CIKLUM | SPEAKER'S CORNER

AUGUST, 17 | 18:00 (EEST) | ZOOM

GET INTO THE FLOW WITH SPECFLOW: MASTERING BEHAVIOR-DRIVEN DEVELOPMENT IN .NET

VLADYSLAV BABYCH

.NET AUTOMATION TECH LEAD, CIKLUM

Meet Ciklum

CIKLUM

We empower companies to meet their digital initiatives by providing end-to-end software, integration and innovation services

Our services:

- Digital Commerce
- Intelligent Automation
- ITO & Managed Service
- Data & Analytics
- Cloud
- Engineering Services

2002 founded

4000+ professionals

20+ offices

300+ clients

Leading companies choose us:

JUST EAT Takeaway.com · METRO MARKETS · eToro · ZURICH · pay · Greensill · KANTAR RETAIL · dacadoo

Speaker:

Vladyslav Babych

.NET Automation Tech Lead at Ciklum

- 8+ years of professional experience, working both as a .NET developer and a QA Automation engineer
- 3+ years with Ciklum
- Regular speaker in knowledge sharing events and conferences, focused mainly on teaching best practices in .NET test automation

CIKLUM

Agenda

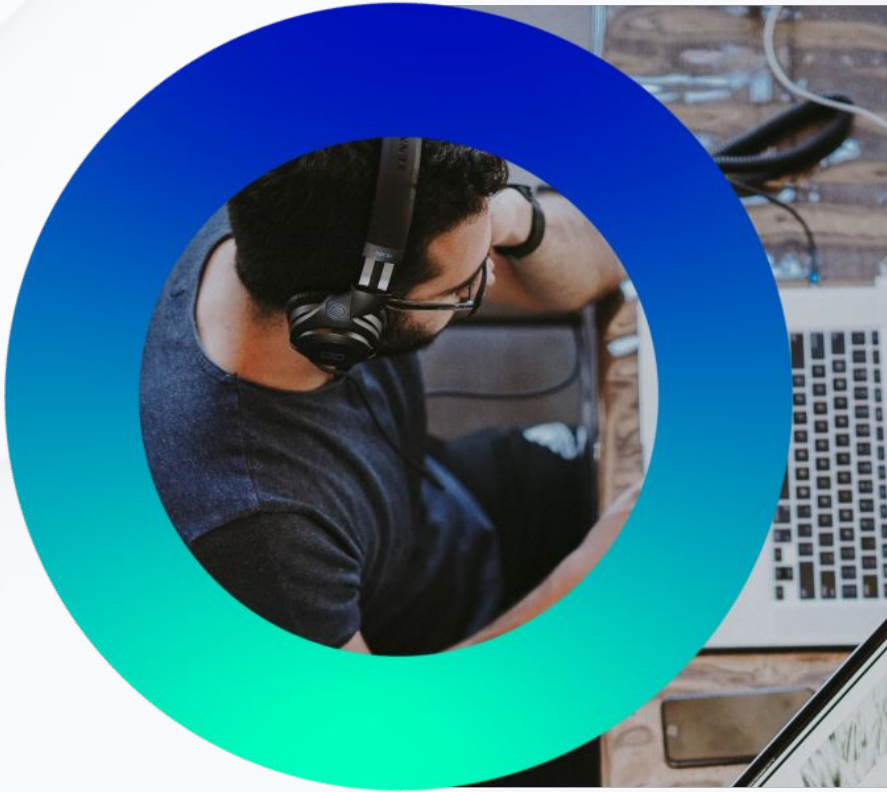
01 **Why BDD?**

02 **Best Practices in Specflow**

03 **More advanced concepts in Specflow**

Why BDD?

01



What is the point of BDD?

- Framework for improved collaboration between developers & stakeholders
- Living Documentation
- Integrates well with Agile framework
- User-centric development approach
- It's not just about testing

Example of BDD test in Gherkin syntax

CIKLUM

Given the user is on the "Reset Password" page.

When the user enters their registered email address and confirms.

Then a confirmation message should be displayed stating "A password reset link has been sent to your email."

When BDD doesn't make much sense

- Simple or short-lived projects
- Lack of stakeholders involvement into development process
- Lack of clear requirements (usually in small or startup projects)
- Late phase of the project when introducing BDD would cause too much overhead to the existing process

CIKLUM

Not a valid reason to dismiss BDD!

Test automation engineers complaining that BDD is hard to maintain or is a “pointless trend” is not a valid reason to dismiss BDD!

Most of the time it’s either prejudice, laziness or resistance to change, and such reaction is unfortunately very common.

CIKLUM

Specflow

Specflow is a leading BDD-Framework for .NET ecosystem.
It’s completely free and open source

- Integrates seamlessly with popular unit testing frameworks in .NET (NUnit, xUnit, MSTest)
- Also provides its own test runner as alternative (SpecFlow+ Runner)
- Maps Gherkin Given/When/Then steps into step definition methods in C#.

Step Definition Example

CIKLUM

Specflow Step

```
When the user logs in using ‘GoogleAuth’
```

Step Definition

```
[When(@"the user logs in using '(.*)'")]
public void WhenTheUserLogsInUsingGoogleAuth(LoginMethod loginMethod)
{
  this.loginStepsDriver.Login(loginMethod);
}
```

Best practices in SpecFlow

02

Use third-person view to describe scenarios

```
Scenario: Successful password reset by user

Given I am on "Reset Password" page
When I provide a valid registered email address
Then I see a confirmation message stating "A password
reset link has been sent to your email."
```

Use third-person view to describe scenarios

CIKLUM

```
Scenario: Successful password reset by user

Given the user is on "Reset Password" page
When the user provides a valid registered email address
Then the user sees a confirmation message stating "A password
reset link has been sent to your email."
```

CIKLUM

Don’t use Imperative syntax (too technical & too much detail)

```
Scenario: Successful password reset by user

Given the user is logged in
When the user clicks on “Profile” button
And the user clicks “Reset password” button
And the user inputs email address “someuser@gmail.com”
And the user clicks “Confirm” button
Then popup “A password reset link has been sent to your
email.” appears
```

Prefer Declarative syntax
(business language)

```
Scenario: Successful password reset by user

Given the user is logged in
When the user is navigated to "Reset Password" page
And the user enters provides registered email address
and confirms
Then password reset confirmation window is displayed
```

Aim for reusability by using parameterized steps

```
Scenario: Successful checkout with credit card
Given the user is logged in using “GoogleAuth"
When the user navigates to products page
And the user adds "Sony TV" product to cart
And user checks out with “credit card" payment
Then the user receives order confirmation via email
```

```
Scenario: Successful checkout with loyalty points payment
Given the user is logged in using "AppleID”
When the user navigates to products page
And the user adds "Samsung TV" product to cart
And user checks out with "loyalty points" payment
Then the user receives order confirmation via email
```

Even better approach – Scenario Outline

```
Scenario Outline: Successful checkout
Given the user is logged in using <authenticationMethod>
authentication method
When the user navigates to products page
And the user adds <product> product to cart
And user checks out using <paymentType> payment method
Then the user receives order confirmation via email

Examples:
  | authenticationMethod | product    | paymentType    |
  | AppleID              | Samsung TV | loyalty points |
  | GoogleAuth           | Sony TV    | credit card    |
```

Never chain scenarios together

- Each scenario should be independently executable from other scenarios
- If you need to do common setup/cleanup, use hooks (will be covered in next chapter)

Never chain scenarios together

```
Scenario: 1 Create Order
Given the user is logged in
When the user navigates to products page
And the user adds "Sony TV" product to cart
And user checks out with “credit card" payment
Then the user receives order confirmation via email
```

```
Scenario: 2 Cancel Order
Given the user is logged in
When the user navigates to orders page
And the user selects "Sony TV" order
And user cancels the order
Then the user receives order cancellation via email
```

Never chain scenarios together

CIKLUM

```
@SetupOrder
Scenario: Cancel Order
   Given the user is logged in
   When the user navigates to orders page
   And the user selects "Sony TV" order
   And user cancels the order
   Then the user receives order cancellation via email
```

CIKLUM

More tips to properly isolate your scenarios

- Try not to reuse existing entities (e.g. products), always create unique set of data for each test if possible
- If tests rely on specific app configuration, inject it at runtime or before whole test run
- Clean up your data (databases, left over files etc.)

Properly isolated Specflow test e.g.

CIKLUM

```
@BeforeScenario

Username = CreateRandomString() + "_automation_user@gmail.com"
Product = CreateRandomString() + "headphones"

@SetupUser(Username).WithLoyaltyPointsBalance('100')
@SetupProduct('Sony MDR 750 headphones').WithPrice('100')
@EnableLoyaltyPointsPayment


Scenario: After login, user navigates to shopping cart, adds a product and checks out an order.

Given user 'Username' is logged in
When user adds product 'Product' to the cart
And user checks out and pays with '100' loyalty points
Then checkout is successful
And total price of an order is '100$'
And user order list has 1 order with title 'Product'

@AfterScenario

@CleanupUser(Username)
@CleanupProduct(Product)
@DisableLoyaltyPointsPayment
```

Advanced Specflow Concepts

03

Hooks

CIKLUM

Should be used to perform additional automation logic at specific times, such as any setup/cleanup required prior to executing a scenario. Available scopes:

[BeforeTestRun]/[AfterTestRun]

[BeforeFeature]/[AfterFeature]

[BeforeScenario]/[AfterScenario]

[BeforeStep]/[AfterStep]

Hooks (Example)

CIKLUM

On test failure, get event logs and messages from db and output in test result

```
[AfterStep]
public void WrapTestFailure(ScenarioContext scenarioContext, DatabaseContext context)
{
  Exception exception = scenarioContext.TestError;
  if(exception == null)
     return;
  List<Guid> correlationIds = GetCorrelationIdEntries(scenarioContext);

  throw new Exception(
     $"Test failure: Original Exception: {exception.GetType()} {exception.Message}\n
Event logs and messages for id: {correlationId}:
{Utils.GetAllExecutionMessagesAndEventLogs(correlationIds, context)}");
}
```

Scenario/Feature Context

CIKLUM

A mechanism to share data between steps in a single scenario. Facilitates cleaner, more modular step definitions by eliminating the need for global/shared variables.

Key Properties of Scenario Context:

- ScenarioContext.Current: Access the current running scenario.
- ScnearioContext.ScenarioInfo: Retrieve details like title, tags, or description.
- Add & TryGetValue: Store and retrieve data.

Best practices for Scenario Context

CIKLUM

- Only use scenario context in step definition classes, don’t propagate it deeper. Only specflow binding classes should “know about it”
- Scenario Context has limitations and is best used for sharing simple data/variables
- If you need to share complex data/objects between steps, it’s better to create a custom object and inject it with DI/Repository pattern

Step Argument Transformations

CIKLUM

- Allows text in steps to be converted into complex types or perform custom transformations on argument values.
- Simplifies step definitions by handling repetitive parsing logic.
- Usage: Apply `[StepArgumentTransformation]` attribute to a method that converts a string or table to a desired type.

Example of Step Argument Transformation

CIKLUM

Specflow Step:

```
Given the user details:
 | Name  | Age | Email             |
 | Alice | 28  | alice@example.com |
```

Step argument transformation:

```
[StepArgumentTransformation]
public User TransformTableToUser(Table table)
{
  return table.CreateInstance<User>();
}
```

Usage:

```
[Given(@"the user details are:")]
public void GivenTheUserDetailsAre(User user)
{
  // Now, you directly work with the 'User'
object
}
```

Handling Relative Dates in Specflow

CIKLUM

- Use StepArgumentTransformations for that
- Avoid mathematical operations (e.g. “Today - 1 day”)
- Define date periods according to your business requirements (e.g. Today, Yesterday, MonthAgo)

Handling Relative Dates in Specflow

CIKLUM

Specflow Step

```
When the order was completed 'after return policy period expired'
Then the order is not eligible for return policy
```

Step argument transformation

```
[StepArgumentTransformation("After return policy period expired")]
public DateTime TransformToDayMoreThanTwoWeeksAgo()
{
  // current return policy is 14 days, can be adjusted if requirements change
  return DateTime.Today.AddDays(-14);
}
```

Usage

```
[When(@"the order was completed '(.*)'")]
public void WhenTheOrderWasCompleted(DateTime date)
{
  // your logic
}
```

Use Driver pattern for your steps

- Provides an additional layer between step definitions and automation code
- Keeps steps short and easy to read/understand
- Can easily combine with Strategy pattern to allow different behaviors of the same steps during runtime
- Allows to reuse the same driver methods across multiple step definitions or even different scenarios

https://docs.specflow.org/projects/specflow/en/latest/Guides/DriverPattern.html

Driver Pattern Example
(with a sprinkle of Strategy pattern)

CIKLUM

Use case:
User can log into your application using several methods, e.g. AppleID, GoogleAuth etc. We need to test login logic

In order to implement driver pattern properly, we first need to create a common login interface:

```
public interface ILoginDriver
{
  public void Login();
}
```

Driver Pattern Example
(with a sprinkle of Strategy pattern)

CIKLUM

Then we create implementations for both supported login methods (strategy)

```
public class GoogleAuthDriver : ILoginDriver
{
  public void Login()
  {
     // login using google auth, you could get login info such as email from ScenarioContext
or just pass it as variables
  }
}


public class AppleIdAuthDriver : ILoginDriver
{
  public void Login()
  {
     // login using apple id, you could get login info such as email from ScenarioContext or
just pass it as variables
  }
}
```

Driver Pattern Example
(with a sprinkle of Strategy pattern)

CIKLUM

Then we can create a simple Factory class which will return correct driver to us depending on login method:

```
public class LoginDriverSimpleFactory
{
  public ILoginDriver GetLoginDriver(LoginMethod loginMethod)
  {
     switch (loginMethod)
     {
        case LoginMethod.GoogleAuth:
           return new GoogleAuthDriver();
        case LoginMethod.AppleId:
           return new AppleIdAuthDriver();
     }
  }
}
```

Driver Pattern Example
(with a sprinkle of Strategy pattern)

CIKLUM

And finally the actual step driver:

```
public class LoginStepDriver
{
  public void Login(LoginMethod loginMethod)
  {
     var loginDriver = new LoginDriverSimpleFactory().GetLoginDriver(loginMethod);
     loginDriver.Login();
  }
}
```

And the step definition itself:

```
[When(@"the user logs in using '(.*)'")]
public void WhenTheUserLogsInUsingGoogleAuth(LoginMethod loginMethod)
{
  this.loginStepsDriver.Login(loginMethod);
}
```

CIKLUM

Driver Pattern Example
(with a sprinkle of Strategy pattern)

Here is how our specflow tests look like. Notice, that there is no duplication of step definitions and steps are also reusable and flexible.

```
Scenario: Login using Google Authentication
When the user logs in using 'GoogleAuth'
Then login is successful


Scenario: Login using Apple Authentication
When the user logs in using 'AppleAuth'
Then login is successful
```

If we want to introduce a new login method in the future, we just need to create a new driver implementation. Changes to the code are minimal, and existing specflow tests/step definitions are untouched.

CIKLUM

Specflow is not hard to maintain **if you do it right**

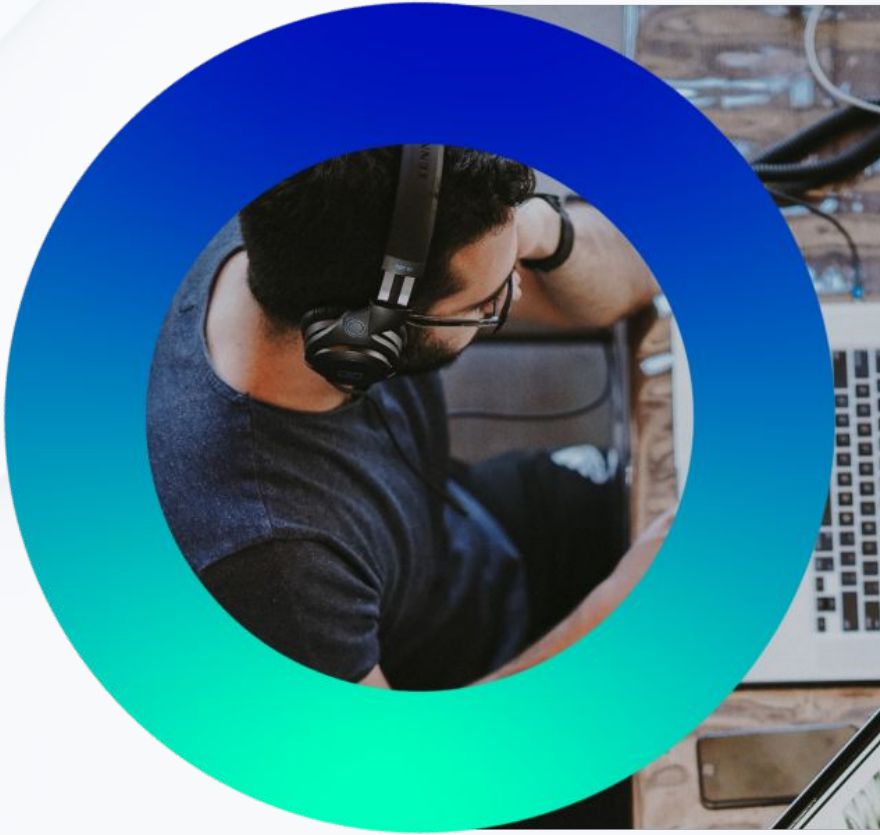
Remember that common complaint that specflow tests are harder to maintain than regular tests written in code?

I would bet that in 90%+ of those cases best practices such as driver pattern or test isolation is not used, or people are not even aware of it.

Specflow is not the problem, it's **YOU**!

Bonus Section:
Integrate Specflow tests with Azure Test Plans

04



CIKLUM

Integrating with Azure Test Plans

If you're working in Azure environment, it's easy to synchronize your Specflow tests with Azure Test Plans using SpecSync tool.

It will seamlessly integrate with Azure Test Plans and convert specflow steps into Azure test cases and update their execution status after each pipeline run.

This is great for automated reporting to your stakeholders.

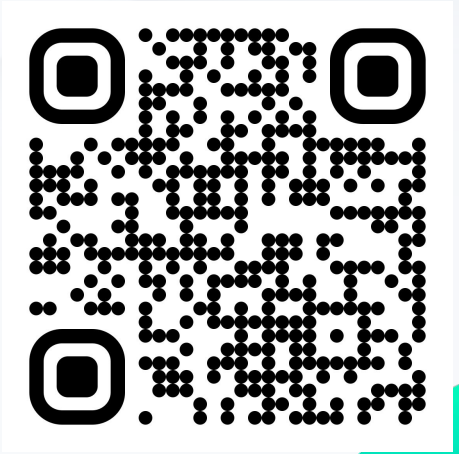
SpecSync tool (requires paid licence):
https://www.specsolutions.eu/specsync/

CIKLUM

Specflow Living Docs

LivingDocs is an extension that allows to display your Specflow tests and their results within your browser.

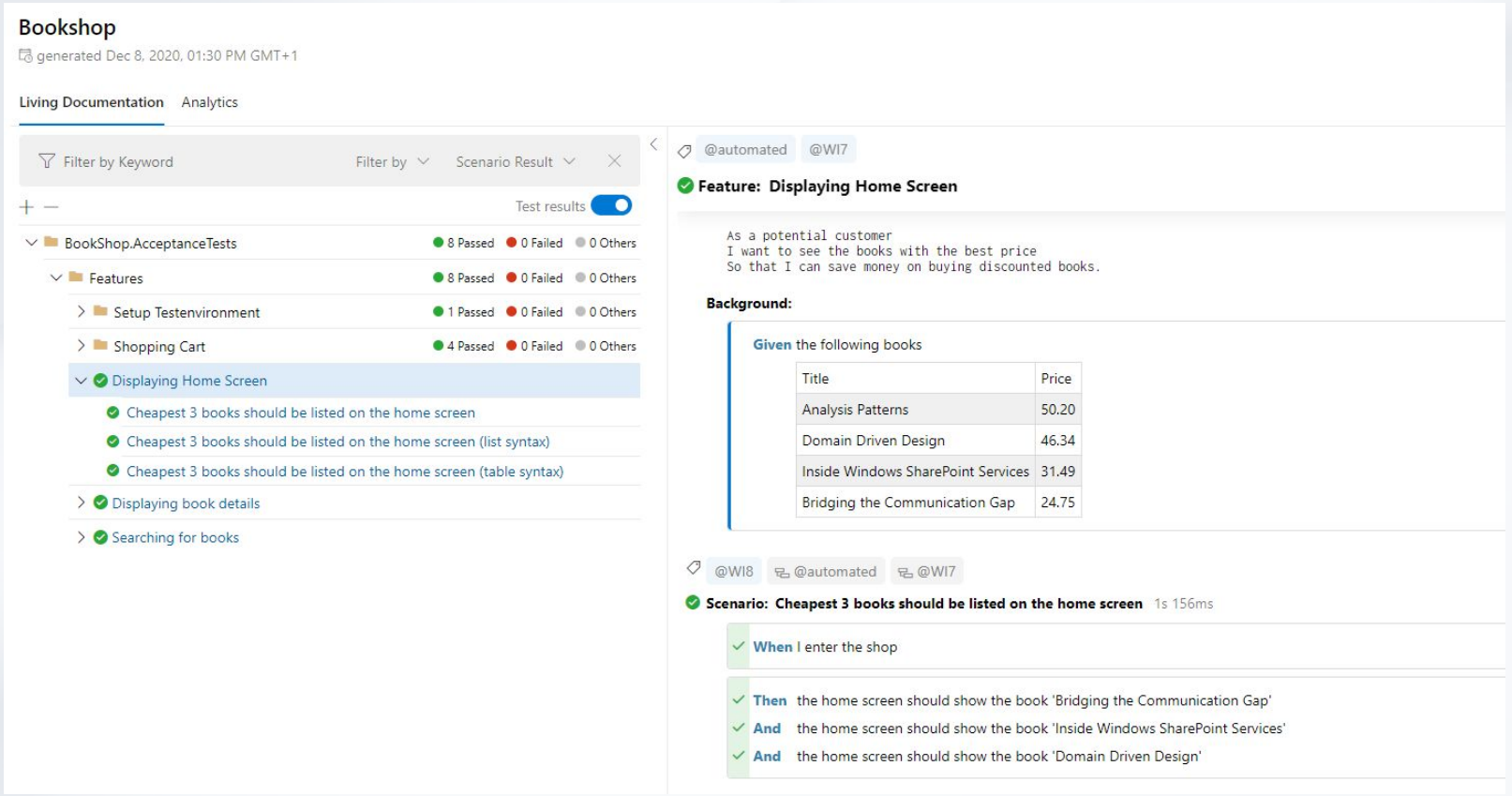
It functions as a living and dynamically changing documentation for your project.



https://specflow.org/tools/living-doc/

Specflow Living Docs Example

CIKLUM



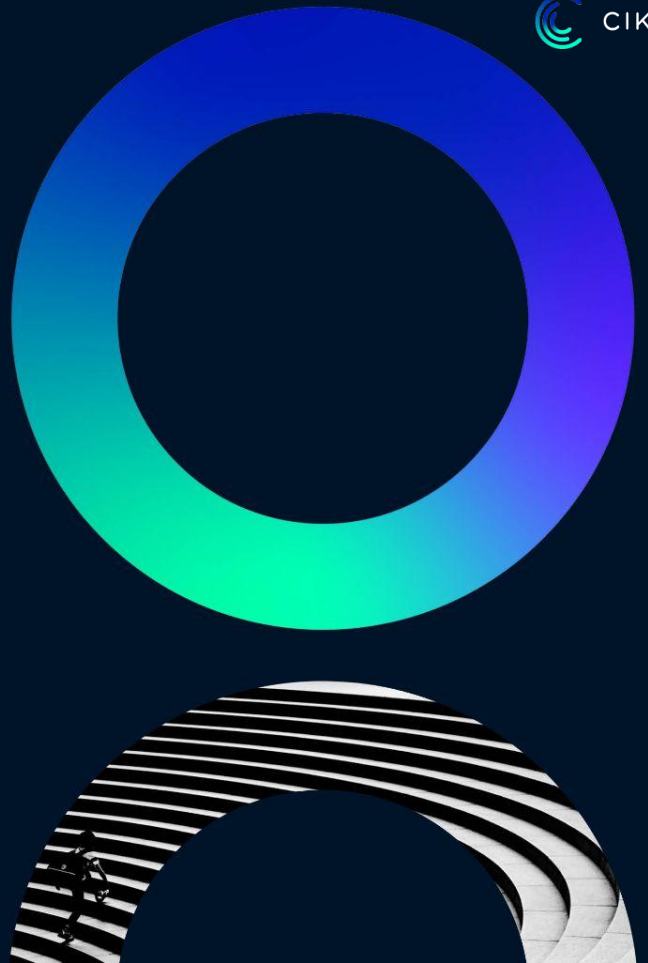
Conclusion

Q&A

Vladyslav Babych
vlbab@ciklum.com
www.ciklum.com

CIKLUM

Explore career opportunities at Ciklum



Thank you!

Vladyslav Babych
vlbab@ciklum.com
www.ciklum.com