektowe:

- Adapter
- Builder
- Singleton

Biblioteka firestore-db-and-auth wykorzystuje wzorce projektowe:

- Data mapper
- Repository

Główny obiekt przechowujący informacje dotyczącą interfejsu „GuiData” wykorzystuje wzorce projektowe:

- Builder
- Pool
- Prototype

Kod obsługujący API [TheCocktailDB](#) i [TheMealDB](#) wykorzystuje następujące wzorce projektowe:

- Data mapper
- Chain of Responsibilities
- Repository

Gdzie tylko było to możliwe wykorzystany w kodzie zostały następujące wzorce projektowe:

- Iterator
- Płynny interfejs
- Wstrzykiwanie zależności

### 3.2 Aplikacja mobilna

### 3.3 Aplikacja webowa

## 4 Instrukcja uruchamiania testów i systemu

### 4.1 Aplikacja desktopowa

Testy można uruchomić jedynie w systemie operacyjnym Linux. W celu lokalnego uruchomienia testów aplikacji należy w katalogu „Desktop” uruchomić w terminalu polecenie:

```
cargo test --test-threads=1
```

W celu zdalnego uruchomienia testów należy udać się na stronę [GitHub Actions](#).

Po pobraniu archiwum przeznaczonego dla systemu Windows lub Linux z strony [GitHub Releases](#), należy je wypakować a następnie uruchomić przy wykorzystaniu skrótu (Windows) lub aktywatora (Linux) o nazwie „Cookbook”.

### 4.2 Aplikacja mobilna

### 4.3 Aplikacja webowa



## **5 Wnioski projektowe**

## **6 Spis rysunków**

### **Spis rysunków**