UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: Informatyka

Piotr Jasina
nr albumu: 279183

Identyfikacja inteligentnych kontraktów w sieci Ethereum

Ethereum smart contracts identification

Praca licencjacka
napisana w Zakładzie Cyberbezpieczeństwa
pod kierunkiem dr. Damiana Rusinka

Lublin rok 2019

Spis treści

Wstęp

...

Rozdział 1

Ethereum

1.1 Historia

Literatura: [2, ?]. TODO

1.2 Opis platformy

Literatura: [2, ?]. TODO

1.3 Ethereum Virtual Machine

Literatura: [2, ?]. TODO

1.4 Inteligentne kontrakty

Literatura: [2, ?]. TODO

Rozdział 2

Solidity

2.1 Sygnatura funkcji

Literatura: [2, ?]. TODO

2.2 Selektor funkcji

Literatura: [2, ?]. TODO

2.3 Generowanie akcesorów podczas kompilacji

Literatura: [2, ?]. TODO

Rozdział 3

Projekt Aplikacji

Celem mojej pracy licencjackiej było stworzenie umożliwiającej identyfikację inteligentnych kontraktów wykorzystywanych w sieci Ethereum. Dzięki aplikacji użytkownik po wprowadzeniu na stronie kodu bajtowego kontraktu jest w stanie otrzymać najbardziej prawdopodobną implementacje kontraktu napisana w języku Solidity bazując na bazie danych aplikacji.

Aplikacja realizująca cel pracy została stworzona przy wykorzystaniu frameworka Spring Boot, jest to aplikacja internetowa, dzięki czemu można ja wykorzystać wielu urządzeniach bez konieczności instalacji. W celu przechowywania danych wykorzystano nierelacyjną bazę danych MongoDB.

Poniżej opisałem działanie aplikacji wraz ze szczegółowym opisem funkcjonalności, architektury oraz wykorzystanych technologi.

3.1 Opis funkcjonalności

Na stronie głównej znajduje się opis aplikacji wraz z aktualna liczba kodów źródłowych znajdujących się w bazie danych oraz podstawowe definicje związane z aplikacją. Cała aplikacji udostępnia trzy główne funkcjonalności: identyfikację

inteligentnych kontraktu, wprowadzanie plików źródłowych kontraktów do aplikacji oraz interfejs programistyczny aplikacji. Wszystkie funkcjonalności zostały opisane poniżej

3.1.1 Identyfikacja inteligentnych kontraktów

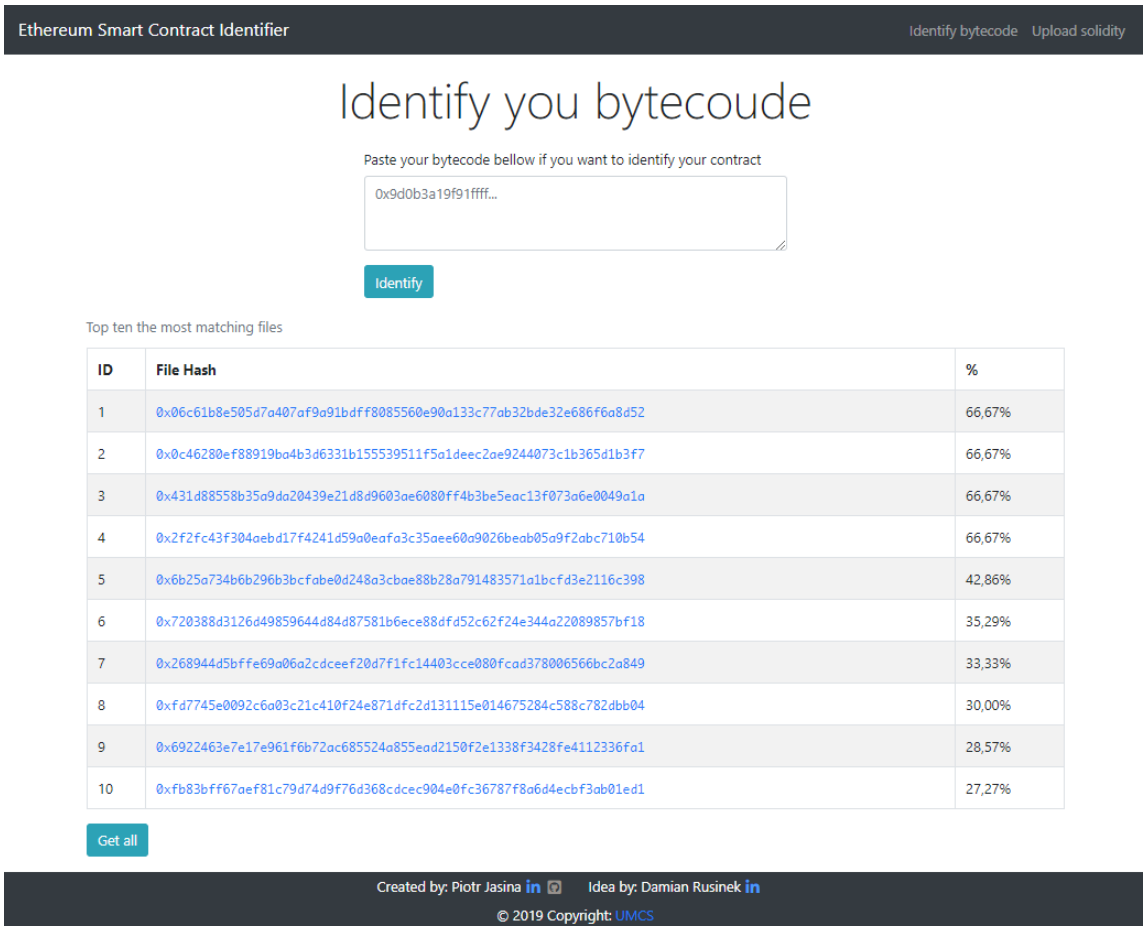
Pierwsza opcją dostępną w aplikacji jest identyfikacja inteligentnych kontraktów. Identyfikacje kontraktu można rozpocząć będąc na stronie główniej lub na podstronie dedykowanej specjalnie identyfikacji kontraktów. Zarówno na stronie głównej, jak i na podstronie znajduje się pole w którym można wprowadzić kod bajtowy. Po wprowadzeniu danych użytkownik zatwierdza je w obu przypadkach klikając przycisk **Identify**. Natomiast, jeśli użytkownik chce udać się na podstronę, należy wybrać przycisk w menu o nazwie **Identify bytecode**, następnie użytkownik zostanie przekierowany na podstronę dedykowaną identyfikacji kontraktów. W przypadku wprowadzenia kodu bajtowego na stronie głównej zostaniemy również przekierowani na podstronę z taka różnica, ze pojawią się od razu wyniki identyfikacji kontraktu. Zarówno na stronie głównej, jak i podstronie dedykowanej identyfikacji, użytkownik jest zobowiązany wprowadzać kod bajtowy w szesnastkowym systemie liczbowym, w innym wypadku identyfikacja nie przejdzie prawidłowo.

Podczas wprowadzania kodu istnieje możliwość wprowadzenia kodu z prefiksem **0x** lub bez tego prefixu. Jeśli użytkownik poda kod z prefiksem to aplikacja podczas przetwarzania tego kodu zignoruje ten prefiks. Takie rozwiązanie zostało zastosowane w celu zapewnienia użytkownikowi większej wygody oraz komfortu w korzystaniu z aplikacji. Użytkownik nie będzie musiał zastanawiać się czy kopiując kod bajtowy z dowolnego źródła, jest on z prefiksem czy nie, ponieważ obie opcje są wspierane.

Po wprowadzeniu danych i zatwierdzeniu ich przyciskiem **Identify**, aplikacja

rozpoczyna proces analizy wprowadzonego kodu bajtowego oraz wyszukiwane są najbardziej prawdopodobne implementacje posortowane malejąco według współczynnika dopasowania. W rezultacie jak możemy zobaczyć na rysunku 3.1 użytkownik otrzymuje listę dziesięciu najbardziej prawdopodobnych implementacji.

Mimo że domyślnie jest wyświetlanych tylko dziesięć najbardziej prawdopodobnych implementacji, istnieje też możliwość pobrania wszystkich wyników identyfikacji kontraktu używając przycisku **Get all**. Jak widać na rysunku 3.1, przycisk **Get all** znajduje się pod pierwszą dziesiątka wyników. Po naciśnięciu przycisku strona zostanie załadowana ponownie wraz z pełną listą haszy plików



Rysunek 3.1: Wynik identyfikacji inteligentnego kontraktu

i ich współczynnikami dopasowania.

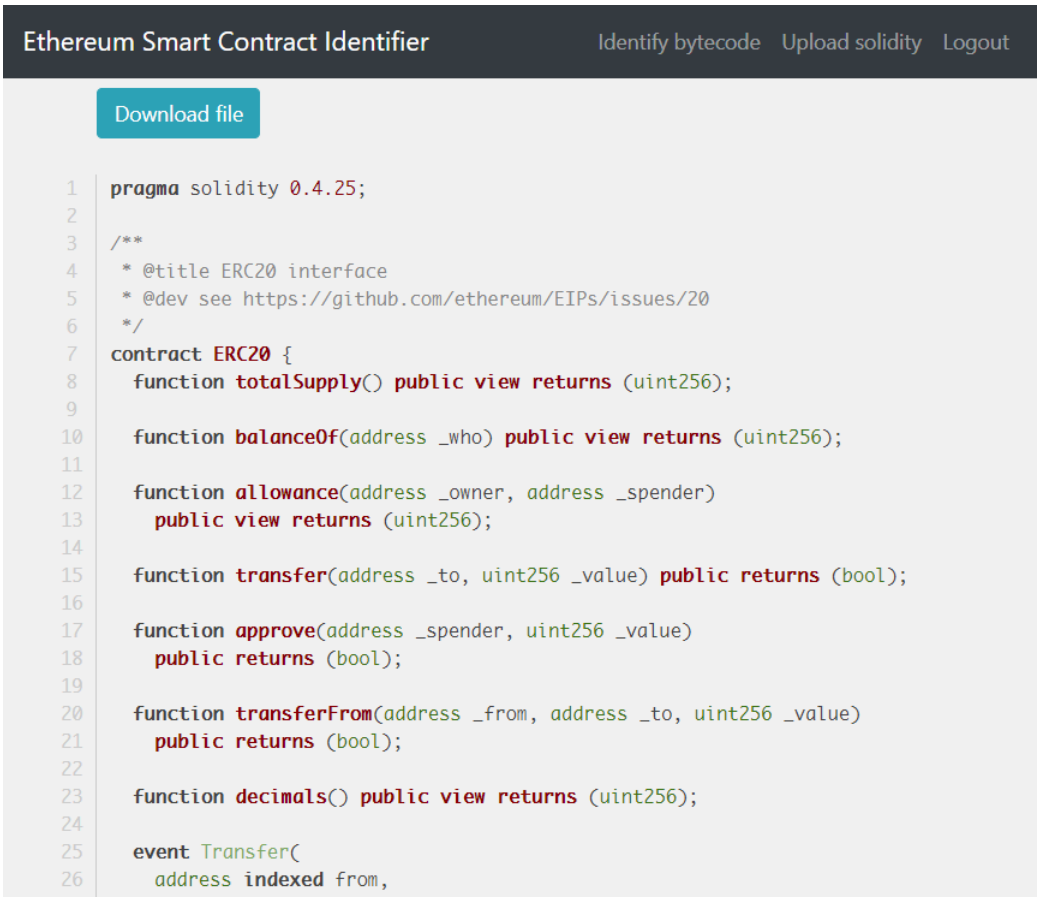
Po naciśnięciu w jedną z wyświetlanych implementacji, użytkownikowi pojawi się w nowej karcie przeglądarki podstrona umożliwiająca podgląd implementacji.

Jak widać na rysunku 3.2 kod źródłowy kontraktu jest wyświetlany ze specjalnie przygotowanym dla języka Solidity podświetleniem składni przygotowanym, natomiast po lewej stronie można zobaczyć przygotowaną numeracje wierszy, która ma za zadanie ułatwić nawigację po kodzie źródłowym na stronie internetowej. Rozwiązanie z numerowaniem linii zostało zaimplementowane w taki sposób, aby podczas kopiowania kodu źródłowego ze strony, nie były kopiowane z nim liczby identyfikujące konkretną linie w kodzie.

3.1.2 Dodanie kodu źródłowego kontraktu do aplikacji

Kolejna funkcjonalnością dostępną dla użytkownika jest możliwość dodania własnego kodu źródłowego kontraktu napisanego w języku Solidity. Opcja ta umożliwia użytkownikowi uzupełnienie aktualnej bazy danych o kolejne kody źródłowe inteligentnych kontraktów. W rezultacie zgromadzenia dużej ilości implementacji w bazie danych, wszyscy pozostali użytkownicy maja większą szansę na precyzyjną identyfikacje kontraktu. W celu wykorzystania tej funkcjonalności użytkownik musi zalogować się za pomocą panelu logowania. Zakładając, że osoba korzystająca z tej części aplikacji posiada już odpowiednie uprawnienia to po naciśnięciu przycisku **Upload solidity**, zostanie przekierowana na podstronę, na której ma możliwość wprowadzenia kodu źródłowego. Zostały utworzone dwie możliwości wprowadzania kodów źródłowych, które opiszę poniżej.

Pierwszą sposobem jest przesłanie do aplikacji pliku zawierającego implementacje kontraktu napisana w języku Solidity. W tym przypadku użytkownik powinien kliknąć przycisk **Browse**, który umożliwi mu wybranie za pomocą przeglądarki internetowej konkretnego pliku znajdującego się na dysku lokalnym, a



Rysunek 3.2: Podgląd implementacji

następnie zatwierdzić go przyciskiem **Upload** znajdującym się obok wcześniej wspomnianego przycisku.

Innym sposobem na przesłanie kodu źródłowego do aplikacji jest wklejenie kodu źródłowego bezpośrednio do formularza znajdującego się po prawej części strony internetowej. Ta opcja została utworzona w celu zapewnienia użytkownikowi większej elastyczności i komfortu w korzystaniu z aplikacji. Przykładowo podczas korzystania z aplikacji, użytkownik może bezpośrednio skopiować kod źródłowy, który jest w dowolnym innym źródle tekstowym np. innej stronie internetowej i wkleić go bezpośrednio do aplikacji bez konieczności tworzenia pliku tymczasowego.

Po prawidłowym dodaniu kodu źródłowego do aplikacji, użytkownik powinien zobaczyć podobny rezultat do tego na rysunku 3.3. W momencie dodania nowej implementacji, na stronie pojawia się hasz dodanego pliku oraz lista sygnatur funkcji wraz z ich selektorami.

Użytkownik ma możliwość przeglądania kodu źródłowego, który wysłał na serwer. W tym celu należy nacisnąć na hasz pliku wyświetlany poniżej formularza dodawania, następnie użytkownik zostanie przeniesiony na stronę na której może zobaczyć dodany przez siebie kod źródłowy z numeracją linii oraz podświetleniem składni.

Na dolnej części rysunku 3.3 znajduje się tabela z funkcjami wyszukanymi w implementacji. W kolumnie **Signature** znajdują się wszystkie sygnatury funkcji, natomiast w kolumnie **Selector** odpowiadajace im selektory funkcji.

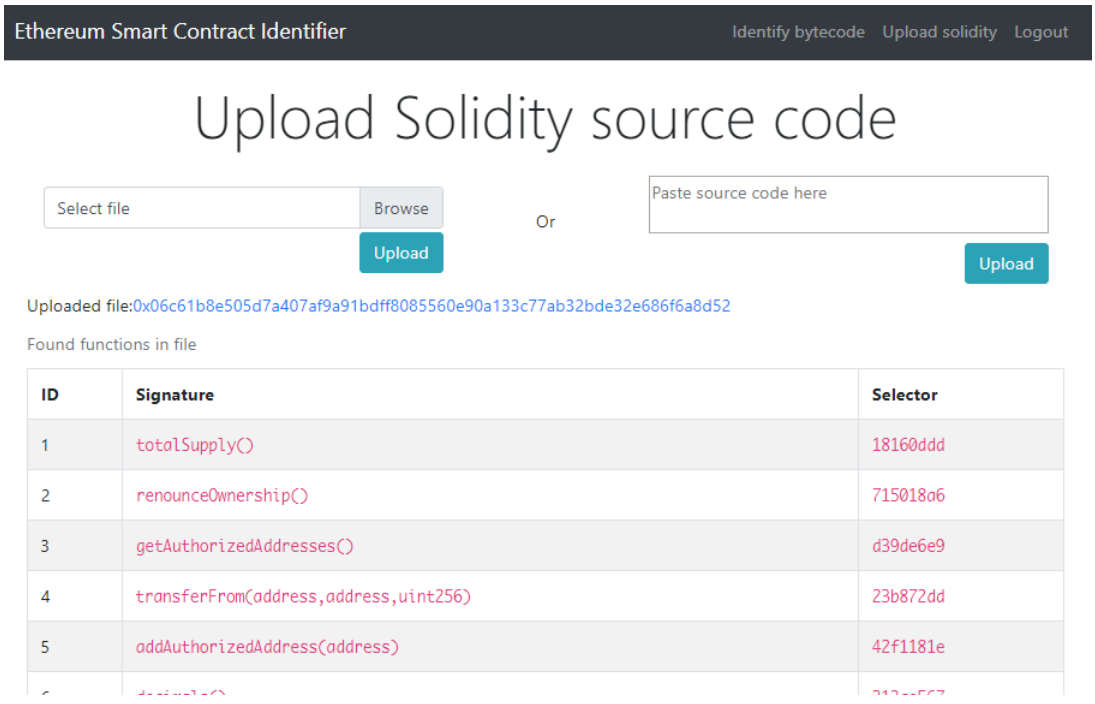
3.1.3 Interfejs programistyczny aplikacji

Trzecia funkcjonalnością aplikacji jest interfejs programistyczny. Umożliwia on tworzenie własnego oprogramowania oraz skryptów, bazując na utworzonej prze zemnie aplikacji, przez innych programistów. Dzięki temu można wykorzy-

stać mechanizmy zaimplementowane w aplikacji w celu rozszerzenia ich w innej aplikacji lub w celu zautomatyzowania niektórych procesów bez wykorzystania GUI (ang. graphical user interface) aplikacji.

W celu skorzystania z interfejsu programistycznego należy utworzyć żądanie HTTP. Za pomocą żądania HTTP użytkownik aplikacji ma możliwość dostarczenia na serwer nowego kodu źródłowego, pobrania kodu źródłowego na podstawie hasza kodu źródłowego znajdującego się na serwerze oraz identyfikacje kontraktu.

Przykładowym zastosowaniem API (ang. Application programming interface) może być utworzenie skryptu umożliwiającego zautomatyzowanie wysyłania kodów źródłowych do aplikacji, bez konieczności korzystania z interfejsu graficznego aplikacji. W tym przypadku użytkownik, który chce przesłać nowy plik do aplikacji musi najpierw przejść proces autoryzacji, natomiast pozostałe funkcje API nie wymagają uwierzytelniania użytkownika przez aplikacje.



Rysunek 3.3: Rezultat przesłania inteligentnego kontraktu do aplikacji

Pobieranie informacji o kodzie źródłowym z API

Jeśli użytkownik chce pobrać za pomocą API informacje o konkretnym kodzie źródłowym kontraktu, w tym celu powinien utworzyć żądanie HTTP. W przypadku tej funkcjonalności nie jest wymagane uwierzytelnienie użytkownika.

Na listingu 3.1 można zaobserwować przykładowe żądanie HTTP, które jest wykorzystane w celu pobrania informacji o konkretnym kodzie źródłowym. Użytkownik podczas korzystania z API powinien zwrócić szczególna uwagę na wykorzystywaną metodę w nagłówku HTTP, ponieważ interfejs aplikacji wspiera tylko wybrane metody. Na listingu 3.1 widać, że używaną metodą jest **GET**, która jest jedną z kilku dostępnych metod w standardzie *HTTP/1.1*, istnieją też między innymi metody: *POST*, *HEAD*, *PUT* czy *DELETE*. Metody HTTP zostały opisane szczegółowo w sekcji 9.2 dokumentu RFC2616 [1].

Zaraz za metoda GET, można zobaczyć listingu 3.1, znajduje się adres pod którym zostanie wysyłane żądanie. Na końcu adresu po znaku zapytania podane zostały atrybuty o nazwie *fileHash* ze wskazaną wartością hasza pliku. Odpowiedzią na tak przygotowane zapytanie będzie w tym przypadku zwykły tekst implementację kontraktu oraz status HTTP 200, mówiący użytkownikowi, że wszystko poszło dobrze. W sytuacji gdy, użytkownik otrzyma status HTTP 404, oznacza to, że nie udało się znaleźć implementacji o podanym haszu.

Listing 3.1: Przykladowe żądanie HTTP

```
1 GET /api/solidityFiles/sourceCode?fileHash=0
    x06c61b8e505d7a407af9a91bdff8085560e90a133c77ab32bde32e686f6a8d52
     HTTP/1.1
2 Host: contractmy.herokuapp.com
3 Connection: keep-alive
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
```

```
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
    /537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
    image/webp,image/apng,*/*;q=0.8
8 Accept-Encoding: gzip, deflate, br
9 Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
```

Identyfikacja inteligentnego kontraktu za pomocą API

Podczas identyfikacji inteligentnego kontraktu, za pomocą interfejsu programistycznego, żądanie pokazane na listingu 3.1 musimy nieco zmodyfikować. Podczas identyfikacji wykorzystywana jest metoda POST. W nagłówku HTTP należy więc wprowadzić tą metodę zamiast GET, a następie zmienić ścieżkę na */api/bytecode*. Pod podana ścieżką zostanie wysłane żądanie HTTP. W przypadku metody POST opisanej w sekcji 9.5 dokumentu RFC2616 [1] podajemy poniżej nagłówka zawartość, która będzie przesłana do serwera. W odróżnieniu od wykorzystanej poprzednio metody GET, tutaj atrybuty powinny zostać podane poniżej nagłówka, zamiast w adresie.

API pod adresem /api/bytecode wymaga od użytkownika dwóch atrybutów o nazwach: bytecode oraz allFiles. Poniżej nagłówka HTTP w żądaniu należy wprowadzić odpowiednio zakodowane atrybuty. W celu zakodowania listy atrybutów, każdy atrybut zapisuje się w postaci:

```
NAZWA_ATRYBUTU=WARTOSC_ATRYBUTU
```

łącząc je wszystkie ampersandem. Poprawnie zakodowana zawartość w żądaniu HTTP w tym przypadku powinna wyglądać następująco:

```
bytecode=KOD_BAJTOWY&allFiles=WARTOSC_LOGICZNA.
```

W tym przypadku wartość KOD_BAJTOWY użytkownik musi zamienić na swój kod bajtowy, który chce przeanalizować, a WARTOSC_LOGICZNA na-

leży zamienić na napis *true* lub *false*, w zależności od tego, czy chcemy w rezultacie otrzymać wszystkie dopasowania implementacji, czy chcemy otrzymać tylko pierwszą dziesiątkę najbardziej pasujących. Dodatkowo w nagłówku należy wprowadzić parametr **Content-Length**, którego wartością powinna być długość wysyłanych danych. Przykładowo dla zawartości:

```
bytecode=234a12b45&allFiles=true
```

fragment nagłówka powinien wyglądać następująco:

```
Content-Length: 32
```

W rezultacie zapytania, użytkownik otrzyma status HTTP 200, listę składająca się z haszu pliku oraz współczynnika dopasowania danego pliku w formacie JSON. W sytuacji, gdy dla konkretnego kodu bajtowego nie udało się dopasować żadnej implementacji, zwrócony zostanie status HTTP 404 mówiący o tym, że nic nie udało się znaleźć.

Przesyłanie nowego kodu źródłowego za pomocą API

W aplikacji istnieje możliwość przesłania kodu źródłowego w sposób programowy, wykorzystując API. W odróżnieniu od sposobu opisanego powyżej, teraz nie będą używane atrybuty z nazwami, tylko zostanie wysłane żądanie HTTP, którego ciałem będzie czysty tekst, czyli będzie wprowadzony tylko kod źródłowy bez wykorzystywania nazw atrybutów.

W celu wysłania takiego żądania najpierw użytkownik musi zostać uwierzytelniony, w innym przypadku próba dodania nowego kodu źródłowego do aplikacji ukończy się niepowodzeniem.

Użytkownik w celu uwierzytelnienia musi wysłać metodą POST w żądaniu HTTP swój login i hasło na adres /**login**. W odpowiedzi otrzyma identyfikator sesji, który znajduje się pod zmienną **JSESSIONID**. Aby wykorzystać zasoby,

które wymagają uwierzytelniania, należy dodać identyfikator sesji do nagłówka wykorzystując atrybut Cookie. Przykładowa linia nagłówka zawierająca informacje o sesji:

```
Cookie: JSESSIONID=50F52FEF5F831800C7E3BD080EF7B495
```

Od tej pory, podczas korzystania z zasobów wymagających uwierzytelnienia należy wysyłać wszystkie żądania HTTP wraz z polem ciasteczka.

Posiadając identyfikator sesji oraz dane, które użytkownik chce przesłać, należy teraz zmodyfikować przykładowe żądanie HTTP z listingu 3.1. Pierwszym krokiem jest ustawienie metody HTTP na POST. Następnie należy wskazać ścieżkę pod którą chcemy wykonać operacje, w tym przypadku /api/solidityFiles. Ostatnią rzeczą będzie wprowadzenie danych poniżej nagłówka oraz ustawienie długości przesyłanych danych w atrybucie *Content-Length*

Po pomyślnym przesłaniu kontraktu, w odpowiedzi od serwera użytkownik otrzymuje status HTTP 200, który informuje osobę korzystającą z API, że wszystko poszło pomyślnie. W dodatku w użytkownik otrzymuje przesłany przez niego kod źródłowy, hasz stworzony na podstawie kodu źródłowego oraz listę znalezionych sygnatur funkcji. Na listingu 3.2 można zaobserwować przykładowe dane w formacie JSON, zwracane przez API po prawidłowym dodaniu nowego kodu źródłowego.

Listing 3.2: Przykładowa odpowiedz w formacie JSON

```
1 {
2     "sourceCodeHash": "0
          x80b739cbf3e89eeea1a96d9cfcf0567ddfef2af82eb14d2c5f97862f71e56265
          ",
3     "sourceCode": "pragma solidity ^0.4.25;\n\ncontract Hello {\n\n
          \tstring public message;\n\t\n\tconstructor(string
          initialMessage) public {\n\t\tmessage = initialMessage;\n\t
```

```
        }\n\t\n\tfunction setMessage(string newMessage) public {\n\
        t\tmessage = newMessage;\n\t}\n\n}",
    "solidityFunctions": [
        {
            "selector": "e21f37ce",
            "signature": "message()"
        },
        {
            "selector": "368b8772",
            "signature": "setMessage(string)"
        }
    ]
}
```

Pod atrybutem **sourceCodeHash** została umieszczona wartość hasza pliku źródłowego, natomiast pod atrybutem **sourceCode** znajduje się przesłana przez użytkownika implementacja w języku Solidity. Następnym elementem odpowiedzi jest **solidityFunction**, pod tą nazwa znajduje się lista funkcji znalezionych na podstawie przesłanej implementacji.

3.2 Przedstawienie architektury

W tym podrozdziale omówię architekturę aplikacji wykorzystywaną do zaimplementowania opisanych w sekcji 3.1 funkcjonalności. Pokaże z jakich komponentów składa się aplikacja oraz opisze zależności miedzy nimi.

Na rysunku 3.4 została przedstawiona architektura całej aplikacji w której skład wchodzi sześc komponentów.

3.2.1 Połączenie z bazą danych

Baza danych - jest to czesc aplikacji odpowiedzialna za obsluge bazy danych. W przypadku tej aplikacji komponent tez zostal zrealizowany za pomoca fra-

meworka Spring Boot oraz projektu Spring Data MongoDB umozliwaiajacego integracje z baza danych MongoDB.

Aplikacja jest zarzadzana za pomoca narzedzia Apache Maven ulatwiajacego budowanie projektu oraz zarzadzanie zaleznosciami. Konfiguracja narzedzia Maven dla projektu jest zapisywana w pliku **pom.xml**

Listing 3.3: Przyklad dodania zależności w pliku pom.xml

```
1       <dependency>
2           <groupId>org.springframework.boot</groupId>
3           <artifactId>spring-boot-starter-data-mongodb</artifactId
                >
4       </dependency>
```

Na listingu 3.3 zostal przedstawiony fragment pliku **pom.xml** odpowiedzialny za dodanie modulu spring-boot-start-data-mongodb umozliwaiajacego integracje frameworka Spring Boot z baza danych MongoDB. Wszystkie pozostale zewnetrzne moduly zostaja dodawane w analogiczny sposob do aplikacji.

Kolejnym krokiem w zapewnieniu polaczenia aplikacji z baza danych MongoDB jest skonfigurowanie pliku application.properties wykorzystywanego przez framework Spring Boot. Przykladowa zawartosc pliku application.properties zostala przedstawiona na listingu 3.4.

Listing 3.4: Przyklad dodania zaleznosci w pliku pom.xml

```
1 spring.data.mongodb.uri=mongodb://${ADMIN_DB_LOGIN}:${
    ADMIN_DB_PASSWORD}@ds129904.mlab.com:29904/${
    DATABASE_NAME_CONTRACT}
2 admin.login=${ADMIN_LOGIN}
3 admin.password=${ADMIN_PASSWORD}
```

W przypadku konfiguracji połączenia z bazą danych ważne jest podanie URI do bazy danych w atrybucie:

`spring.data.mongodb.uri`

Spring umożliwia w pliku application.properties wykorzystanie zmiennych środowiskowych, w tym celu nalezy skorzystac ze skladni

`${NAZWA_PLIKU}`

Dzięki temu można zabezpieczyć aplikacje przed wyciekiem poufnych informacji, ponieważ wszystkie wrażliwe informacje są wyciągnięte poza kod źródłowy aplikacji. W tym przypadku na listingu 3.4 widać ze dane logowania do bazy dan

W celu wykorzystania bazy danych w kodzie Javy, nalezy utworzyc repozytoria ktora pozwola zapisywac obiekty bo bazy danych. Utworzone w aplikacji repozytorium mozna zobaczyc na listingu ponizej:

Listing 3.5: Stworzenie repozytorium za pomoca Spring Data MongoDb

```
1 @Repository
2 interface SolidityFileRepository extends MongoRepository<
    SolidityFile, String> {
3
4     @Query("{\"solidityFunctions\": {$elemMatch: {\"selector\": {$
        in: ?0}}}}")
5     List<SolidityFile> findSolidityFilesBySelectorContainsAll(List<
        String> functionSelector);
6
7     Optional<SolidityFile> findBySourceCodeHash(String
        sourceCodeHash);
8 }
```

W pierwszej lini znajduje sie @Repository, jest to adnotacja informujaca framework, ze interfejs ten jest abstrakcja repozytorium, ktore utworzy framework.

Kolejna adnotacja jest @Query wewnatrz ktorej znajduje sie zapytanie do bazy danych MongoDB, pytajace o liste plikow, w ktorych znajduja sie przekazane przez uzytkownika selektory funkcji.

Repozytorium to jest wykorzystywane do operacji z baza danych, wszystkie dane wyciagniete z bazy danych zostaja zapisywane w postaci obiektow zdefiniowanych w jezku Java. Aby to umozliwiec nalezalo utworzyc odpowiednie klasy do ktorych framework bedzie zapisywal informacje pobrane z bazy danych. Te same klasy sa tez wykorzystywane w momencie gdy chcemy zapisac cos do bazy danych. W tym przypadku do zapisu i odczytu jest wykorzystywana klasa SolidityFily, ktorej implementacje widac na listingu dsdddd.

Listing 3.6: Przyklad klasy wykorzystywanej przez Spring Data MongoDb

```
1  public class SolidityFile {
2
3      @Id
4      private final String sourceCodeHash;
5      private final String sourceCode;
6      private final Set<SolidityFunction> solidityFunctions;
7
8      SolidityFile(String sourceCodeHash, String sourceCode, Set<
           SolidityFunction> solidityFunctions) {
9          requireNonNull(sourceCodeHash, "Expected not-null
               sourceCodeHash");
10         requireNonNull(sourceCode, "Expected not-null sourceCode");
11         requireNonNull(solidityFunctions, "Expected not-null
               solidityFunctions");
12         this.sourceCodeHash = sourceCodeHash;
13         this.sourceCode = sourceCode;
14         this.solidityFunctions = solidityFunctions;
15     }
16
17     public String getSourceCodeHash() { return sourceCodeHash; }
18
19     public String getSourceCode() { return sourceCode; }
```

```
20
21      public Set<SolidityFunction> getSolidityFunctions() { return
            solidityFunctions; }
22
23      @Override
24      public String toString() {
25          return "SolidityFile{" + "sourceCodeHash='"
26                  + sourceCodeHash
27                  + '\'' + ", sourceCode='" + sourceCode + '\''
28                  + ", solidityFunctions=" + solidityFunctions + '}';
29      }
30
31      @Override
32      public boolean equals(Object o) {
33          if (this == o) return true;
34          if (!(o instanceof SolidityFile)) return false;
35          SolidityFile that = (SolidityFile) o;
36          return Objects.equals(sourceCodeHash, that.sourceCodeHash)
                &&
37                  Objects.equals(sourceCode, that.sourceCode) &&
38                  Objects.equals(solidityFunctions, that.
                        solidityFunctions);
39      }
40
41      @Override
42      public int hashCode() {
43          return Objects.hash(sourceCodeHash, sourceCode,
                solidityFunctions);
44      }
45  }
```

Klasa ta sklada sie z atrybutow, z ktorych jeden atrybut powinien byc z adnotacja @Id ktora informuje Springa, ze jest to identyfikator klasy. Poza atrybutami znajduje sie konstruktor klasy ktory przypisuje parametron konstruktora, wartosci w atrybutach klasy. Dodatkowo konstruktor posiada metody upewniajace osoby korzystajaca z klasy ze nie zostana wprowadzone tam wartosci null.

Kolejnym elementem tej klasy sa akcesory, nadpisana metoda toString na

taka ktora bedzie przedstawiac stan obiektu oraz nadpisane metody equals oraz hashCode, które są wykorzystywane podczas porownywania lub korzystania z kolekcji zaimplementowanych w Javie.

Klasa SolidityFile przechowuje liste obiektow klasy SolidityFunction. Klasa SolidityFunction zostala zaimplementowana w analogiczny sposob z taka roznica ze posiada ona dwa atrybuty oraz nie posiada adnotacji @Id. Adnotacja @Id nie jest w tym przypadku konieczna poniewaz ta klasa nie bedzie wykorzystywana jako niezalezny obiekt w bazie danych, tylko bedzie zawsze czescia jakiegos pliku, czyli czescia obiektu klasy SolidityFile.

Bytecode Controller - jest to komponent odpowiedzialny ze obsluge żadan http wysylanych przez klienta do aplikajci. Kontroler ma za zadanie mapowanie adresu /bytecode obslugujac metody GET oraz POST.

W przypadku gdy uzytkownik wejdzie na wymieniony adres realizowana jest metoda GET, wtedy kontroler za pomoca frameworka Spring Boot oraz Spring Web MVC wysyla do klienta kod HTML. W rezultacie uzytkownik otrzymuje strone ktora nastepnie umozliwia przeslanie kodu bajtowego.

@GetMapping("/bytecode") public String showPage(Model model) requireNonNull(model, "Expected not-null model");

model.addAttribute("implementationsWithValueOfMatch", new HashMap<>()); return "bytecode-page";

Po poprawnym wyrenderowaniu kodu html, uzytkownik ma do dyspozycji formularz w ktorym moze wprowadzic by

3.3 Wyszukiwanie sygnatur funkcji w kodzie źródłowym

Literatura: [2, ?]. TODO

3.4 Wyszukiwanie selektorów funkcji w kodzie bajtowym

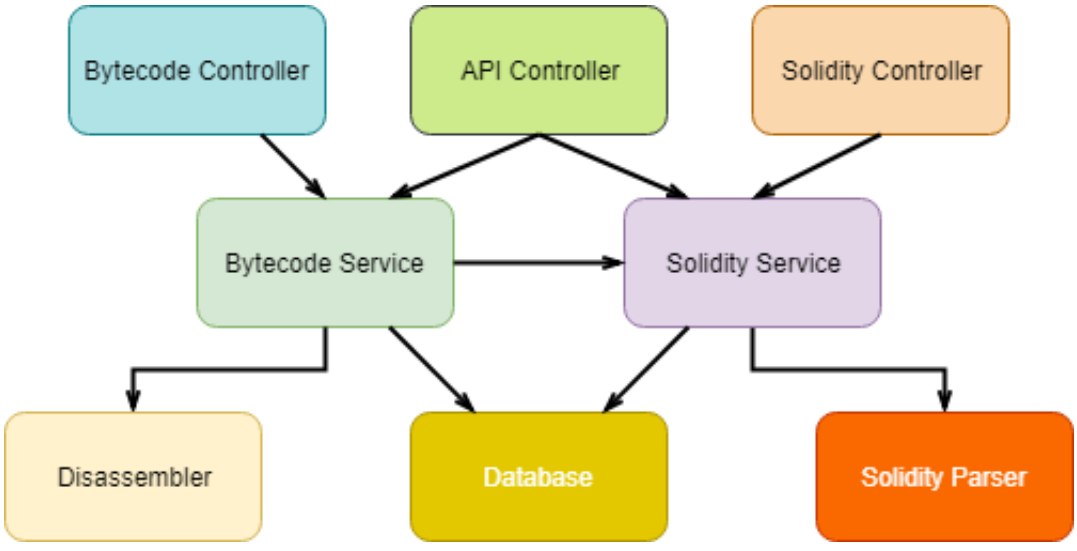
Literatura: [2, ?]. TODO

3.5 Dopasowywanie implementacji na podstawie kodu bajtowego

Literatura: [2, ?]. TODO

3.6 Wykorzystane technologie

Literatura: [2, ?]. TODO



Rysunek 3.4: Architektura aplikacji

Bibliografia

[1] Network Working Group `https://www.ietf.org/rfc/rfc2616.txt`.

[2] Bibliografia 2. *Nazwa*.

Spis tabel

Spis rysunków

Spis listingów