



ОСОБЛИВОСТІ МОВИ ТА ЕКОСИСТЕМИ CRYSTAL У КОНТЕКСТІ РОЗРОБКИ ВЕБ-СЕРВЕРНИХ ЗАСТОСУНКІВ

ПРЕЗЕНТАЦІЯ ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

КЕРІВНИК КВАЛІФІКАЦІЙНОЇ РОБОТИ

ЗАХОЖЕНКО П.О.

ВИКОНАВ СТУДЕНТ

КРИВОШЕЄВ І.С.

ВСТУП

- Crystal — це молода статично типізована компільована мова програмування, яка має синтаксис, подібний до Ruby. Вона була створена з метою поєднати виразність Ruby зі швидкістю таких мов, як C. . І хоча ця мова є молодого, перша версія з'явилася у 2014 році, вона вже має достатньо розвинену екосистему для створення веб-серверних застосунків.
- Метою цієї роботи є розробка ефективного веб-серверного застосунку з використанням мови та екосистеми Crystal, дослідження переваг та недоліків роботи з молодого та не повністю сформованою екосистемою у контексті розробки RESTful застосунку.

НАЙПОПУЛЯРНІШІ БЕКЕНД ФРЕЙМВОРКИ



CRYSTAL

Мова програмування Crystal — це високорівнева мова програмування загального призначення з синтаксисом, схожим на Ruby, але з покращеною продуктивністю та статичною типізацією. Crystal розроблена з метою поєднання зручності та елегантності Ruby з ефективністю та безпекою, характерними для компільованих мов.

Синтаксис мови Crystal був успадкований багато в чому від Ruby.

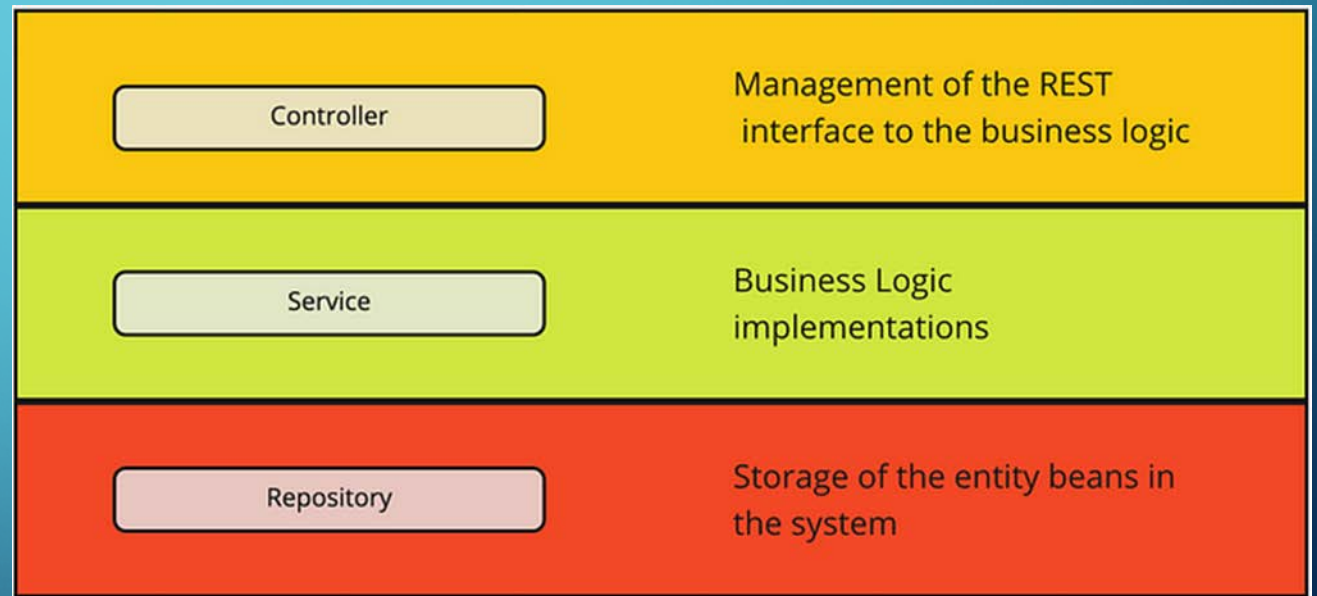
На відміну від Ruby, Crystal дозволяє визначати обмеження типів для параметрів методів, значення, яке повертає метод, обмеження типу змінної при її визначенні, обмеження для полів класів.



CRYSTAL

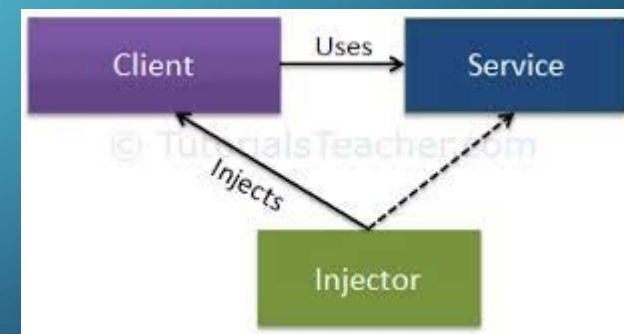
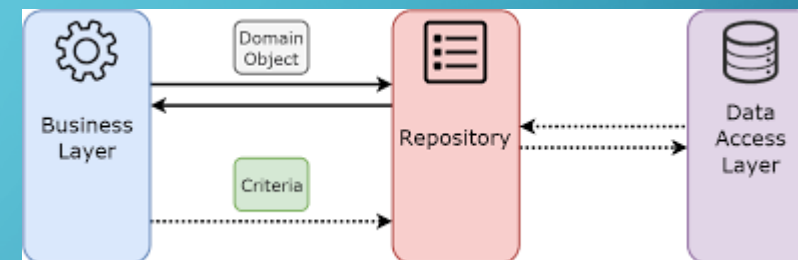
АРХІТЕКТУРНІ ПРИНЦИПИ ЗАСТОСУНКУ

- Веб-сервер буде побудований з дотриманням архітектурного паттерну Controller-Service-Repository. Цей паттерн розбиває функціональність додатка на три основні компоненти: Контролер (Controller), Сервіс (Service) і Репозиторій (Repository). Кожен з цих компонентів має свої відповідні обов'язки і допомагає досягти принципу розділення відповідальностей (Separation of Concerns) і принципу єдиного обов'язку (Single Responsibility Principle)



АРХІТЕКТУРНІ ПРИНЦИПИ ЗАСТОСУНКУ

- Шаблон проектування Repository — це шаблон проектування програмного забезпечення, який діє як проміжний рівень між бізнес-логікою програми та сховищем даних. Його основна мета — надати структурований і стандартизований спосіб доступу, керування та маніпулювання даними, абстрагуючись від базових деталей технологій зберігання даних.
- Dependency Injection (DI) — це шаблон проектування, який використовується в об'єктно-орієнтованому програмуванні при розробці програмного забезпечення для досягнення інверсії керування (IoC) між класами та їхніми залежностями. У DI залежності класу (тобто інші об'єкти, необхідні класу для виконання своїх функцій) «впроваджуються» в клас спеціальним сервісом-ін'єктором, а сам клас не створює залежностей і не керує ними.



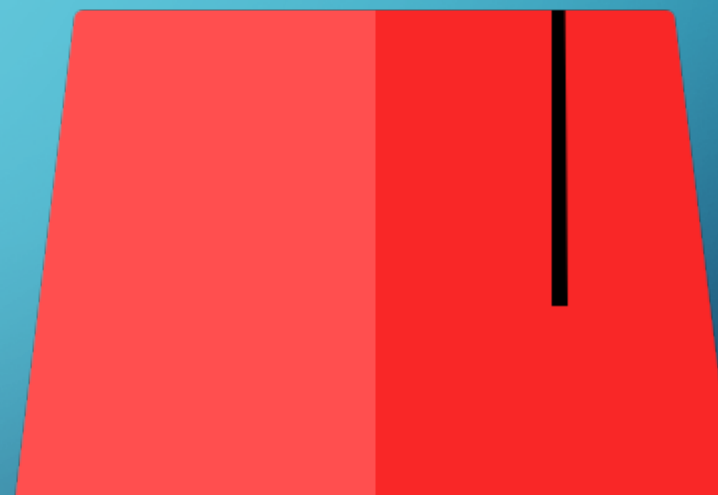
АВТОРИЗАЦІЯ

- Через те, що архітектура RESTful веб-сервіса не передбачає стану, зберігання інформації про користувача, його дозволи покладено на клієнт. Зазвичай для цього використовуються дві технології – HTTP-cookies та JSON Web Token (JWT). В цій роботі було використано JWT.
- JWT — це відкритий стандарт (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами як об'єкт JSON.

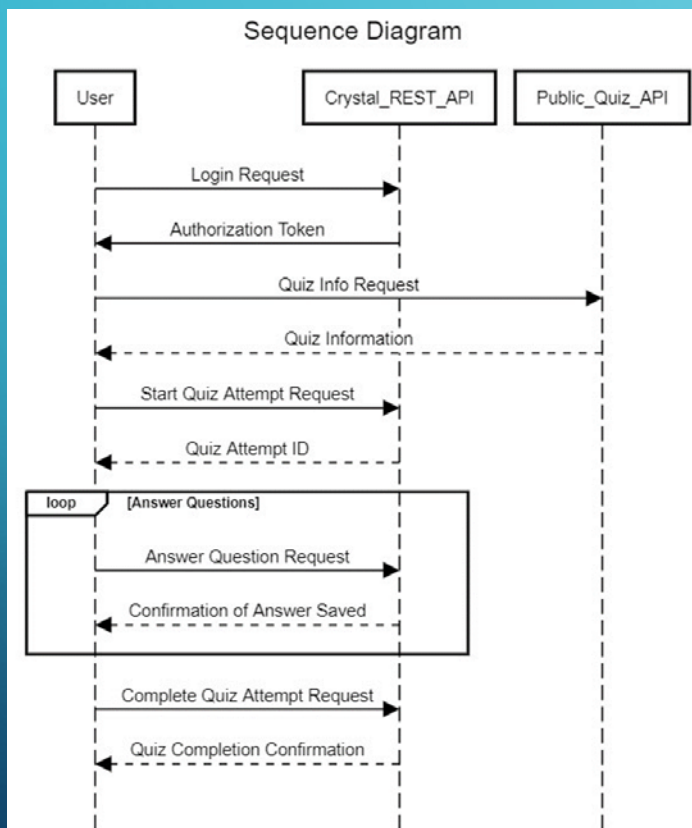


ВИКОРИСТАНІ БІБЛІОТЕКИ CRYSTAL

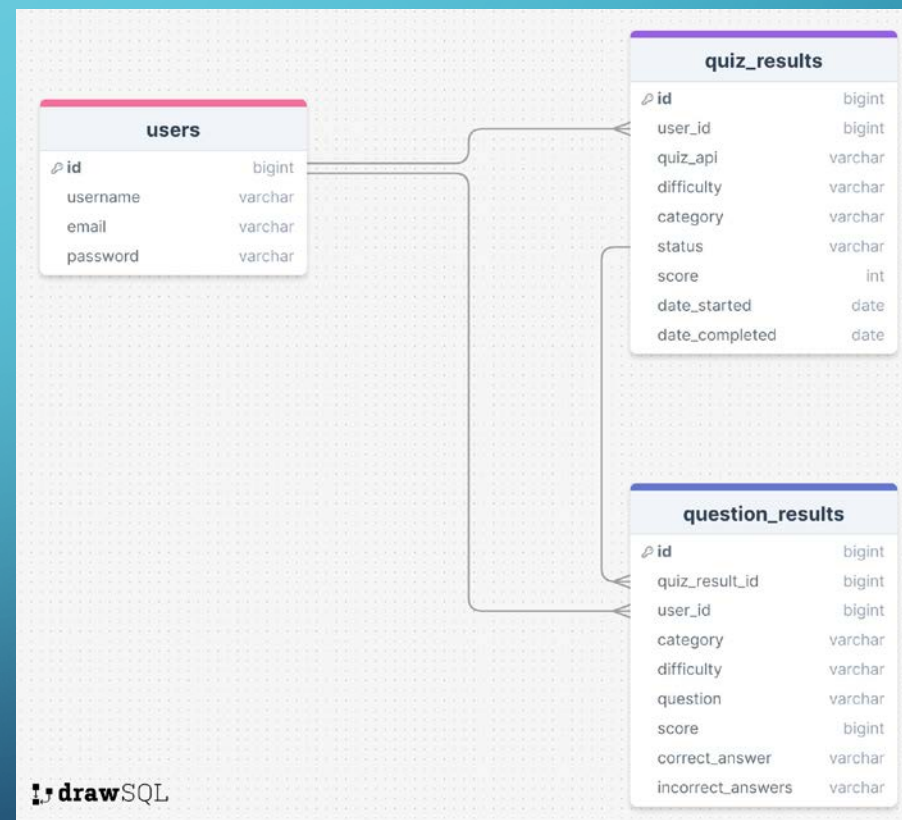
- Для реалізації Restful архітектури був використаний фреймворк Kemal. Kemal — це легкий веб-фреймворк для мови програмування Crystal, спроектований для швидкої розробки веб-додатків. Він надає зручний і елегантний спосіб створення веб-серверів і обробки HTTP-запитів за допомогою Crystal. Для обробки запиту достатньо визначити шлях до ресурсу та назву HTTP-методу.
- Для реалізації шаблону Repository була використана бібліотека Jennifer. Основним елементом Jennifer є моделі, які розробники створюють як абстракцію для доступу до таблиці у реляційній базі даних.
- Для реалізації шаблону DI в Crystal використовувалась бібліотека Crystal-DI. Вона підтримує реєстрацію залежностей в окремому модулі, автоматичну ін'єкцію залежностей в конструктори класів, збереження єдиного об'єкту сервісу



АРХИТЕКТУРА ЗАСТОСУНКУ

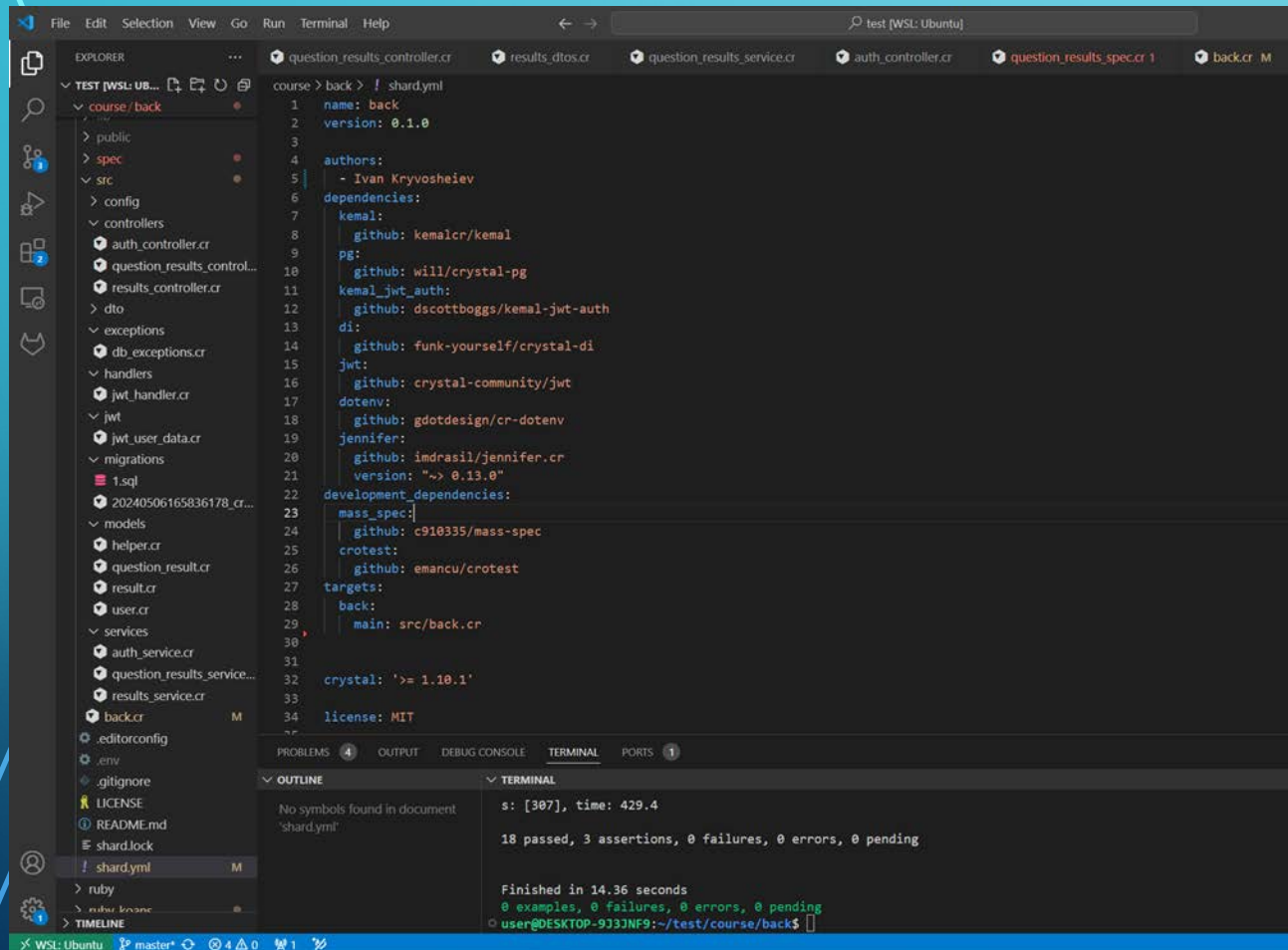


Sequence Diagram



Database diagram

РЕАЛІЗАЦІЯ ДОДАТКУ



The screenshot shows the VS Code interface with a project named 'course' open. The Explorer panel on the left shows the project structure, including folders like 'public', 'spec', 'src', 'config', 'controllers', 'dto', 'exceptions', 'handlers', 'jwt', 'migrations', 'models', and 'services'. The main editor displays the 'shard.yml' file, which defines the project's dependencies and development dependencies. The terminal window at the bottom shows the output of a command, indicating that the project was successfully built and tested.

```
1 name: back
2 version: 0.1.0
3
4 authors:
5   - Ivan Kryvosheiev
6 dependencies:
7   kemal:
8     github: kemalcr/kemal
9   pg:
10    github: will/crystal-pg
11  kemal_jwt_auth:
12    github: dscottbogs/kemal-jwt-auth
13  di:
14    github: funk-yourself/crystal-di
15  jwt:
16    github: crystal-community/jwt
17  dotenv:
18    github: gdotdesign/cr-dotenv
19  jennifer:
20    github: imdrasil/jennifer.cr
21  version: "~> 0.13.0"
22 development_dependencies:
23   mass_spec:
24     github: c910335/mass-spec
25   crotest:
26     github: emancu/crotest
27 targets:
28   back:
29     main: src/back.cr
30
31
32 crystal: '>= 1.10.1'
33
34 license: MIT
```

Terminal Output:

```
s: [307], time: 429.4
18 passed, 3 assertions, 0 failures, 0 errors, 0 pending

Finished in 14.36 seconds
0 examples, 0 failures, 0 errors, 0 pending
user@DESKTOP-933JNF0:~/test/course/back$
```

```
course > back > src > services > results_service.cr > ...
1 require "db"
2 require "pg"
3 require "../config/"
4 require "crypto/bcrypt/password"
5 require "../models/result"
6 require "../exceptions/"
7
8 module Services
9   class ResultsService
10     def result_statuses
11       { started: "Started", completed: "Completed" }
12     end
13
14     def get_result_by_id(id : Int32)
15       Result.find_by ([:id => id])
16     end
17
18     def get_user_results(user : Int32)
19       Result.where { c("user_id") == user }.to_a
20     end
21
22     def has_result_by_id(id : Int32)
23       Result.where { c("id") == id }.exists?
24     end
25
26     def set_result_completed(id : Int32, date_completed : String)
27       statuses = result_statuses
28       res = get_result_by_id(id)
29       if (!res)
30         raise NoSuchRecordException.new(id)
31       end
32
33       res.status = statuses[:completed]
34       res.date_completed = date_completed
35       res.save
36     end
37   end
38 end
```

РЕАЛІЗАЦІЯ ДОДАТКУ

```
course > back > src > controllers > question_results_controller.cr > ...
1 require "../services/*"
2 require "json"
3 require "../dto/*"
4 require "../jwt/*"
5 require "../config/inject"
6
7 question_results_service = Inject.resolve(Services::QuestionResultsService)
8 results_service = Inject.resolve(Services::ResultsService)
9
10 get "/questionResults/:quiz_id" do |env|
11   user = (env.get "user").as(JWTUserData::User)
12
13   quiz_id = env.params.url["quiz_id"].to_i?
14   if (!quiz_id || !results_service.has_result_by_id(quiz_id))
15     halt env, status_code: 400, response: "No such quiz result"
16   end
17
18   quizResult = results_service.get_result_by_id(quiz_id)
19   if (!quizResult || quizResult.user_id != user.id)
20     halt env, status_code: 403, response: "Forbidden"
21   end
22
23   res = question_results_service.get_results_by_quiz_result_id(quiz_id)
24   res.to_json
25 end
26
27 post "/questionResults/:quiz_id" do |env|
28   user = (env.get "user").as(JWTUserData::User)
29
30   quiz_id = env.params.url["quiz_id"].to_i?
31   if (!quiz_id || !results_service.has_result_by_id(quiz_id))
32     halt env, status_code: 400, response: "No such quiz result"
33   end
34
35   quizResult = results_service.get_result_by_id(quiz_id)
36   if (!quizResult || quizResult.user_id != user.id)
37     halt env, status_code: 403, response: "Forbidden"
```

```
course > back > src > handlers > jwt_handler.cr > ...
1 require "kernal"
2 require "../jwt/*"
3
4 class JWTHandler < Kemal::Handler
5   exclude ["/auth/*"], "POST"
6
7   def call(env)
8     return call_next(env) if exclude_match?(env)
9
10    if (env.request.method === "OPTIONS")
11      env.response.status_code = 200
12      env.response.headers.add("Access-Control-Allow-Headers", "authorization")
13      return
14    end
15
16    begin
17      auth_header = env.request.headers["Authorization"]
18      if (!auth_header.starts_with?("Bearer "))
19        raise "Incorrect Authorization header"
20      end
21      token = auth_header[7..-1]
22
23      user = JWTUserData.decode_user(token)
24      env.set "user", user
25    rescue ex
26      puts ex.message
27      env.response.status_code = 401
28      return
29    end
30    call_next(env)
31  end
32 end
```


РЕЗУЛЬТАТИ

В результаті була розроблена REST API з відповідними ендпоінтами:

- POST /auth/register – реєстрація користувача за переданими параметрами нікнейму, пошти та пароля. Повертається токен авторизації
- POST /auth/login – логін користувача за переданими параметрами нікнейму та пароля. Повертається токен авторизації
- GET /results/user – отримати всі результати тестів користувача
- POST /results – створення спроби тесту користувача.
- PUT /completed/:id – переведення тесту у закінчений стан за ідентифікатором
- GET /questionResults/:quiz_id – отримати всі результати відповідей користувача на питання певного тесту
- POST /questionResults/:quiz_id – створення результату відповіді користувача на питання певного тесту.

Id	Quiz App	Score	Status	Date started	Date completed	Category	Difficulty
30	openTrivia	9	Completed	2024-03-24 16:50:59 +0000	2024-03-24 16:51:12 +0000	Not Set	Not Set
31	trivia	15	Started	2024-03-24 17:15:06 +0000		Not Set	Not Set
32	trivia	15	Started	2024-03-24 17:18:17 +0000		Not Set	Not Set
33	trivia	13	Started	2024-03-24 17:18:17 +0000		Not Set	Not Set
34	trivia	13	Completed	2024-03-24 17:18:27 +0000	2024-03-24 17:18:34 +0000	Not Set	Not Set
35	quiz	7	Started	2024-03-24 17:21:02 +0000		Not Set	Not Set
36	trivia	12	Started	2024-03-24 17:21:08 +0000		Not Set	Not Set
37	api	15	Started	2024-04-28 15:21:35 +0000		Not Set	medium

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

Під час розробки були створені API тести, які дуже допомагали швидко перевірити, чи внесені зміни у код не спричинили небажаних змін у логіці роботи додатку.

```
d\\"", args: ["user", "user@gmail.com", "$2a$10$10W9mROaFZco3djbBiZTjuK0w1cFNEALMIkdrQmxOIiKh2Y9ZE6HC"], time: 299.8
2024-05-14T20:26:40.495531Z DEBUG - db: -- query: "COMMIT", time: 0.1
335
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InVzZXIiLCJpc2CI6ImZmM1LCJleHAiOiJlMTU4MDQ4MDEu-tu-qqp5NbEgIRi3pro7-s1410bmXmorBibDyvwrb30
2024-05-14 20:26:40 UTC 200 POST /auth/register 484.1ms
2024-05-14T20:26:40.502080Z DEBUG - db: -- query: "START", time: 0.2
2024-05-14T20:26:40.502483Z DEBUG - db: -- query: "INSERT INTO \"quiz_results\"(\"score\", \"quiz_api\", \"difficulty\", \"category\", \"user_id\", \"status\", \"date_started\", \"date_completed\") VALUES ($1, $2, $3, $4, $5, $6, $7, $8) RETURNING \"id\"", args: [10, "Quiz API", "easy", "Science", 335, "Started", "2024-05-14 20:26:40 +00:00", nil], time: 373.3
2024-05-14T20:26:40.502485Z DEBUG - db: -- query: "COMMIT", time: 0.1
2024-05-14 20:26:40 UTC 200 POST /results 2.27ms
2024-05-14T20:26:40.504677Z DEBUG - db: -- query: "SELECT \"quiz_results\".* FROM \"quiz_results\" WHERE \"quiz_results\".\"id\" = $1 LIMIT 1 ", args: [254], time: 348.8
2024-05-14T20:26:40.504908Z DEBUG - db: -- query: "SELECT \"quiz_results\".* FROM \"quiz_results\" WHERE \"quiz_results\".\"id\" = $1 LIMIT 1 ", args: [254], time: 204.5
2024-05-14T20:26:40.505094Z DEBUG - db: -- query: "START", time: 0.2
2024-05-14T20:26:40.505427Z DEBUG - db: -- query: "UPDATE \"quiz_results\" SET \"status\"= $1, \"date_completed\"= $2 WHERE \"id\" = $3", args: ["Completed", "2024-05-14 20:26:40 +00:00", 254], time: 313.4
2024-05-14T20:26:40.505429Z DEBUG - db: -- query: "COMMIT", time: 0.1
2024-05-14 20:26:40 UTC 200 PUT /results/completed/254 4.84ms
2024-05-14 20:26:40 UTC 200 GET /results/user 673.8µs
2024-05-14T20:26:40.509868Z DEBUG - db: -- query: "SELECT \"quiz_results\".* FROM \"quiz_results\" WHERE \"quiz_results\".\"user_id\" = $1 ", args: [335], time: 493.0

18 passed, 3 assertions, 0 failures, 0 errors, 0 pending

Finished in 16.47 seconds
0 examples, 0 failures, 0 errors, 0 pending
```

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

```
● user@DESKTOP-9J3JNF9:~/test$ wrk -t2 -c100 -d30s --latency --header "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InRlc3QzNTUiLCJpZCI6OCwiZXhwIjoxNzE1ODgzODYwfQ.Ei0CXnii7xRcNpXrB0_f5exEdI9DdHXC4M4mu8EBiIU" http://127.0.0.1:3000/results/user
Running 30s test @ http://127.0.0.1:3000/results/user
  2 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/-  Stdev
    Latency    218.17ms  171.84ms   1.98s   82.93%
    Req/Sec    260.25    68.05   555.00   74.79%
  Latency Distribution
    50%    121.57ms
    75%    325.80ms
    90%    517.46ms
    99%    558.40ms
  15459 requests in 30.03s, 25.20MB read
Requests/sec:    514.72
Transfer/sec:    859.04KB
○ user@DESKTOP-9J3JNF9:~/test$
```


ВИСНОВКИ

- Під час виконання роботи спочатку було розглянуто найбільш популярні існуючі фреймворки та екосистеми для створення REST API сервісів. Потім було проаналізовано можливості мови Crystal, архітектурні паттерни та підходи до розробки REST API сервісів такі як Controller-Service-Repository, Repository Design Pattern та Dependency Injection. Також було показано які бібліотеки та фреймворки існують у екосистемі Crystal для реалізації даних шаблонів проектування. Іще було проаналізовано можливості Crystal для автоматизованого тестування коду.
- У практичній частині роботи було реалізовано REST API застосунок з використанням відповідних паттернів на тему платформи для онлайн-квізів. Було використано фреймворк Kemal, авторизацію на основі JWT токенів, бібліотеку Jennifer для підключення до бази даних. Також були розроблені автоматизовані API тести для даного застосунку. Було продемонстровано роботу даного застосунку, виконання тестів, проведено аналіз продуктивності додатку.
- З результатів виконання роботи можна зробити висновок, що хоча Crystal ще є досить молодою мовою програмування з не повністю розвиненою екосистемою, він добре підходить для ефективної розробки REST API застосунків