Софтуер за генериране на изпитни билети

Даниел Василев

30 март 2023 г.

Съдържание

1 Увод

В настоящата дисертация ще бъде представен проект, който има за цел да улесни генерирането на изпитни билети за учебни заведения. Програмата е изградена на езика Rust и предлага лесен и ефективен начин за създаване на изпитни билети.

За да може един софтурен продукт да бъде завършен на време, трябва да задачите и целите на продукта да бъдат разделени на по-малки под задачи.

2 Използвани езици и технологии

2.1 Rust

Rust е програмен език от високо ниво създаден през 2006 година от Graydon Hoare, който по това време работи за Mozilla. През 2009 година разработката на езика бива спонсорирана от Mozilla, а през 2010 езика е обявен публично. [1]

2.1.1 Отличаващи се особености на езика

Езици като C#, Python и JavaScript използват система за освобождаване на паметта наречена Garbage Collector (GC). За да може да се освободят неизползваните променливи, изпълнението на програмата трябва да бъде спряно на пауза и да се провери дали има заделени региони от паметта, към които вече не се използват или са маркиране за освобождаване от програмиста [2]

Rust използва система наречена borrow checker, която проверява, по време на компилация, дали програмата следва следните принципи:

- Ресурсите (отделената памет за стойноста) могат да имат само един собственик и това е самата промелива. Когато променлива вече не може да бъде достъпена тя бива освободена.
- Когато една променлива бъде подадена към някоя функция, собственик на ресурсите става променливата във функцията. Ако се пробваме подадем отново променливата, компилатора ще ни каже, че променливата е преместена (Use of moved value).

2.1.2 Enum

Enum е един от основните типове в Rust. Всеки вариянт на enum-а може да има съдържа информация от различен вид [3]. Така са имплементирани някои от най-важните типове: Option<T> и Result<T, E>.

```
enum Option<T> {
    Some(T),
    None,
}

enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

Фигура 1: Стандартната имплементация на Option<T> и Result<T, E>

2.1.3 Option типа

В повечето езици съществува идеята за NULL пойнтери. Когато един pointer е Null това означа, че той сочи към нищо. Идеята за Null на теория е много добра, но на практика създава повече проблеми. Ако се пробваме да достъпим pointer който е Null, програмата ще крашне или в някои езици като C# ще хвърли NullReferenceException.

Разработчиците на Rust са намерили много добър заместител на Null и това е Option enum-а, който има два варинта. Това са Some(T) когато имаме някаква стойнос и None когато нямаме нищо.

2.1.4 Result типа

Когато програмираме на C# много често ни се случва да хвърляме Exception-и и съответно да ги хващаме с try/catch блока. Exception-ите се ползват когато в една функция възникне грешка.

Във Фигура 2 е даден код който на пръв поглед изглежда добре, но има скрити бъгове. Какво ще стане ако потребителя въведе дума вместо число? Ще получим Exception който ни казва: "Input string was not in a correct format"[Фигура 3].

```
1   int ReadNumberFromUser()
  1 {
  2     string user_input = Console.ReadLine();
  3     int parsed_int = int.Parse(user_input);
  4     return parsed_int;
  5 }
  6
  7 int number = ReadNumberFromUser();
  8 Console.WriteLine($"{number} * 2 = {number * 2}");
```

Фигура 2: Пример за скрит Exception

```
~/Programming/C#/Scratchpad
>  dotnet run
word
Unhandled exception. System.FormatException: Input string was not in a correct format.
   at System.Number.ThrowOverflowOrFormatException(ParsingStatus status, TypeCode type)
   at System.Number.ParseInt32(ReadOnlySpan`1 value, NumberStyles styles, NumberFormatInfo info)
   at System.Int32.Parse(String s)
   at Program.<<Main>$>g__ReadNumberFromUser|0_0() in /home/daniel/Programming/C#/Scratchpad/Program.cs:line 4
   at Program.<Main>$(String[] args) in /home/daniel/Programming/C#/Scratchpad/Program.cs:line 8
```

Фигура 3: Изход на кода от Фигура 2

Проблема е че ние като програмисти не знаем, че int.Parse може да хвърли Exception без да се консултираме с документацията [4]. Същият код написан на Rust би изглеждал по следния начин [Фигура 4].

Разликата между C# и Rust е че Rust кода ни кара ни показва типовете при успех и грешка. Функцията връща променлива от тип Result<T, E> където T е променливата от тип i32 (int) ако всичко се и изпълнило без проблем, а E е от тип ParseIntError.

```rust
use std::num::ParseIntError;

fn read_number_from_user() -> Result<i32, ParseIntError> {
    let mut user_input = String::new(); => String
    std::io::stdin().read_line(&mut user_input).unwrap(); <- (buf)
    return user_input.trim().parse();
}

fn main() {
    let number = read_number_from_user(); => Result<i32, ParseIntError>

    match number {
        Ok(number) => { => i32
            // NOTE принтира резултата
            println!("{} * 2 = {}", number, number * 2);
        }
        Err(err) => { => ParseIntError
            // NOTE принтира грешката в конзолата, но не crash-ва!
            println!("{:?}", err);
        }
    }
}
```

Фигура 4: Кода от Фигура 2 написан на Rust

За да използваме резултата от функцията, какъвто и да е той, можем да използваме match. С match можем да проверим дали резултата е Ok или Err.

2.2 egui

egui е проста, бърза и много преносима библиотека за графични потребителски интерфейси (GUI). Egui работи на много платформи включително: уеб браузъри, като обикновено приложение и в някои game engine-а. Написана е на Rust и има много лесен и интуативен API за разработване.

Главните цели на проекта са:

- Най-лесната за използване GUI библиотека
- Отзивчив: цели поне 60 FPS при компилация с Debug опциите
- Преносим: кода да работи в браузър и като собствено приложение
- Лесен за интегриране във всяка среда
- Разширяем: лесно да пишете свои собствени джаджи за egui
- Модулен: можете да използвате малки части от egui и да ги комбинирате по нови начини
- Минимален брой завивисимости (библиотеки)

2.2.1 Минимален пример

egui библиотека ни дава достъп до App интерфейса. Този интерфес съдържа една функция update. Тя се извиква всеки път когато потребителския интерфейс се е променил или се получи някакъв евент от мишката или клавиатурата. [Фигура 5]

```
impl eframe::App for MyApp {
    fn update(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {
        egui::CentralPanel::default().show(ctx, |ui| {
            ui.heading("My egui Application");
            ui.horizontal(|ui| {
                let name_label = ui.label("Your name: ");
                ui.text_edit_singleline(&mut self.name)
                    .labelled_by(name_label.id);
            });
            ui.add(egui::Slider::new(&mut self.age, 0..=120).text("age"));
            if ui.button("Click each year").clicked() {
                self.age += 1;
            }
            ui.label(format!("Hello '{}', age {}", self.name, self.age));
        });
    }
}
```

Фигура 5: Имплементация на egui интерфейса

Библиотеката е достатъчно умна сама да прецени дали се нуждае от повторно изобразяване.

egui е GUI библиотека от незабавен режим (Immediate Mode). Това означава че начина по който искаме да изглежда графичния интерфейс се описва извиквайки методи. По този начин се упростява разработката. За да покажем едит прост бутон се нуждаем от един if оператор.

3 Заключение

Литература

[1] Wikipedia contributors. Rust (programming language) — Wikipedia, The Free Encyclopedia. 2023. URL: https://en.wikipedia.org/w/index.php?title=Rust_(programming_language)&oldid=1146879721#Origins_(2006%E2%80%932012).

[2] Wikipedia contributors. Garbage collection (computer science) — Wikipedia, The Free Encyclopedia. 2023. URL: https://en.wikipedia.org/w/index.php?title=Garbage_collection_(computer_science)&oldid=1146816153.

[3] Rust contributors. Defining an Enum. 2021. URL: https://doc.rust-lang.org/book/ch06-01-defining-an-enum.html.

[4] Microsoft. Int32.Parse Method. 2023. URL: https://learn.microsoft.com/en-us/dotnet/api/system.int32.parse?view=net-8.0.