THE H@CK SUMMIT

wcześniej WHAT THE H@CK

C:\>

Dobre zasady code review

Adrian Sroka
Architekt, Konsultant, British Council

www.thehacksummit.com 5/11/2021 online

ORGANIZATORZY: Academic Partners FUNDACJA

Bezpieczeństwo podczas dewelopmentu

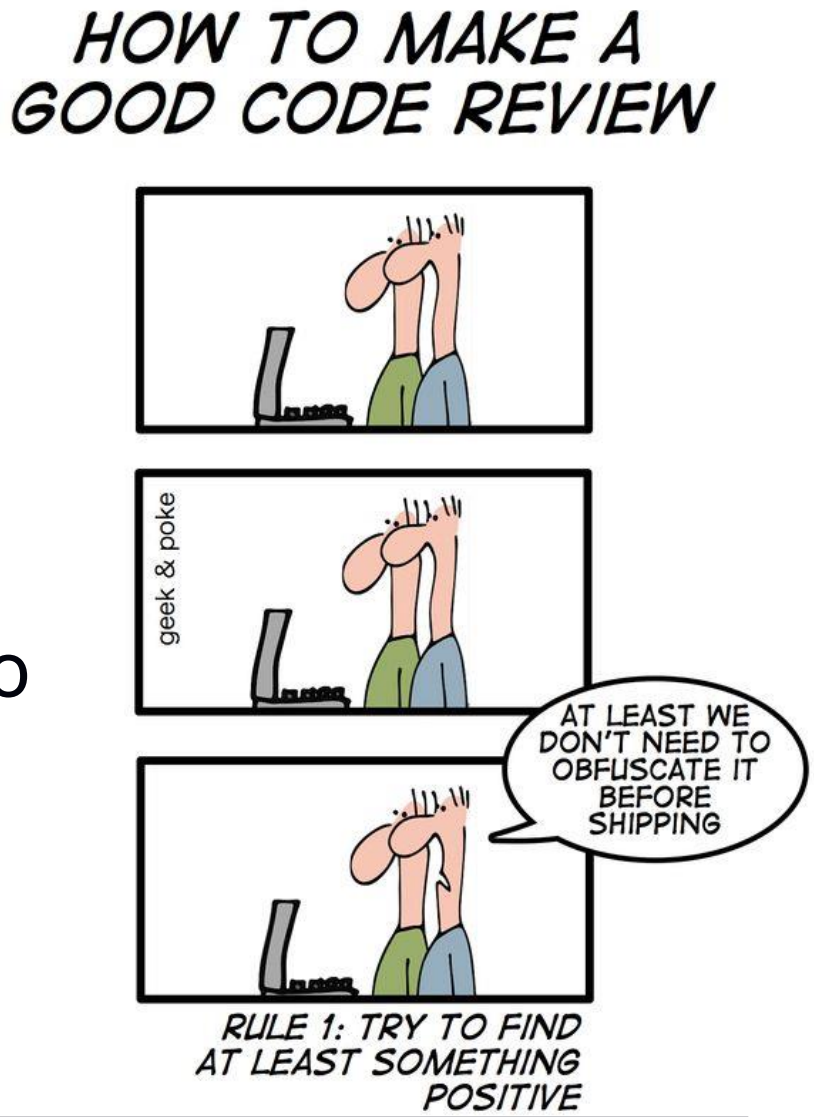
- Threat modeling
- Code review
- Testy
  - Automatyczne
  - Manualne

Co złego może się kryć w kodzie?

celowe	• time bomb • backdoor • złośliwy skrypt
przypadkowe	• najzwyklejszy błąd

Przyrostowe Code Review

- stosunkowo małe (ale nie zawsze)
- nastawione na konkretne funkcjonalności
- gate keeper - pilnują żeby nie dodać na Produkcję czegoś niebezpiecznego



Przejdźmy do rzeczy

Na co większość ludzi zwraca uwagę podczas Code Review?

Instrukcja obsługi

- Zagadka i odpowiedź
- Coraz trudniejsze przykłady
- Może się wydawać łatwe
- Będzie szybko ☺

```
public IEnumerable<User> GetData(string username)
{
    using (var connection = GetConnection())
    {
        return connection.Query<User>("SELECT Email, IsActive, Role " +
            "FROM Users " +
            "WHERE Username = " + username).ToList();
    }
}

public OrderPrice GetOrderPrice(OrderDto order)
{
    using (var connection = GetConnection())
    {
        return connection.Query<OrderPrice>($@"SELECT
                    or.{nameof(Order.Price)} as [{nameof(OrderPrice.Price)}]
                FROM [{nameof(Customer)}] cu
                LEFT JOIN [{nameof(Order)}] or
                    ON cu.{nameof(Customer.Id)} = or.{nameof(Order.CustomerId)}
                WHERE cu.{nameof(Customer.Phone)} = {order.CustomerPhone}
                   AND or.{nameof(Order.OrderNumber)} = @{nameof(order.OrderNumber)}", order).Single();
    }
}
```

PROBLEM

SQL Injection

```
public IEnumerable<User> GetData(string username)
{
    using (var connection = GetConnection())
    {
        return connection.Query<User>("SELECT Email, IsActive, Role " +
            "FROM Users " +
            "WHERE Username = " + username).ToList();
    }
}

public OrderPrice GetOrderPrice(OrderDto order)
{
    using (var connection = GetConnection())
    {
        return connection.Query<OrderPrice>($@"SELECT
                    or.{nameof(Order.Price)} as [{nameof(OrderPrice.Price)}]
                FROM [{nameof(Customer)}] cu
                LEFT JOIN [{nameof(Order)}] or
                    ON cu.{nameof(Customer.Id)} = or.{nameof(Order.CustomerId)}
                WHERE cu.{nameof(Customer.Phone)} = {order.CustomerPhone}
                    AND or.{nameof(Order.OrderNumber)} = @{nameof(order.OrderNumber)}", order).Single();
    }
}
```

Klejenie warunków w zapytaniu

Nie skorzystanie z QueryParams

```
public class CustomerValidator : AbstractValidator<Customer>
{
    public CustomerValidator()
    {
        RuleFor(x => x.FirstName).NotEmpty();
        RuleFor(x => x.LastName).NotEmpty().WithMessage("Please specify a last name");
        RuleFor(x => x.Email).EmailAddress();
        RuleFor(x => x.Address).Length(20, 250).Must(NotHaveInvalidCharacters);
    }

    private bool NotHaveInvalidCharacters(string text)
    {
        return Regex.IsMatch(text, @"[<>\/=]");
    }
}

public class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string Address { get; set; }
    public string Postcode { get; set; }
}
```

PROBLEM

Brak walidacji danych od użytkownika

```
public class CustomerValidator : AbstractValidator<Customer>
{
    public CustomerValidator()
    {
        RuleFor(x => x.FirstName).NotEmpty();
        RuleFor(x => x.LastName).NotEmpty().WithMessage("Please specify a last name");
        RuleFor(x => x.Email).EmailAddress();
        RuleFor(x => x.Address).Length(20, 250).Must(NotHaveInvalidCharacters);
    }

    private bool NotHaveInvalidCharacters(string text)
    {
        return Regex.IsMatch(text, @"[<>\/=]");
    }
}

public class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string Address { get; set; }
    public string Postcode { get; set; }
}
```

Powinniśmy stosować whitelisty

```
public string CalculateHash(string input)
{
    using (SHA1Managed sha1 = new SHA1Managed())
    {
        var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));
        var sb = new StringBuilder(hash.Length * 2);

        foreach (byte b in hash)
        {
            sb.Append(b.ToString("X2"));
        }

        return sb.ToString();
    }
}
```

PROBLEM

Użycie niebezpiecznych algorytmów

```
public string CalculateHash(string input)
{
    using (SHA1Managed sha1 = new SHA1Managed())
    {
        var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));
        var sb = new StringBuilder(hash.Length * 2);

        foreach (byte b in hash)
        {
            sb.Append(b.ToString("X2"));
        }

        return sb.ToString();
    }
}
```

SHA1 jest skompromitowane

To za proste

Przecież to można załatwić statyczną analizą kodu

```
{
  "name": "testapp",
  "version": "1.0.0",
  "dependencies": {
    "bootstrap": "^4.1.3",
    "jquery": "^3.4.1",
    "merge": "^1.2.1",
    "oidc-client": "^1.9.0",
    "react": "^16.0.0",
    "react-dom": "^16.0.0",
    "react-router-bootstrap": "^0.25.0",
    "react-router-dom": "^5.1.2",
    "react-scripts": "^3.4.1",
    "reactbootstrap": "^8.4.1",
    "rimraff": "^2.6.2"
  }
}
```

PROBLEM

Niepoprawne wersje bibliotek

```
{
  "name": "testapp",
  "version": "1.0.0",
  "dependencies": {
    "bootstrap": "^4.1.3",
    "jquery": "^3.4.1",
    "merge": "^1.2.1",
    "oidc-client": "^1.9.0",
    "react": "^16.0.0",
    "react-dom": "^16.0.0",
    "react-router-bootstrap": "^0.25.0",
    "react-router-dom": "^5.1.2",
    "react-scripts": "^3.4.1",
    "reactbootstrap": "^8.4.1",
    "rimraff": "^2.6.2"
  }
}
```

```
{
  "name": "testapp",
  "version": "1.0.0",
  "dependencies": {
    "bootstrap": "^4.1.3",
    "jquery": "^3.4.1",
    "merge": "^1.2.1",
    "oidc-client": "^1.9.0",
    "react": "^16.0.0",
    "react-dom": "^16.0.0",
    "react-router-bootstrap": "^0.25.0",
    "react-router-dom": "^5.1.2",
    "react-scripts": "^3.4.1",
    "reactstrap": "^8.4.1",
    "rimraf": "^2.6.2"
  }
}
```

Literówki w nazwach paczek

```
public async Task<string> GetPayPalSecret()
{
    const string secretName = "payPalSecret";
    SecretClient client = GetKeyVaultClient();

    var secret = await client.GetSecretAsync(secretName);
    return secret.Value.Value;
}

private static SecretClient GetKeyVaultClient()
{
    var kvUri = $"https://paymentGateway.vault.azure.net";
    return new SecretClient(new Uri(kvUri), new ClientSecretCredential("testTenant", "paymentGateway", "FzJ&W@!8ZEXspQt5sCWSdgo8uU"));
}
```

PROBLEM

Dane wrażliwe zaszyte w kodzie

```
public async Task<string> GetPayPalSecret()
{
    const string secretName = "payPalSecret";
    SecretClient client = GetKeyVaultClient();

    var secret = await client.GetSecretAsync(secretName);
    return secret.Value.Value;
}

private static SecretClient GetKeyVaultClient()
{
    var kvUri = $"https://paymentGateway.vault.azure.net";
    return new SecretClient(new Uri(kvUri), new ClientSecretCredential("testTenant", "paymentGateway", "FzJ&W@!8ZEXspQt5sCWSdgo8uU"));
}
```

Klucz ukryty w kodzie

```
public string GenerateNewApiKey(string clientId)
{
    return $"PROD_@{clientId}_@{(GetRandom(numberCount: 24))}";
}

public int GetRandom(int numberCount)
{
    var minValue = (long)Math.Pow(10, numberCount);
    var maxExclusiveValue = (long)Math.Pow(10, numberCount + 1);
    if (minValue >= maxExclusiveValue)
        throw new ArgumentOutOfRangeException("minValue must be lower than maxExclusiveValue");

    long diff = (long)maxExclusiveValue - minValue;
    long upperBound = uint.MaxValue / diff * diff;

    uint ui;
    do
    {
        ui = GetRandomUInt();
    } while (ui >= upperBound);
    return (int)(minValue + (ui % 10));
}
```

PROBLEM

Dane wrażliwe zaszyte w kodzie

```
public string GenerateNewApiKey(string clientId)
{
    return $"PROD_@{clientId}_@{(GetRandom(numberCount: 24))}";
}

public int GetRandom(int numberCount)
{
    var minValue = (long)Math.Pow(10, numberCount);
    var maxExclusiveValue = (long)Math.Pow(10, numberCount + 1);
    if (minValue >= maxExclusiveValue)
        throw new ArgumentOutOfRangeException("minValue must be lower than maxExclusiveValue");

    long diff = (long)maxExclusiveValue - minValue;
    long upperBound = uint.MaxValue / diff * diff;

    uint ui;
    do
    {
        ui = GetRandomUInt();
    } while (ui >= upperBound);
    return (int)(minValue + (ui % 10));
}
```

Przewidywalne klucze

Zmniejszona entropia liczby losowej

```
public async Task<string> GetPayPalSecret()
{
    const string secretName = "payPalSecret";
    var cacheValue = _cacheManager.Get(secretName);
    if (!string.IsNullOrEmpty(cacheValue))
    {
        return cacheValue;
    }
    SecretClient client = GetKeyVaultClient();

    var secret = await client.GetSecretAsync(secretName);
    _cacheManager.Set(secretName, secret.Value.Value, TimeSpan.FromDays(7));
    return secret.Value.Value;
}
```

PROBLEM

Brak wsparcia szybkiej podmiany sekretów

```
public async Task<string> GetPayPalSecret()
{
    const string secretName = "payPalSecret";
    var cacheValue = _cacheManager.Get(secretName);
    if (!string.IsNullOrEmpty(cacheValue))
    {
        return cacheValue;
    }
    SecretClient client = GetKeyVaultClient();

    var secret = await client.GetSecretAsync(secretName);
    _cacheManager.Set(secretName, secret.Value.Value, TimeSpan.FromDays(7));
    return secret.Value.Value;
}
```

Długi czas cache'owania

```
public void ExecuteQuery(string sqlQuery)
{
    using (SqlConnection connection = GetDbConnection())
    {
        SqlCommand command = new SqlCommand(sqlQuery, connection);
        try
        {
            command.Connection.Open();
            command.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
            throw new Exception($"The SQL query @{sqlQuery} failed due to the @{ex.Message}");
        }
    }
}

public Result Login(LoginDto loginDto)
{
    _validator.Validate(loginDto);

    var loginDataCorrect = _authenticationService.IsPasswordCorrect(loginDto.Username, loginDto.Password);
    if (!loginDataCorrect)
    {
        return Result.Error($"The password @{loginDto.Password} is incorrect. Please try different one.");
    }
    return Result.Success();
}
```

PROBLEM

Pokazywanie sekretów w błędach

```
public void ExecuteQuery(string sqlQuery)
{
    using (SqlConnection connection = GetDbConnection())
    {
        SqlCommand command = new SqlCommand(sqlQuery, connection);
        try
        {
            command.Connection.Open();
            command.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
            throw new Exception($"The SQL query @{sqlQuery} failed due to the @{ex.Message}");
        }
    }
}
```

Wrażliwe dane

```
public Result Login(LoginDto loginDto)
{
    _validator.Validate(loginDto);

    var loginDataCorrect = _authenticationService.IsPasswordCorrect(loginDto.Username, loginDto.Password);
    if (!loginDataCorrect)
    {
        return Result.Error($"The password @{loginDto.Password} is incorrect. Please try different one.");
    }
    return Result.Success();
}
```

```
public class UserController : Controller
{
    public UserService _userService { get; private set; }

    [HttpGet]
    [Route("api/user/{userId}")]
    public UserEntity GetUser(long userId)
    {
        return _userService.GetUser(userId);
    }
}

public class UserEntity
{
    public long Id { get; set; }
    public string UserName { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public Role Role { get; set; }
    public DateTime LastLoggedOn { get; set; }
}
```

PROBLEM

Niepotrzebne zwracanie wrażliwych danych

```
public class UserController : Controller
{
    public UserService _userService { get; private set; }

    [HttpGet]
    [Route("api/user/{userId}")]
    public UserEntity GetUser(long userId)
    {
        return _userService.GetUser(userId);
    }
}

public class UserEntity
{
    public long Id { get; set; }
    public string UserName { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public Role Role { get; set; }
    public DateTime LastLoggedOn { get; set; }
}
```

Zwracanie modelu z bazy danych

```
public class PermissionCheck : Controller
{
    Services

    [RequireRole("reader")]
    [HttpGet]
    [Route("api/order/{orderId}")]
    public Order GetOrders(int orderId)
    {
        return _orderService.Get(orderId);
    }
}

public Order Get(int orderId)
{
    var permissions = _userContext.GetPermissions();
    var order = _orderRepository.Get(orderId);

    if (order.CountryCode != permissions.CountryCode
        && order.BusinessBranch != permissions.BusinessBranch)
    {
        return null;
    }
    return order;
}
```

PROBLEM

Błędy w weryfikacji zakresu uprawnień

```
public class PermissionCheck : Controller
{
    Services

    [RequireRole("reader")]
    [HttpGet]
    [Route("api/order/{orderId}")]
    public Order GetOrders(int orderId)
    {
        return _orderService.Get(orderId);
    }
}

public Order Get(int orderId)
{
    var permissions = _userContext.GetPermissions();
    var order = _orderRepository.Get(orderId);

    if (order.CountryCode != permissions.CountryCode
        && order.BusinessBranch != permissions.BusinessBranch)
    {
        return null;
    }
    return order;
}
```

Pomyłka w łączeniu warunków

```
public class StrongHashAlgorithm : IHash
{
    public string CalculateHash(string input)
    {
        var bytes = Encoding.UTF8.GetBytes(input);
        long hash = bytes[0];
        for (int i = 1; i < bytes.Length; i++)
        {
            hash += (i+1)*bytes[i];
        }
        return hash.ToString();
    }
}
```

PROBLEM

Własna implementacja algorytmów kryptograficznych

```
public class StrongHashAlgorithm : IHash
{
    public string CalculateHash(string input)
    {
        var bytes = Encoding.UTF8.GetBytes(input);
        long hash = bytes[0];
        for (int i = 1; i < bytes.Length; i++)
        {
            hash += (i+1)*bytes[i];
        }
        return hash.ToString();
    }
}
```

```
public void SendMail(Mail mail)
{
    try
    {
        var client = MailClient.Create(_configuration.MailServer.Url, _configuration.MailServer.ApiKey);
        client.SendMail(mail);
    }
    catch (Exception ex)
    {
        _log.Error(ex.Message);
    }
}
```

```
public class MailClient
{
    Hidden

    public static MailClient Create(string url, string apiKey)
    {
        try
        {
            var client = new MailClient(url, apiKey);
            client.Connect();
            return client;
        }
        catch (Exception)
        {
            throw new MailException($"Cannot connect to @{url}:@{apiKey}.");
        }
    }
}
```

PROBLEM

Logowanie wrażliwych danych

```
public void SendMail(Mail mail)
{
    try
    {
        var client = MailClient.Create(_configuration.MailServer.Url, _configuration.MailServer.ApiKey);
        client.SendMail(mail);
    }
    catch (Exception ex)
    {
        _log.Error(ex.Message);
    }
}
```

```
public class MailClient
{
    Hidden

    public static MailClient Create(string url, string apiKey)
    {
        try
        {
            var client = new MailClient(url, apiKey);
            client.Connect();
            return client;
        }
        catch (Exception)
        {
            throw new MailException($"Cannot connect to @{url}:@{apiKey}.");
        }
    }
}
```

```
public double CalculateDiscount(OrderDto order)
{
    var discountLevel = _customerService.GetDiscountLevel(order.CustomerId);
    var discount = _discountService.Calculate(discountLevel);

    if (order.CountryId != _userContext.AllowedCountryId)
    {
        throw new NotAllowedException("You don't have an access to do this.");
    }

    return discount;
}
public object GetDiscountLevel(object customerId)
{
    // ...
    if (!CustomerExists(customerId))
    {
        throw new NotExistException($"The customer @{customerId} not exists.");
    }

    // ...

    return null;
}
```

PROBLEM

Logika biznesowa przed uprawnieniami

```
public double CalculateDiscount(OrderDto order)
{
    var discountLevel = _customerService.GetDiscountLevel(order.CustomerId);
    var discount = _discountService.Calculate(discountLevel);

    if (order.CountryId != _userContext.AllowedCountryId)
    {
        throw new NotAllowedException("You don't have an access to do this.");
    }

    return discount;
}
public object GetDiscountLevel(object customerId)
{
    // ...
    if (!CustomerExists(customerId))
    {
        throw new NotExistException($"The customer @{customerId} not exists.");
    }

    // ...

    return null;
}
```

Zdradza istnienie użytkownika

```
public void CreateAccont(AccountData accountData)
{
    var createdAccount = _accountRepostory.Create(accountData.Email, accountData.PhoneNumber);
    _mailClient.Send(accountData.PhoneNumber, $"You succesfully created an account. " +
        $"Your login is @{createdAccount.Email}, your password is @{createdAccount.Password}");
}
```

PROBLEM

Wysyłka wrażliwych danych poza system

```
public void CreateAccont(AccountData accountData)
{
    var createdAccount = _accountRepostory.Create(accountData.Email, accountData.PhoneNumber);
    _mailClient.Send(accountData.PhoneNumber, $"You succesfully created an account. " +
        $"Your login is @{createdAccount.Email}, your password is @{createdAccount.Password}");
}
```

Pójdźmy jeszcze o krok dalej

Tym razem coś, co jeszcze trudniej znaleźć. Coś, co wymaga łączenia faktów

```
public void MakePayment(OrderDto orderDto)
{
    _payPalService.CreatePayment(new PayPalPayment()
    {
        Amount = orderDto.Amount,
        TaxAmount = orderDto.TaxAmount,
        PayerData = new Payer()
        {
            Name = orderDto.CustomerName,
            AddressDto = orderDto.CustomerAddress,
            PhoneNumber = orderDto.CustomerPhone
        }
    });
}
```

PROBLEM

Wysyłanie wrażliwych danych do zewnętrznych serwisów

```
public void MakePayment(OrderDto orderDto)
{
    _payPalService.CreatePayment(new PayPalPayment()
    {
        Amount = orderDto.Amount,
        TaxAmount = orderDto.TaxAmount,
        PayerData = new Payer()
        {
            Name = orderDto.CustomerName,
            AddressDto = orderDto.CustomerAddress,
            PhoneNumber = orderDto.CustomerPhone
        }
    });
}
```

RODO/GDPR

```
public async void UpdateAddress(AddressDto addressDto)
{
    await _messageQueue.Publish(new ShippingAddressChangedMessage(addressDto));
}

// one service
public class UpdateCustomerData : IConsumer<ShippingAddressChangedMessage>
{
    public Result Handle(ShippingAddressChangedMessage data)
    {
        return Result.Success;
    }
}

// other service
public partial class UpdateAllOrders : IConsumer<ShippingAddressChangedMessage>
{
    public Result Handle(ShippingAddressChangedMessage data)
    {
        var orders = _orderService.GetAllOrders(data.CustomerId);
        foreach (var order in orders)
        {
            UpdateAddress(order.Address);
        }
        return Result.Success;
    }
}
```

PROBLEM

Wpływ na dane, na które nie trzeba

```
public async void UpdateAddress(AddressDto addressDto)
{
    await _messageQueue.Publish(new ShippingAddressChangedMessage(addressDto));
}

// one service
public class UpdateCustomerData : IConsumer<ShippingAddressChangedMessage>
{
    public Result Handle(ShippingAddressChangedMessage data)
    {
        return Result.Success;
    }
}

// other service
public partial class UpdateAllOrders : IConsumer<ShippingAddressChangedMessage>
{
    public Result Handle(ShippingAddressChangedMessage data)
    {
        var orders = _orderService.GetAllOrders(data.CustomerId);
        foreach (var order in orders)
        {
            UpdateAddress(order.Address);
        }
        return Result.Success;
    }
}
```

Niechciane efekty uboczne

```
public ActionResult Login(LoginDto loginDto)
{
    if (!string.IsNullOrEmpty(Request.Cookies["quickLogin"]))
    {
        return _authenticationService.GetTokenFor(Request.Cookies["quickLogin"]);
    }
    return _authenticationService.Login(loginDto);
}
```

PROBLEM

Funkcjonalności deweloperskie

```
public ActionResult Login(LoginDto loginDto)
{
    if (!string.IsNullOrEmpty(Request.Cookies["quickLogin"]))
    {
        return _authenticationService.GetTokenFor(Request.Cookies["quickLogin"]);
    }
    return _authenticationService.Login(loginDto);
}
```

Logowanie bez logowania

```
public class OrderModificationController : Controller
{
    [HttpPost]
    [Route("api/order/{orderId}/addItem")]
    [DisableWhenFeatureSet("order:orderModificetion")]
    public void AddItem()
    {

    }

    [HttpPost]
    [Route("api/order/{orderId}/removeItem")]
    [DisableWhenFeatureSet("order:orderModification")]
    public void RemoveItem()
    {

    }


    [HttpPost]
    [Route("api/payment/{paymentId}/refund")]
    public void RefundPay()
    {

    }

    [HttpPost]
    [Route("api/payment/{paymentId}/supplement")]
    [DisableWhenFeatureSet("order:orderModification")]
    public void SupplementPay()
    {

    }
}
```

```
public void MakeOrder(OrderDto orderDto)
{
    Validate(orderDto);
    _orderRepository.Add(CreateEntity(orderDto));
    if (!_featureFlagManager.IsSet("order:orperModification"))
    {
        _paymentService.PayImmediately(orderDto.Amount);
    } else
    {
        // postpone the payment
        _paymentService.CreatePaymentPlan(orderDto.Amount);
    }
}

public Package FinalizePackage(OrderDto orderDto)
{
    if (!_featureFlagManager.IsSet("order:orderModification"))
    {
        _paymentService.VerifyPayment(orderDto.OrderId);
    }
    else
    {
        _paymentService.PayImmediately(orderDto.Amount);
    }
    return _shippingService.PreparePagckage(orderDto);
}
```

PROBLEM

Błędny zakres feature flag

```
public class OrderModificationController : Controller
{
    [HttpPost]
    [Route("api/order/{orderId}/addItem")]
    [DisableWhenFeatureSet("order:orderModification")]
    public void AddItem()
    {

    }

    [HttpPost]
    [Route("api/order/{orderId}/removeItem")]
    [DisableWhenFeatureSet("order:orderModification")]
    public void RemoveItem()
    {

    }


    [HttpPost]
    [Route("api/payment/{paymentId}/refund")]
    public void RefundPay()
    {

    }

    [HttpPost]
    [Route("api/payment/{paymentId}/supplement")]
    [DisableWhenFeatureSet("order:orderModification")]
    public void SupplementPay()
    {

    }
}
```

```
public void MakeOrder(OrderDto orderDto)
{
    Validate(orderDto);
    _orderRepository.Add(CreateEntity(orderDto));
    if (!_featureFlagManager.IsSet("order:orperModification"))
    {
        _paymentService.PayImmediately(orderDto.Amount);
    } else
    {
        // postpone the payment
        _paymentService.CreatePaymentPlan(orderDto.Amount);
    }
}

public Package FinalizePackage(OrderDto orderDto)
{
    if (!_featureFlagManager.IsSet("order:orderModification"))
    {
        _paymentService.VerifyPayment(orderDto.OrderId);
    }
    else
    {
        _paymentService.PayImmediately(orderDto.Amount);
    }
    return _shippingService.PreparePagckage(orderDto);
}
```

```
public void Add(string productCode, decimal price)
{
    _dbContext.Prices.Add(new PriceEntity(productCode, price));
}

public void Update(string productCode, decimal price)
{
    var oldPrice = _dbContext.Prices.Get(productCode);
    oldPrice.Amount = price;

    _priceAudit.AddEntry(productCode, price);
}

public void Delete(string productCode)
{
    _dbContext.Prices.Remove(productCode);
}
```

PROBLEM

Niepoprawny audyt danych

```
public void Add(string productCode, decimal price)
{
    _dbContext.Prices.Add(new PriceEntity(productCode, price));
}

public void Update(string productCode, decimal price)
{
    var oldPrice = _dbContext.Prices.Get(productCode);
    oldPrice.Amount = price;

    _priceAudit.AddEntry(productCode, price);
}

public void Delete(string productCode)
{
    _dbContext.Prices.Remove(productCode);
}
```

Możliwość skasowania
i dodania rekordu bez logu

```
public enum RoleLevel
{
    Admin,
    CountryAdmin
}
```

```
public void AddUser(UserDto userDto)
{
    if (userDto.CountryId != _currentUserPermission.CountryId)
    {
        throw new NotAllowedException("You are not allowed to do this");
    }

    _dbContext.Users.Add(CreateUserEntity(userDto));
}
```

PROBLEM

Eskalacja uprawnień

```
public enum RoleLevel
{
    Admin,
    CountryAdmin
}
```

```
public void AddUser(UserDto userDto)
{
    if (userDto.CountryId != _currentUserPermission.CountryId)
    {
        throw new NotAllowedException("You are not allowed to do this");
    }

    _dbContext.Users.Add(CreateUserEntity(userDto));
}
```

Możliwość stworzenia Admina przez Coutry Admina

Podpowiedzi

- Pobierz kod i nawiguj się w nim, tak jest łatwiej.
- Przegląd kodu nie musi ograniczać się tylko do kodu (np. anonimizacja danych w osobnym repo, szyfrowanie sekretów w narzędziu do deploymentów).
- Patrz szerzej (np. inne systemy, regulacje).

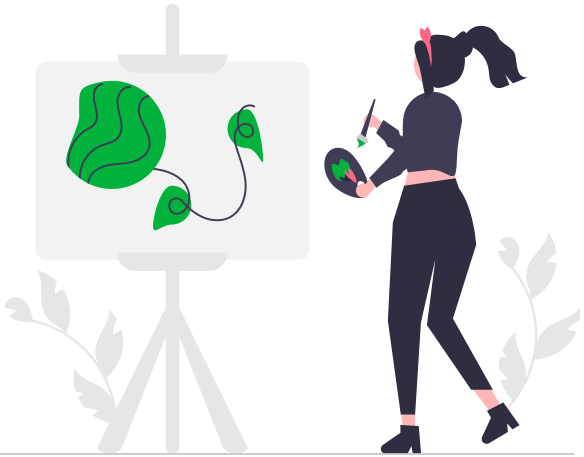
Obszary warte uwagi

- Wrażliwe dane
- Obszary związane z bezpieczeństwem (autentykacja, autoryzacja)
- Obsługa błędów
- Czytelność (security through obscurity)

Wnioski

- Zainwestuj w automatyzację.
- Nie w każdym przypadku trzeba sprawdzać to samo.
- Zainwestuj trochę czasu w zbudowanie swojej checklisty:
  - co warto sprawdzać
  - jak to weryfikować
  - specyfika różnych systemów

Dobre code review to sztuka, a trening czyni mistrza



Zadanie domowe

Weź tę prezentację:
https://www.diwebsity.com/code_review

i zrób ćwiczenia CodeReview
w swoim zespole.

THE H@CK SUMMIT

wcześniej WHAT THE H@CK

Dziękujemy za uwagę!

Zapraszamy do **zadawania pytań**
oraz **oceny wystąpienia**
pod nagraniem.

www.thehacksummit.com | 5/11/2021 | online

ORGANIZATORZY: Academic Partners FUNDACJA | CBC