Przegląd fuzzerów dla języka programowania Dart

Autor: Arkadiusz Charliński s451499

Spis treści

Wstęp

Celem tego raportu jest przegląd narzędzi do fuzzowania dla języka programowania Dart. W szczególności omówione zostaną dwie biblioteki: *fuzz_dart* oraz *Dust*. Oba te narzędzia są wykorzystywane do generowania losowych danych wejściowych, które następnie są testowane pod kątem wywołania wyjątków lub nieoczekiwanych zachowań w aplikacjach.

Fuzzing to technika testowania oprogramowania, polegająca na automatycznym generowaniu i wysyłaniu losowych lub nieprzewidzianych danych wejściowych do aplikacji w celu wykrycia błędów, podatności i problemów z bezpieczeństwem. Głównym celem fuzzingu jest identyfikacja nieoczekiwanych zachowań programu, takich jak awarie, błędy związane z pamięcią czy podatności na ataki, które mogą być wykorzystane przez złośliwych użytkowników.

Fuzzing jest istotnym krokiem w procesie zapewnienia bezpieczeństwa aplikacji, ponieważ pozwala na wczesne wykrycie potencjalnych problemów, zanim program trafi do użytku. Umożliwia to programistom poprawę jakości i bezpieczeństwa oprogramowania, co jest szczególnie ważne w kontekście aplikacji, które przetwarzają wrażliwe dane. Fuzzing jest również wykorzystywany do testowania różnorodnych protokołów sieciowych, formatów plików oraz interfejsów API, co czyni go wszechstronnym narzędziem testującym.

1 Funkcje do testowania

W celu skutecznego testowania fuzzerów stworzono dwie funkcje odpowiedzialne za walidację formularzy rejestracji i logowania.
Plik **login-and-register.dart**

```dart
import 'dart:developer' as dev;
// Add this package to your pubspec.yaml

/// Symulowana funkcja logowania
void login(String email, String password) {
  dev.log('Fuzzed login attempt: Email: $email, Password: $password');

  // Symulowana walidacja logowania
  if (email.isEmpty || password.isEmpty) {
    throw Exception('Email or password cannot be empty');
  }
  if (!RegExp(r'^[^@]+@[^@]+\.[^@]+').hasMatch(email)) {
    throw Exception('Invalid email format');
  }
  if (password.length < 6) {
    throw Exception('Password must be at least 6 characters long');
  }

  dev.log('Login successful for: $email');
}

/// Symulowana funkcja rejestracji
void register(String name, String surname, String email, String phone, String password)
    {
  dev.log('Fuzzed register attempt: Name: $name, Surname: $surname, Email: $email, Phone
    : $phone');

  // Symulowana walidacja rejestracji
  if (name.isEmpty || surname.isEmpty || email.isEmpty || phone.isEmpty || password.
    isEmpty) {
    throw Exception('All fields are required');
  }

  if (name.length < 2 || name.length > 15) {
    throw Exception('Name must be between 2 and 15 characters long');
  }
  if (!RegExp(r'^[a-zA-Z\s]+$').hasMatch(name)) {
    throw Exception('Name cannot contain special characters');
  }

  if (surname.length < 2 || surname.length > 15) {
    throw Exception('Surname must be between 2 and 15 characters long');
  }
```

```
41   if (!RegExp(r'^[a-zA-Z\s]+$').hasMatch(surname)) {
42     throw Exception('Surname cannot contain special characters');
43   }
44
45   if (!RegExp(r'^[^@]+@[^@]+\.[^@]+').hasMatch(email)) {
46     throw Exception('Invalid email format');
47   }
48
49   if (!RegExp(r'^\d{9}$').hasMatch(phone)) {
50     throw Exception('Phone number must be exactly 9 digits long and contain only numbers
       ');
51   }
52
53   if (password.length < 8) {
54     throw Exception('Password must be at least 8 characters long');
55   }
56   if (!RegExp(r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*()_+={}\[\]:;"\'<>,.?~`-]).+
       $').hasMatch(password)) {
57     throw Exception('Password must contain at least one uppercase letter, one lowercase
       letter, one number, and one special character');
58   }
59
60   dev.log('Registration successful for: $name $surname with email: $email');
61 }
```

Listing 1: Funkcje login oraz register

1.1 Opis działania kodu

- **_login(String email, String password)**: Symuluje proces logowania, sprawdzając poprawność adresu e-mail i długości hasła. Jeśli dane są niepoprawne, rzuca wyjątek.
- **_register(String name, String surname, String email, String phone, String password)**: Symuluje proces rejestracji, walidując dane wejściowe. Jeśli dane są nieprawidłowe, rzuca wyjątek.

2 fuzz_dart

Aby przeprowadzić testowanie fuzzerów, skorzystano z pakietu fuzz_dart, który oferuje zaawansowane funkcje do generowania losowych danych wejściowych.

2.1 Omówienie biblioteki

Biblioteka *fuzz_ dart* jest narzędziem do fuzzowania, które umożliwia automatyczne testowanie funkcji w aplikacjach napisanych w języku Dart. Jej głównym celem jest znalezienie błędów poprzez generowanie losowych danych wejściowych i sprawdzanie, jak zachowują się funkcje aplikacji w takich sytuacjach. Biblioteka ta jest w stanie iterować przez wiele kombinacji wartości wejściowych, symulując realistyczne scenariusze, które mogą wystąpić w aplikacji produkcyjnej.

2.2 Wyjaśnienie działania

fuzz_ dart działa poprzez definiowanie typów danych, które mają być testowane, a następnie iteruje przez różne możliwe kombinacje wartości tych danych. W każdym kroku fuzzingu wywoływana jest funkcja testowa, która przetwarza wygenerowane dane. Jeśli podczas przetwarzania danych wejściowych wystąpi wyjątek, fuzzer rejestruje wynik, co pozwala na wykrycie błędów w kodzie, takich jak brak walidacji, nieoczekiwane wyjątki, czy inne problemy logiczne.

Każdy test fuzzer'a można dostosować, ustalając takie parametry jak liczba iteracji (liczba generowanych kombinacji danych wejściowych) oraz typ danych, które mają być testowane, np. stringi, liczby całkowite, booleany. W fuzz_dart możliwe jest także śledzenie pokrycia kodu, co daje programiście informacje na temat tego, które fragmenty kodu są testowane.

2.3 Przykład użycia

W poniższym przykładzie używamy *fuzz_ dart* do testowania funkcji logowania i rejestracji użytkownika, gdzie każda z tych funkcji podlega walidacji danych wejściowych, takich jak format adresu e-mail, poprawność numeru telefonu, siła hasła itp.
Plik **fuzz_dart_test.dart**

```
import 'dart:developer';
import 'package:fuzz_dart/fuzz_dart.dart' as fuzz_dart;

void main() {
  fuzzLoginAndRegister();
}

void fuzzLoginAndRegister() {
  void loginFuzz(String email, String password) {
    try {
      _login(email, password); // Funkcja _login
    } catch (e) {
      log('Exception during login fuzzing: $e');
      rethrow;  // Propagowanie wyj tku w celu zasygnalizowania niepowodzenia testu
    }
  }

  void registerFuzz(String name, String surname, String email, String phone, String
    password) {
    try {
      _register(name, surname, email, phone, password); // Funkcja _register
    } catch (e) {
      log('Exception during register fuzzing: $e');
      rethrow;  // Propagowanie wyj tku w celu zasygnalizowania niepowodzenia testu
    }
  }

  // Utworzenie fuzzer'a dla funkcji logowania
  fuzz_dart.Fuzzer loginFuzzer = fuzz_dart.Fuzzer(
    type: [fuzz_dart.AcceptedTypes.string, fuzz_dart.AcceptedTypes.string],
    iterateCount: 100,
    fileName: 'login-fuzzer-form',
  );

  // Create fuzzer for register function
  fuzz_dart.Fuzzer registerFuzzer = fuzz_dart.Fuzzer(
    type: [
      fuzz_dart.AcceptedTypes.string, // imie
      fuzz_dart.AcceptedTypes.string, // nazwisko
      fuzz_dart.AcceptedTypes.string, // email
      fuzz_dart.AcceptedTypes.string, // nr telefonu
      fuzz_dart.AcceptedTypes.string  // has o
    ],
    iterateCount: 100,
    fileName: 'register-fuzzer-form',
  );

  // Uruchomienie fuzzera
  loginFuzzer.iterate(loginFuzz, 'Login Fuzzing', description: "Simulates login form,
    returns email and password.");
  registerFuzzer.iterate(registerFuzz, 'Register Fuzzing', description: "Simulates
    register form, returns login, password, name, surname, email, phone number");
}
```

Listing 2: Przykładowy kod użycia fuzz_dart

2.4 Opis działania kodu

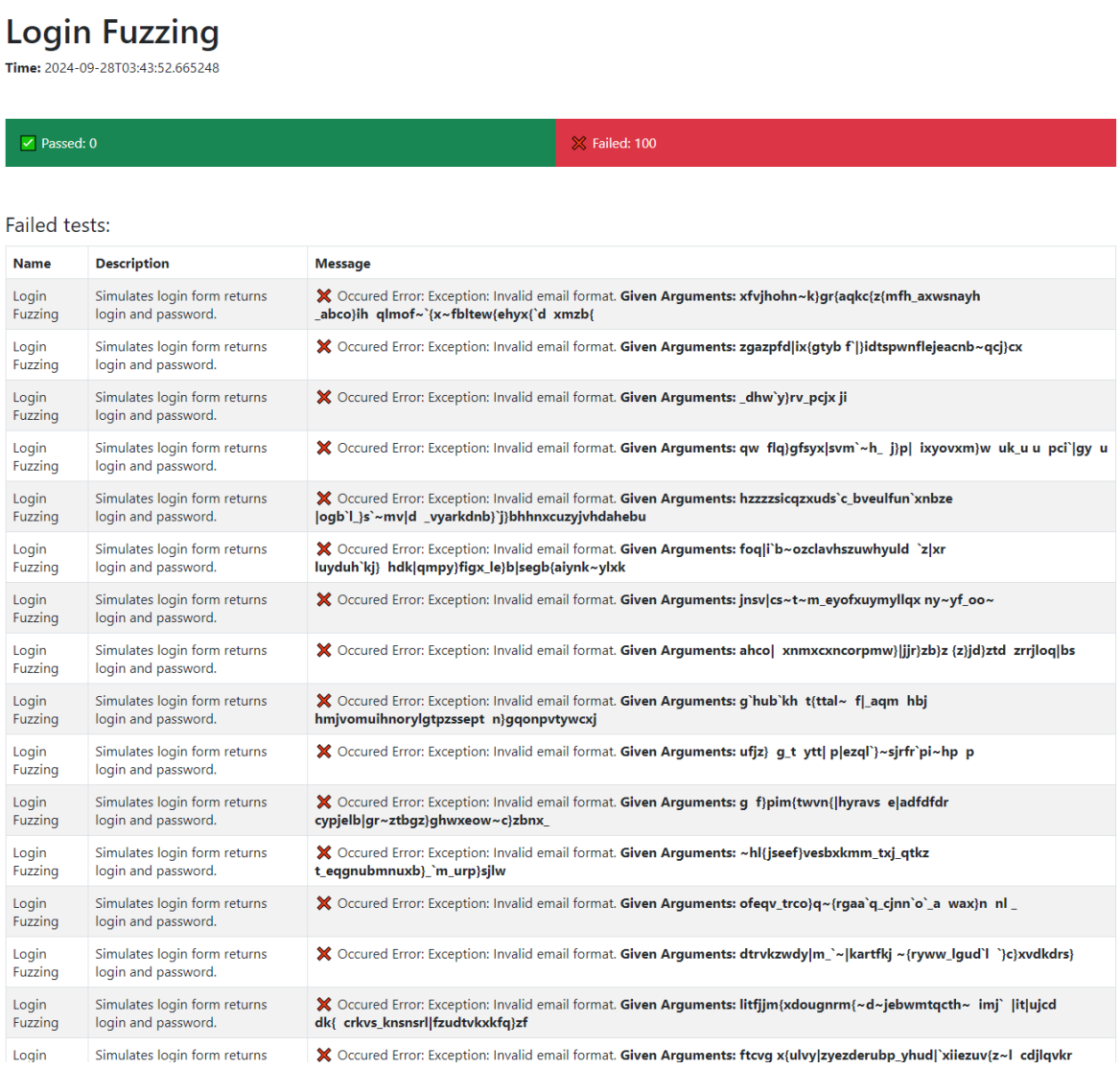
- **main()**: Główna funkcja programu, wywołuje funkcję `fuzzLoginAndRegister()`, która inicjuje testy fuzzingowe.
- **fuzzLoginAndRegister()**: Tworzy dwie funkcje fuzzujące (`loginFuzz()` i `registerFuzz()`), definiuje parametry testowe dla każdej z funkcji i uruchamia fuzzer dla logowania i rejestracji.

- **loginFuzz(String email, String password)**: Symuluje testowanie funkcji logowania za pomocą losowych wartości dla e-maila i hasła. W razie błędu zgłasza wyjątek.
- **registerFuzz(String name, String surname, String email, String phone, String password)**: Symuluje testowanie funkcji rejestracji za pomocą losowych danych osobowych i hasła. W razie błędu zgłasza wyjątek.

2.5 Wyniki

Testy wykonane za pomocą biblioteki *fuzz_dart* pozwoliły na wykrycie kilku kluczowych błędów w funkcjach walidacyjnych aplikacji. Przykładowo, funkcja logowania zgłosiła wyjątek w przypadku nieprawidłowego formatu adresu e-mail oraz za krótkiego hasła. Podobnie funkcja rejestracji poprawnie wykryła niepoprawne dane takie jak brak wymaganych pól, nieprawidłowe numery telefonów, czy też zbyt krótkie hasła.

Fuzzowanie wykazało, że walidacja w aplikacji jest zgodna z założeniami, jednak możliwe jest dalsze rozszerzenie zakresu testów poprzez testowanie dodatkowych przypadków brzegowych. Biblioteka *fuzz_dart* zwraca wyniki w postaci plików HTML, które zawierają szczegółowe informacje na temat wykonanych testów oraz wykrytych błędów. Poniżej są widoczne te właśnie wyniki w skróconej formie.



Login Fuzzing

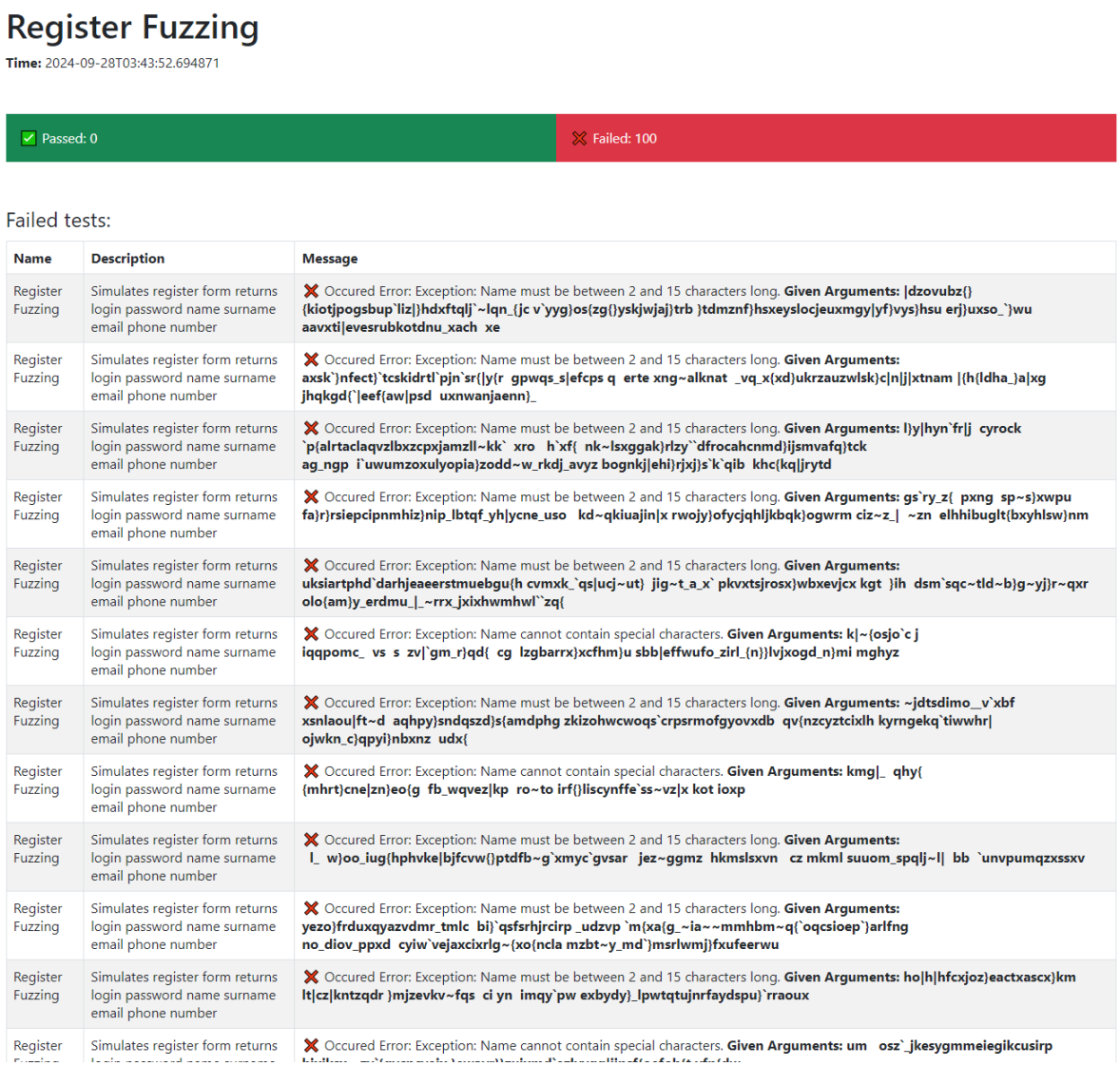
Time: 2024-09-28T03:43:52.665248

Passed: 0 | Failed: 100

Failed tests:

Name	Description	Message
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: xfvjhohn~k}gr{aqkc{z{mfh_axwsnayh _abco}ih qlmof~`{x~fbltew{ehyx{`d xmzb{**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: zgazpfd|ix{gtyb f`|}idtspwnflejeacnb~qcj}cx**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: _dhw`y}rv_pcjx ji**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: qw flq}gfsyx|svm`~h_ j}p| ixyovxm}w uk_u u pci`|gy u**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: hzzzzsicqzxuds`c_bveulfun`xnbze |ogb`l_}s`~mv|d _vyarkdnb}`j}bhhnxcuzyjvhdahebu**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: foq|i`b~ozclavhszuwhyuld `z|xr luyduh`kj} hdk|qmpy}figx_le}b|segb{aiynk~ylxk**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: jnsv|cs~t~m_eyofxuymyllqx ny~yf_oo~**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: ahco| xnmxcxncorpmw}|jjr}zb}z {z}jd}ztd zrrjloq|bs**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: g`hub`kh t{ttal~ f|_aqm hbj hmjvomuihnorylgtpzssept n}gqonpvtywcxj**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: ufjz} g_t ytt| p|ezql`}~sjrfr`pi~hp p**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: g f}pim{twvn{|hyravs e|adfdfdr cypjelb|gr~ztbgz}ghwxeow~c}zbnx_**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: ~hl{jseef}vesbxkmm_txj_qtkz t_eqgnubmnuxb}_`m_urp}sjlw**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: ofeqv_trco}q~{rgaa`q_cjnn`o`_a wax}n nl _**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: dtrvkzwdy|m_`~|kartfkj ~{ryww_lgud`l `}c}xvdkdrs}**
Login Fuzzing	Simulates login form returns login and password.	❌ Occured Error: Exception: Invalid email format. **Given Arguments: litfjjm{xdougnrm{~d~jebwmtqcth~ imj` |it|ujcd dk{ crkvs_knsnsrl|fzudtvkxkfq}zf**
Login	Simulates login form returns	❌ Occured Error: Exception: Invalid email format. **Given Arguments: ftcvg x{ulvy|zyezderubp_yhud|`xiiezuv{z~l cdjlqvkr**

Rysunek 1: Wynik testu logowania



Register Fuzzing

Time: 2024-09-28T03:43:52.694871

Passed: 0 | Failed: 100

Failed tests:

Name	Description	Message
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: |dzovubz{} {kiotjpogsbup`liz|}hdxftqlj`~lqn_{jc v`yyg}os{zg{}yskjwjaj}trb }tdmznf}hsxeyslocjeuxmgy|yf}vys}hsu erj}uxso_`}wu aavxti|evesrubkotdnu_xach xe**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: axsk`}nfect}`tcskidrtl`pjn`sr{|y{r gpwqs_s|efcps q erte xng~alknat _vq_x{xd}ukrzauzwlsk}c|n|j|xtnam |{h{ldha_}a|xg jhqkgd{`|eef{aw|psd uxnwanjaenn}_**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: l}y|hyn`fr|j cyrock `p{alrtaclaqvzlbxzcpxjamzll~kk` xro h`xf{ nk~lsxggak}rlzy``dfrocahcnmd}ijsmvafq}tck ag_ngp i`uwumzoxulyopia}zodd~w_rkdj_avyz bognkj|ehi}rjxj}s`k`qib khc{kq|jrytd**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: gs`ry_z{ pxng sp~s}xwpu fa}r}rsiepcipnmhiz}nip_lbtqf_yh|ycne_uso kd~qkiuajin|x rwojy}ofycjqhljkbqk}ogwrm ciz~z_| ~zn elhhibuglt{bxyhlsw}nm**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: uksiartphd`darhjeaeerstmuebgu{h cvmxk_`qs|ucj~ut} jig~t_a_x` pkvxtsjrosx}wbxevjcx kgt }ih dsm`sqc~tld~b}g~yj}r~qxr olo{am}y_erdmu_|_~rrx_jxixhwmhwl``zq{**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name cannot contain special characters. **Given Arguments: k|~{osjo`c j iqqpomc_ vs s zv|`gm_r}qd{ cg lzgbarrx}xcfhm}u sbb|effwufo_zirl_{n}}lvjxogd_n}mi mghyz**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: ~jdtsdimo__v`xbf xsnlaou|ft~d aqhpy}sndqszd}s{amdphg zkizohwcwoqs`crpsrmofgyovxdb qv{nzcyztcixlh kyrngekq`tiwwhr| ojwkn_c}qpyi}nbxnz udx{**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name cannot contain special characters. **Given Arguments: kmg|_ qhy{ {mhrt}cne|zn}eo{g fb_wqvez|kp ro~to irf{}liscynffe`ss~vz|x kot ioxp**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: l_ w}oo_iug{hphvke|bjfcvw{}ptdfb~g`xmyc`gvsar jez~ggmz hkmslsxvn cz mkml suuom_spqlj~l| bb `unvpumqzxssxv**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: yezo}frduxqyazvdmr_tmlc bi}`qsfsrhjrcirp _udzvp `m{xa{g_~ia~~mmhbm~q{`oqcsioep`}arlfng no_diov_ppxd cyiw`vejaxcixrlg~{xo{ncla mzbt~y_md`}msrlwmj}fxufeerwu**
Register Fuzzing	Simulates register form returns login password name surname email phone number	Occured Error: Exception: Name must be between 2 and 15 characters long. **Given Arguments: ho|h|hfcxjoz}eactxascx}km lt|cz|kntzqdr }mjzevkv~fqs ci yn imqy`pw exbydy}_lpwtqtujnrfaydspu}`rraoux**
Register	Simulates register form returns	Occured Error: Exception: Name cannot contain special characters. **Given Arguments: um osz`_jkesygmmeiegikcusirp**

Rysunek 2: Wynik testu rejestracji

3 Dust

Biblioteka *Dust* jest narzędziem do fuzzowania dla języka Dart, które wykorzystuje techniki fuzz testowania prowadzonego przez pokrycie kodu. Inspiracją dla tej biblioteki były popularne narzędzia, takie jak libFuzzer i AFL. Działa ona poprzez losowe generowanie danych wejściowych i ich przekazywanie do programu w celu wykrywania błędów oraz nieoczekiwanych zachowań.

Aby skorzystać z biblioteki, wystarczy napisać program w Dart, który przyjmuje argumenty w funkcji main:

```
void main(List<String> args) {
  final input = args[0];
  /* use input */
}
```

Następnie można uruchomić test fuzzowania, używając polecenia:

```
pub global activate dust
pub global run dust path/to/script.dart
```

Zaleca się utworzenie snapshotu skryptu przed uruchomieniem dla lepszej wydajności. Podczas testowania, *Dust* generuje dane wejściowe, sprawdza, czy występują błędy, i rejestruje informacje o nieudanych próbach. Domyślnie biblioteka tworzy katalog z danymi testowymi, które służą do dalszego eksplorowania kodu, co jest kluczowe w procesie fuzzowania.

Dzięki możliwości ręcznego określenia ziarna oraz opcji minimalizacji i łączenia zbiorów, *Dust* pozwala na elastyczne dostosowanie procesu testowania.

Niestety, nie udało mi się uruchomić biblioteki na moim komputerze. Poświęciłem na to kilka godzin i pobrałem kod z repozytorium GitHub (`https://github.com/MichaelRFairhurst/dust`), które nie było aktualizowane od pięciu lat. Mimo obniżenia wersji Dart do *Dart SDK version: 2.19.0-146.2.beta* (beta), nie mogłem uruchomić kodu *Dust*. Próbowałem poprawić i debugować kod, jednak na jeden rozwiązany błąd pojawiały się dwa kolejne, co uniemożliwiło mi przeprowadzenie skutecznego testu z użyciem tej biblioteki. Co gorsza, *Dust* nie jest kompatybilna z wersjami Dart wyższymi niż 3, co czyni ją obecnie całkowicie bezużyteczną.

4 Własna implementacja

Z powodu niedziałającej biblioteki *Dust* zdecydowałem się dodać własną implementację fuzzowania. Moja implementacja wykorzystuje popularne techniki generowania losowych danych wejściowych do testowania funkcji rejestracji i logowania w aplikacji.

Do testowania wykorzystywane są funkcje *register* oraz *login*, podobnie jak w projekcie *fuzz_ dart*. Oto kod mojej implementacji:
Plik **custom-fuzzing.dart**

```
import 'dart:io';
import 'package:xml/xml.dart';
import 'package:faker/faker.dart';
import 'login-and-register.dart';

const int NUM_LOGIN_TESTS = 100;
const int NUM_REGISTRATION_TESTS = 100;

List<XmlElement> fuzzTestLogin(int numTests) {
  final faker = Faker(); // Utw rz instancj  Faker
  List<XmlElement> loginResults = [];

  for (int i = 0; i < numTests; i++) {
    String email = faker.internet.email(); // Wygeneruj losowy email
    String password = faker.internet.password(); // Wygeneruj losowe has o

    print('Test $i: Attempting login with Email: $email, Password: $password');
    try {
      login(email, password);
      print('Login successful');
```

```
      loginResults.add(XmlElement(XmlName('loginResult'), [], [
        XmlElement(XmlName('test'), [], [
          XmlElement(XmlName('email'), [], [XmlText(email)]),
          XmlElement(XmlName('password'), [], [XmlText(password)]),
          XmlElement(XmlName('status'), [], [XmlText('success')])
        ])
      ]));
    } catch (e) {
      print('Login failed: $e');
      loginResults.add(XmlElement(XmlName('loginResult'), [], [
        XmlElement(XmlName('test'), [], [
          XmlElement(XmlName('email'), [], [XmlText(email)]),
          XmlElement(XmlName('password'), [], [XmlText(password)]),
          XmlElement(XmlName('status'), [], [XmlText('failed: $e')])
        ])
      ]));
    }
  }

  return loginResults;
}

List<XmlElement> fuzzTestRegister(int numTests) {
  final faker = Faker(); // Utw rz instancj  Faker
  List<XmlElement> registrationResults = [];

  for (int i = 0; i < numTests; i++) {
    String name = faker.person.firstName();
    String surname = faker.person.lastName();
    String email = faker.internet.email();
    String phone =
        faker.randomGenerator.integer(100000000, min: 100000000).toString();
    String password = faker.internet.password();

    print(
        'Test $i: Attempting register with Name: $name, Surname: $surname, Email: $email
    , Phone: $phone, Password: $password');
    try {
      register(name, surname, email, phone, password);
      print('Registration successful');
      registrationResults.add(XmlElement(XmlName('registrationResult'), [], [
        XmlElement(XmlName('test'), [], [
          XmlElement(XmlName('name'), [], [XmlText(name)]),
          XmlElement(XmlName('surname'), [], [XmlText(surname)]),
          XmlElement(XmlName('email'), [], [XmlText(email)]),
          XmlElement(XmlName('phone'), [], [XmlText(phone)]),
          XmlElement(XmlName('password'), [], [XmlText(password)]),
          XmlElement(XmlName('status'), [], [XmlText('success')])
        ])
      ]));
    } catch (e) {
      print('Registration failed: $e');
      registrationResults.add(XmlElement(XmlName('registrationResult'), [], [
        XmlElement(XmlName('test'), [], [
          XmlElement(XmlName('name'), [], [XmlText(name)]),
          XmlElement(XmlName('surname'), [], [XmlText(surname)]),
          XmlElement(XmlName('email'), [], [XmlText(email)]),
          XmlElement(XmlName('phone'), [], [XmlText(phone)]),
          XmlElement(XmlName('password'), [], [XmlText(password)]),
          XmlElement(XmlName('status'), [], [XmlText('failed: $e')])
        ])
      ]));
    }
  }

  return registrationResults;
}

void main() {
  List<XmlElement> loginResults = fuzzTestLogin(NUM_LOGIN_TESTS);
  List<XmlElement> registrationResults =
      fuzzTestRegister(NUM_REGISTRATION_TESTS);

```

```
93    // Tworzenie dokumentu XML
94    final builder = XmlBuilder();
95    builder.processing('xml', 'version="1.0"');
96    builder.element('fuzzTestingResults', nest: () {
97      builder.element('loginTests', nest: loginResults);
98      builder.element('registrationTests', nest: registrationResults);
99    });
100
101   final document = builder.buildDocument();
102   final xmlString = document.toXmlString(pretty: true, indent: '  ');
103   // Zapis do pliku xml
104   File('fuzz_testing_results.xml').writeAsStringSync(xmlString);
105   print('Results have been written to fuzz_testing_results.xml');
106 }
```

Listing 3: Własna implementacja

4.1 Opis działania kodu

Kod składa się z dwóch głównych funkcji do fuzzowania: `fuzzTestLogin` i `fuzzTestRegister`, a także funkcji `main`.

- **fuzzTestLogin(int numTests)**:
  - Parametr: `numTests` - liczba testów do przeprowadzenia.
  - Działanie: Funkcja generuje losowe dane wejściowe (adres e-mail i hasło) przy użyciu biblioteki Faker. Następnie wywołuje funkcję `login` z tymi danymi. Jeśli logowanie zakończy się sukcesem, dodaje wynik do listy wyników jako element XML. W przeciwnym razie rejestruje błąd.
- **fuzzTestRegister(int numTests)**:
  - Parametr: `numTests` - liczba testów do przeprowadzenia.
  - Działanie: Funkcja generuje losowe dane wejściowe (imię, nazwisko, adres e-mail, telefon i hasło) przy użyciu biblioteki Faker. Następnie wywołuje funkcję `register` z tymi danymi. Jeśli rejestracja zakończy się sukcesem, dodaje wynik do listy wyników jako element XML. W przeciwnym razie rejestruje błąd.
- **main()**:
  - Działanie: Funkcja główna wywołuje `fuzzTestLogin` i `fuzzTestRegister` w celu przeprowadzenia testów. Następnie tworzy dokument XML z wynikami testów i zapisuje go do pliku `fuzz_testing_results.xml`.

4.2 Opis biblioteki Faker

Biblioteka *Faker* jest używana do generowania fikcyjnych danych, które mogą być wykorzystywane w testach oraz do symulacji danych użytkowników w aplikacjach. Umożliwia ona łatwe tworzenie losowych wartości, takich jak imiona, nazwiska, adresy e-mail, numery telefonów, hasła oraz inne dane.

4.3 Wyniki

Poniżej znajduje się fragment outputu z kodu:

```
Test 0: Attempting login with Email: sydni_marquardt@kilback.name, Password: 2d(Z7:/8!R
Login successful
Test 1: Attempting login with Email: stefan-kutch@johns.name, Password: le:lE\7'yi
Login successful
Test 2: Attempting login with Email: torphy_lea@lesch.co.uk, Password: 6{RB.K"h'Z
Login successful
Test 3: Attempting login with Email: linnie-terry@oga.name, Password: v&SG‘85zj:
Login successful
Test 4: Attempting login with Email: hauck_federico@reilly.co.uk, Password: wKz3Am#$8Z
```

```
Login successful
Test 5: Attempting login with Email: augustine_schiller@ledner.co.uk, Password: ‘h(4y;/AFg
Login successful
...
Test 96: Attempting register with Name: Bradly, Surname: Cormier, Email: edyth-mccullough@oconnell.us
Registration successful
Test 97: Attempting register with Name: Amari, Surname: Schmitt, Email: ernser-eryn@kuhic.info, Phone
Registration failed: Exception: Password must contain at least one uppercase letter, one lowercase le
Test 98: Attempting register with Name: Randall, Surname: Veum, Email: cruz_tillman@swaniawski.us, Ph
Registration successful
Test 99: Attempting register with Name: Natalia, Surname: Schoen, Email: eunice.hyatt@conroy.name, Ph
Registration successful
Results have been written to fuzz_testing_results.xml
```

Fragment wyników zapisanych do pliku XML:

```
<fuzzTestingResults>
  <loginTests>
    <loginResult>
      <test>
        <email>sydni_marquardt@kilback.name</email>
        <password>2d(Z7:/8!R</password>
        <status>success</status>
      </test>
      <test>
        <email>stefan-kutch@johns.name</email>
        <password>le:lE\7’yi</password>
        <status>success</status>
      </test>
      <!-- więcej testów -->
    </loginResult>
  </loginTests>
  <registrationTests>
    <registrationResult>
      <test>
        <name>Bradly</name>
        <surname>Cormier</surname>
        <email>edyth-mccullough@oconnell.us</email>
        <phone>100000000</phone>
        <password>3wh8mlN4>U</password>
        <status>success</status>
      </test>
      <test>
        <name>Amari</name>
        <surname>Schmitt</surname>
        <email>ernser-eryn@kuhic.info</email>
        <phone>100000000</phone>
        <password>=nRXF?d+E></password>
        <status>failed: Exception: Password must contain at least one uppercase letter, one lowercase
      </test>
      <!-- więcej testów -->
    </registrationResult>
  </registrationTests>
</fuzzTestingResults>
```

4.4 Podsumowanie

Własna implementacja fuzzowania okazała się skuteczna w identyfikowaniu potencjalnych problemów w funkcjach rejestracji i logowania. Dzięki wykorzystaniu biblioteki Faker udało się wygenerować różno-

rodne dane wejściowe, co zwiększyło zasięg testów. Implementacja ta, mimo iż jest podstawowa, dostarcza cennych informacji na temat funkcjonalności aplikacji i wskazuje obszary, które mogą wymagać dalszej uwagi i poprawek. Ponadto, zapis wyników w formacie XML ułatwia analizę wyników testów i integrację z innymi narzędziami.

5 Wnioski

W przeprowadzonym przeglądzie narzędzi fuzzowania dla języka Dart zwrócono szczególną uwagę na dwie biblioteki: *fuzz_ dart* oraz *Dust*. Obie biblioteki oferują unikalne podejścia do automatyzacji testów i pomagają w wykrywaniu błędów.

Biblioteka *fuzz_ dart* okazała się być funkcjonalnym narzędziem do fuzzowania, które umożliwiło wykrycie błędów w implementacji funkcji logowania i rejestracji.

Z drugiej strony, trudności z uruchomieniem *Dust* podkreślają wyzwania związane z utrzymywaniem i aktualizowaniem narzędzi fuzzujących. Niewystarczająca dokumentacja oraz brak wsparcia dla nowszych wersji Dart mogą ograniczać ich zastosowanie w praktyce. Pomimo tych trudności, opracowana własna implementacja fuzzowania z wykorzystaniem biblioteki Faker i XML do generowania wyników testów również spełniła swoje zadanie.