

Софтуер за генериране на изпитни билети

Даниел Василев

1 април 2023 г.

Съдържание

1	Увод	2
2	Използвани езици и технологии	3
2.1	Rust	3
2.1.1	Отличаващи се особености на езика	3
2.1.2	Enum	3
2.1.3	Option типа	4
2.1.4	Result типа	4
2.2	egui	6
2.2.1	Минимален пример	6
2.3	genpdf	7
2.3.1	Елементи	7
3	Заклучение	8

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

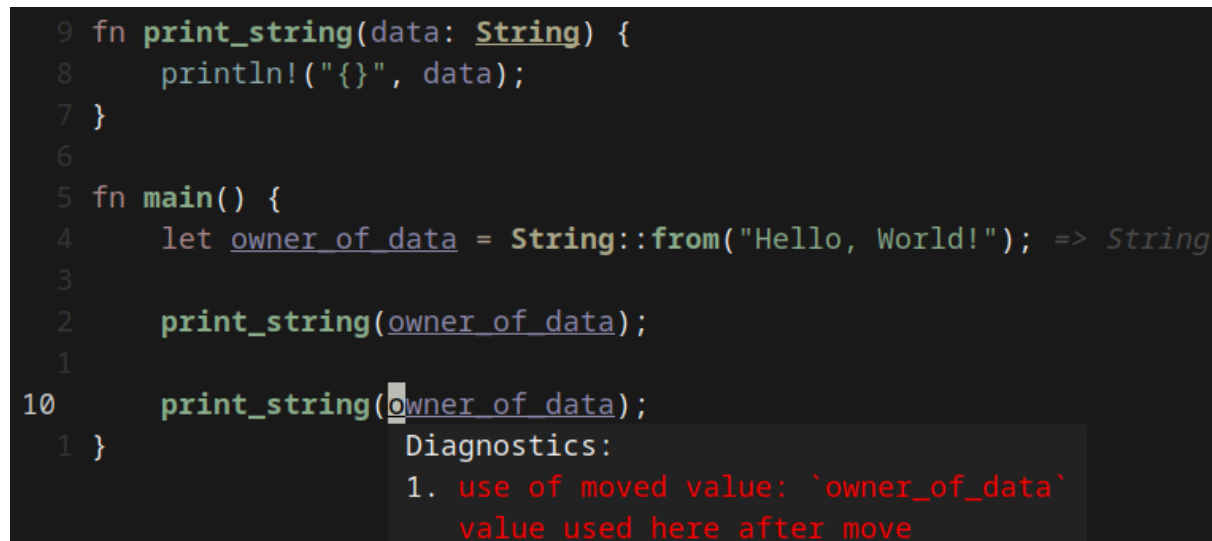
Rust е програмен език от високо ниво създаден през 2006 година от Graydon Hoare, който по това време работи за Mozilla. През 2009 година разработката на езика бива спонсорирана от Mozilla, а през 2010 езика е обявен публично. [1]

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 стойността) могат да имат само един собственик и това е самата променлива. Когато променлива вече не може да бъде достъпена ресурсите биват освободени.
- Когато една променлива бъде подадена към някоя функция, собственик на ресурсите става функцията. Ако се пробваме подадем отново променливата, компилатора ще ни каже, че променливата е била преместена (Use of moved value).



```
9 fn print_string(data: String) {
8     println!("{}", data);
7 }
6
5 fn main() {
4     let owner_of_data = String::from("Hello, World!"); => String
3
2     print_string(owner_of_data);
1
10    print_string(owner_of_data);
1 }
```

Diagnostics:
1. use of moved value: `owner_of_data`
value used here after move

Фигура 1: Модела на собственик в Rust

2.1.2 Enum

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

```

1  enum Option<T> {
2      Some(T),
3      None,
4  }
5  enum Result<T, E> {
6      Ok(T),
7      Err(E),
8  }

```

Фигура 2: 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 означава, 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 варианта. Това са `Some(T)` когато имаме някаква стойност 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 3 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".

```

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

```

Фигура 3: Пример за скрит Exception

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 5].

```
~/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 4: Изход на кода от Фигура 3

```
1 use std::num::ParseIntError;
2
3 fn read_number_from_user() -> Result<i32, ParseIntError> {
4     let mut user_input = String::new(); => String
5     std::io::stdin().read_line(&mut user_input).unwrap(); <- (buf)
6     return user_input.trim().parse();
7 }
8
9 fn main() {
10     let number = read_number_from_user(); => Result<i32, ParseIntError>
11
12     match number {
13         Ok(number) => { => i32
14             /* NOTE принтира резултата
15             println!("{}", number * 2);
16         }
17         Err(err) => { => ParseIntError
18             /* NOTE принтира грешката в конзолата, но не crash-ва!
19             println!("{:?}", err);
20         }
21     }
22 }
```

Фигура 5: Кода от Фигура 3 написан на Rust

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`.

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 интуитивен 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 зависимости (библиотеки)

2.2.1 Минимален пример

egui библиотека ни дава достъп до App интерфейса. Този интерфейс 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 6]

```
1
2 impl eframe::App for MyApp {
3     fn update(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {
4         egui::CentralPanel::default().show(ctx, |ui| {
5             ui.heading("My egui Application");
6             ui.horizontal(|ui| {
7                 let name_label = ui.label("Your name: ");
8                 ui.text_edit_singleline(&mut self.name)
9                     .labelled_by(name_label.id);
10            });
11            ui.add(egui::Slider::new(&mut self.age, 0..=120).text("age"));
12            if ui.button("Click each year").clicked() {
13                self.age += 1;
14            }
15            ui.label(format!("Hello '{}', age {}", self.name, self.age));
16        });
17    }
18 }
19
```

Фигура 6: 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 упростиава разработката. За да покажем един прост бутон се нуждаем от един if оператор.

2.3 genpdf

genpdf е библиотека която абстрахира създаването на PDF файлове. Тя се грижи за оформлението на страницата, подравняването на текста и изобразяването на структурата на документа в PDF файл. Библиотека както и всичките и зависимости са написани на Rust и следват добрите практики на езика [5].

2.3.1 Елементи

genpdf използва елементи за да опише оформлението на документа. Всеки елемент имплементира Element интерфейса. Интерфейса съдържа функция render която бива извикана всеки път когато елемента трябва да бъде показан в PDF файла [6].

Използвайки Element интерфейса, разработчиците на genpdf са ни предоставили най-често използваните елементи:

- Контейнери:
 - LinearLayout: подрежда елементите си последователно
 - TableLayout: подрежда елементите си в колони и редове
 - OrderedList/UnorderedList: подреждат елементите им последователно с bullet-и
- Текст:
 - Text: един ред текст
 - Paragraph: подравнен параграф
- Обвивки:
 - FramedElement: елемент с рамка
 - PaddedElement: добавя разстояние между елементите
 - StyledElement: задава стил по подразбиране за обвития елемент и неговите деца
- Други:
 - Image: снимка с описание
 - Break: добавя прекъсвания на редове като разделител
 - PageBreak: добавя принудително прекъсване на страницата

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_\(2006%E2%80%932012\)](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](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>.
- [5] ireas. genpdf-rs - User-friendly PDF generator written in pure Rust. 2021. URL: <https://git.sr.ht/~ireas/genpdf-rs>.
- [6] ireas. docs.rs - Trait genpdf::Element. 2021. URL: <https://docs.rs/genpdf/0.2.0/genpdf/trait.Element.html>.