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 молодою, 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 молодою та не повністю сформованою екосистемою у контексті розробки RESTful застосунку.

НАЙПОПУЛЯРНІШІ БЕКЕНД ФРЕЙМВОРКИ

CRYSTAL

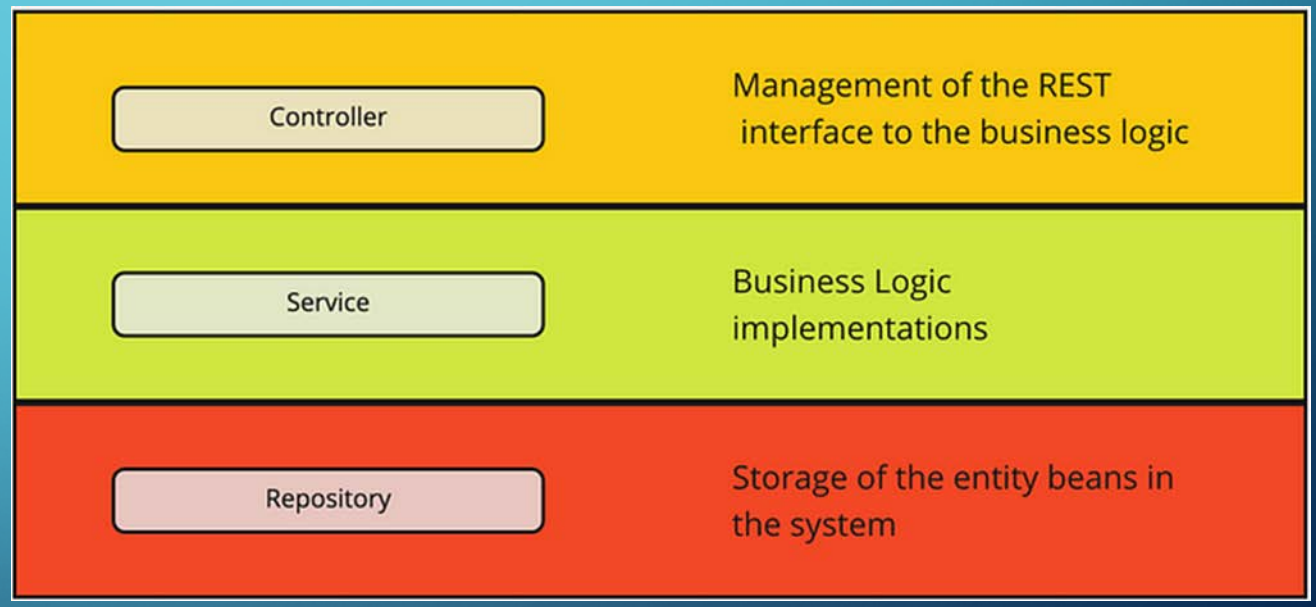
Мова програмування Crystal — це високорівнева мова програмування загального призначення з синтаксисом, схожим на Ruby, але з покращеною продуктивністю та статичною типізацією. Crystal розроблена з метою поєднання зручності та елегантності Ruby з ефективністю та безпекою, характерними для компільованих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

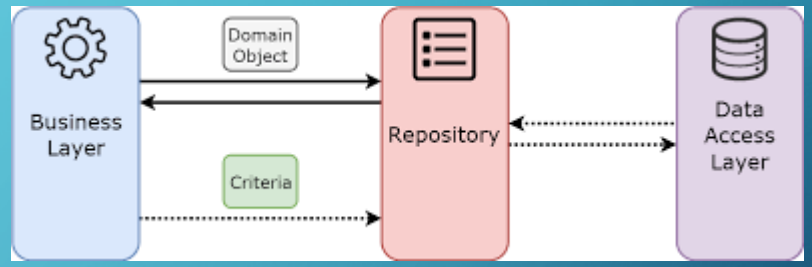
АРХІТЕКТУРНІ ПРИНЦИПИ ЗАСТОСУНКУ

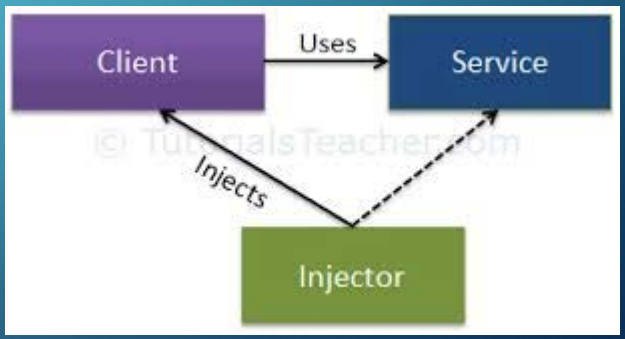
- Веб-сервер буде побудований з дотриманням архітектурного паттерну Controller-Service-Repository. Цей паттерн розбиває функціональність додатка на три основні компоненти: Контролер (Controller), Сервіс (Service) і Репозиторій (Repository). Кожен з цих компонентів має свої відповідні обов'язки і допомагає 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.





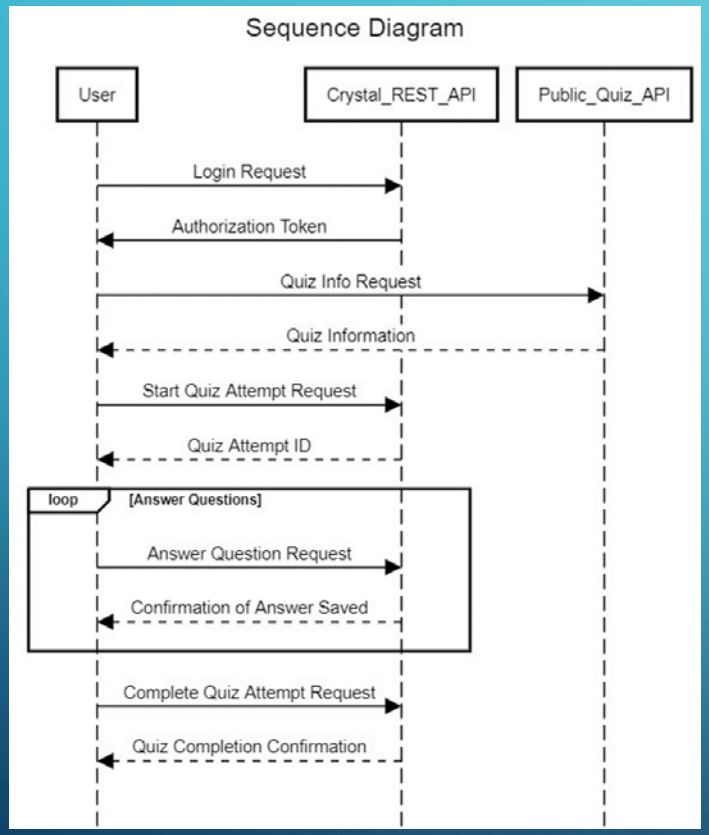
АВТОРИЗАЦІЯ

- 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.

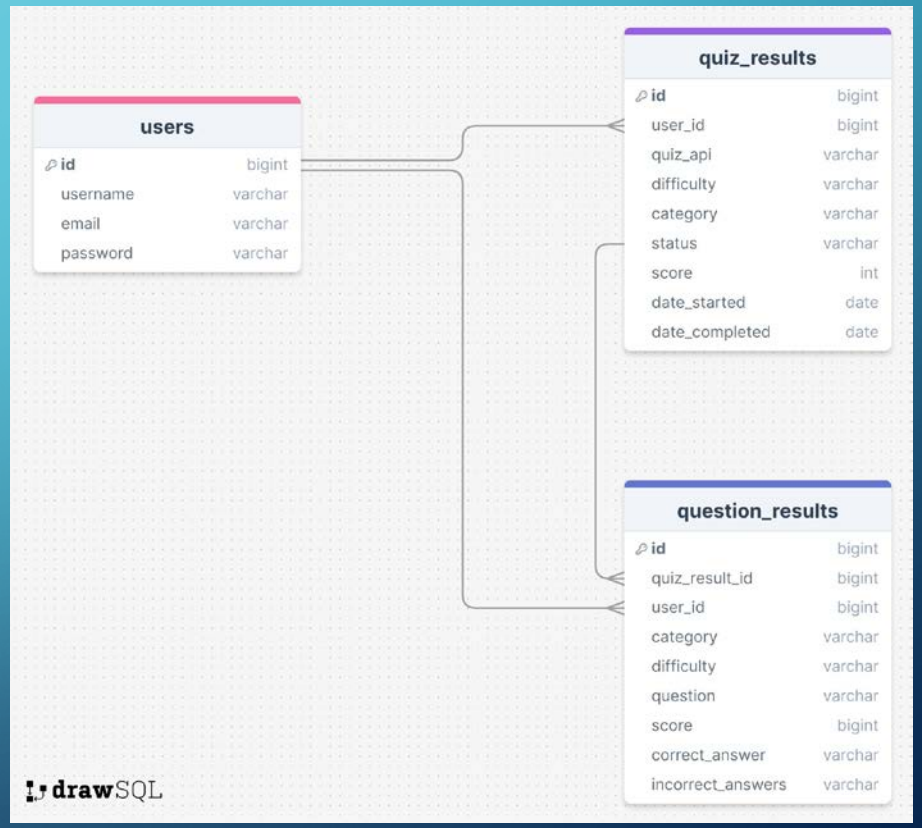
ВИКОРИСТАНІ БІБЛІОТЕКИ CRYSTAL

- 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

АРХІТЕКТУРА ЗАСТОСУНКУ

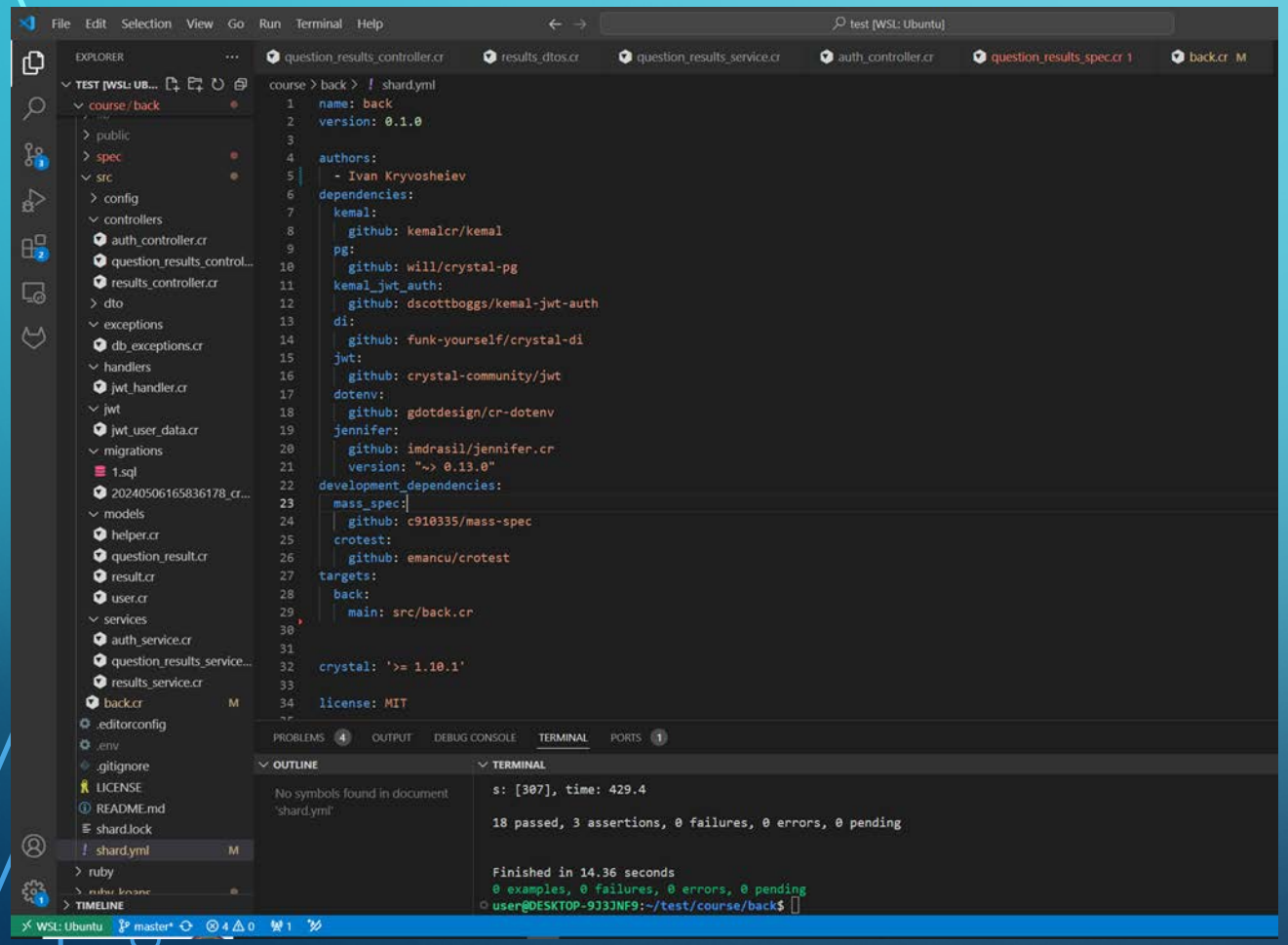


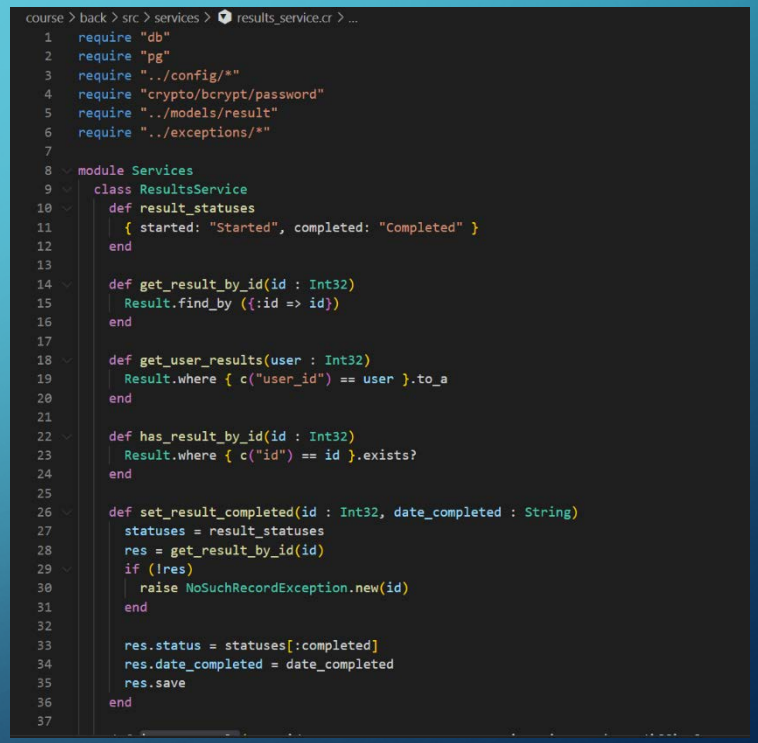
Sequence Diagram



Database diagram

РЕАЛІЗАЦІЯ ДОДАТКУ





РЕАЛІЗАЦІЯ ДОДАТКУ

course > back > src > controllers > question_results_controller.cr > ...

```
require "../services/*"
require "json"
require "../dto/*"
require "../jwt/*"
require "../config/inject"

question_results_service = Inject.resolve(Services::QuestionResultsService)
results_service = Inject.resolve(Services::ResultsService)

get "/questionResults/:quiz_id" do |env|
  user = (env.get "user").as(JWTUserData::User)

  quiz_id = env.params.url["quiz_id"].to_i?
  if (!quiz_id || !results_service.has_result_by_id(quiz_id))
    halt env, status_code: 400, response: "No such quiz result"
  end

  quizResult = results_service.get_result_by_id(quiz_id)
  if (!quizResult || quizResult.user_id != user.id)
    halt env, status_code: 403, response: "Forbidden"
  end

  res = question_results_service.get_results_by_quiz_result_id(quiz_id)
  res.to_json
end

post "/questionResults/:quiz_id" do |env|
  user = (env.get "user").as(JWTUserData::User)

  quiz_id = env.params.url["quiz_id"].to_i?
  if (!quiz_id || !results_service.has_result_by_id(quiz_id))
    halt env, status_code: 400, response: "No such quiz result"
  end

  quizResult = results_service.get_result_by_id(quiz_id)
  if (!quizResult || quizResult.user_id != user.id)
    halt env, status_code: 403, response: "Forbidden"
```

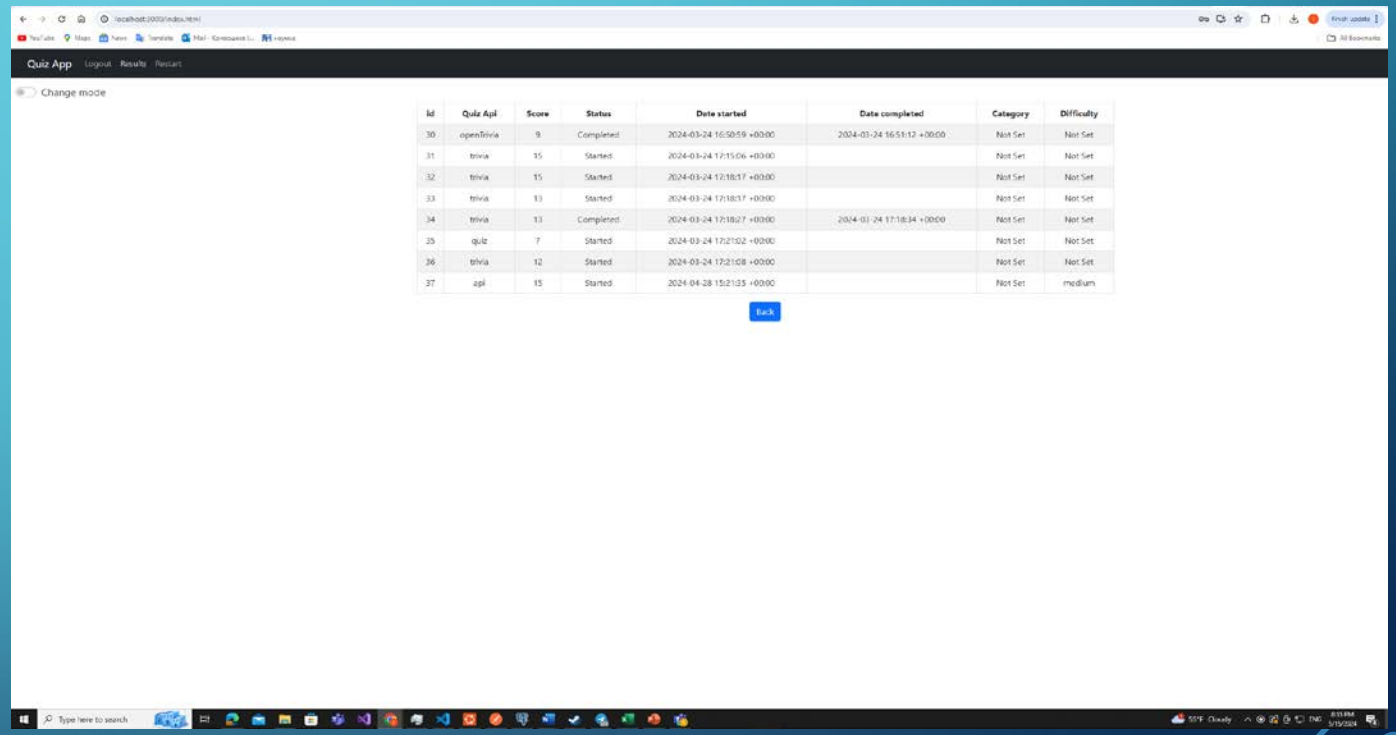
course > back > src > handlers > jwt_handler.cr > ...

```
require "kemal"
require "../jwt/*"

class JWTHandler < Kemal::Handler
  exclude ["/auth/*"], "POST"

  def call(env)
    return call_next(env) if exclude_match?(env)

    if (env.request.method === "OPTIONS")
      env.response.status_code = 200
      env.response.headers.add("Access-Control-Allow-Headers", "authorization")
      return
    end

    begin
      auth_header = env.request.headers["Authorization"]
      if (!auth_header.starts_with?("Bearer "))
        raise "Incorrect Authorization header"
      end
      token = auth_header[7..-1]

      user = JWTUserData.decode_user(token)
      env.set "user", user
    rescue ex
      puts ex.message
      env.response.status_code = 401
      return
    end
    call_next(env)
  end
end
```

РЕЗУЛЬТАТИ

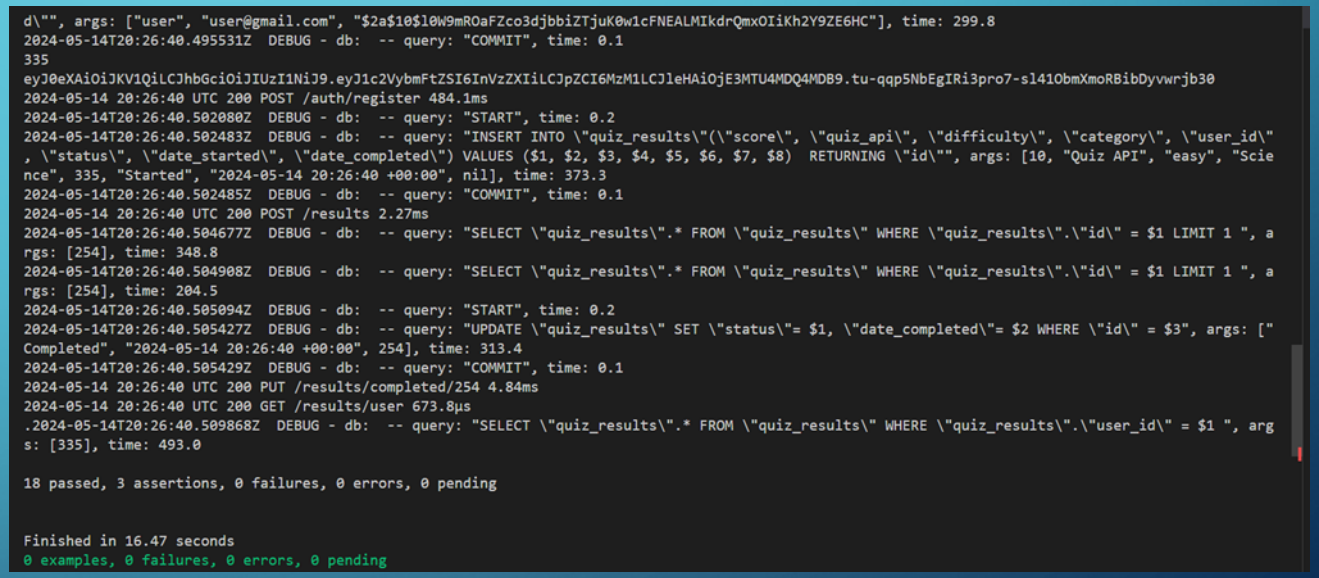
В результаті була розроблена REST API з відповідними ендпоінтами:

- 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.



РЕЗУЛЬТАТИ ТЕСТУВАННЯ

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



РЕЗУЛЬТАТИ ТЕСТУВАННЯ

```
user@DESKTOP-9J3JNF9:~/test$ wrk -t2 -c100 -d30s --latency --header "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InRl
c3QzNTUiLCJpZCI6OCwiZXhwIjoxNzE1ODgzODYwfQ.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