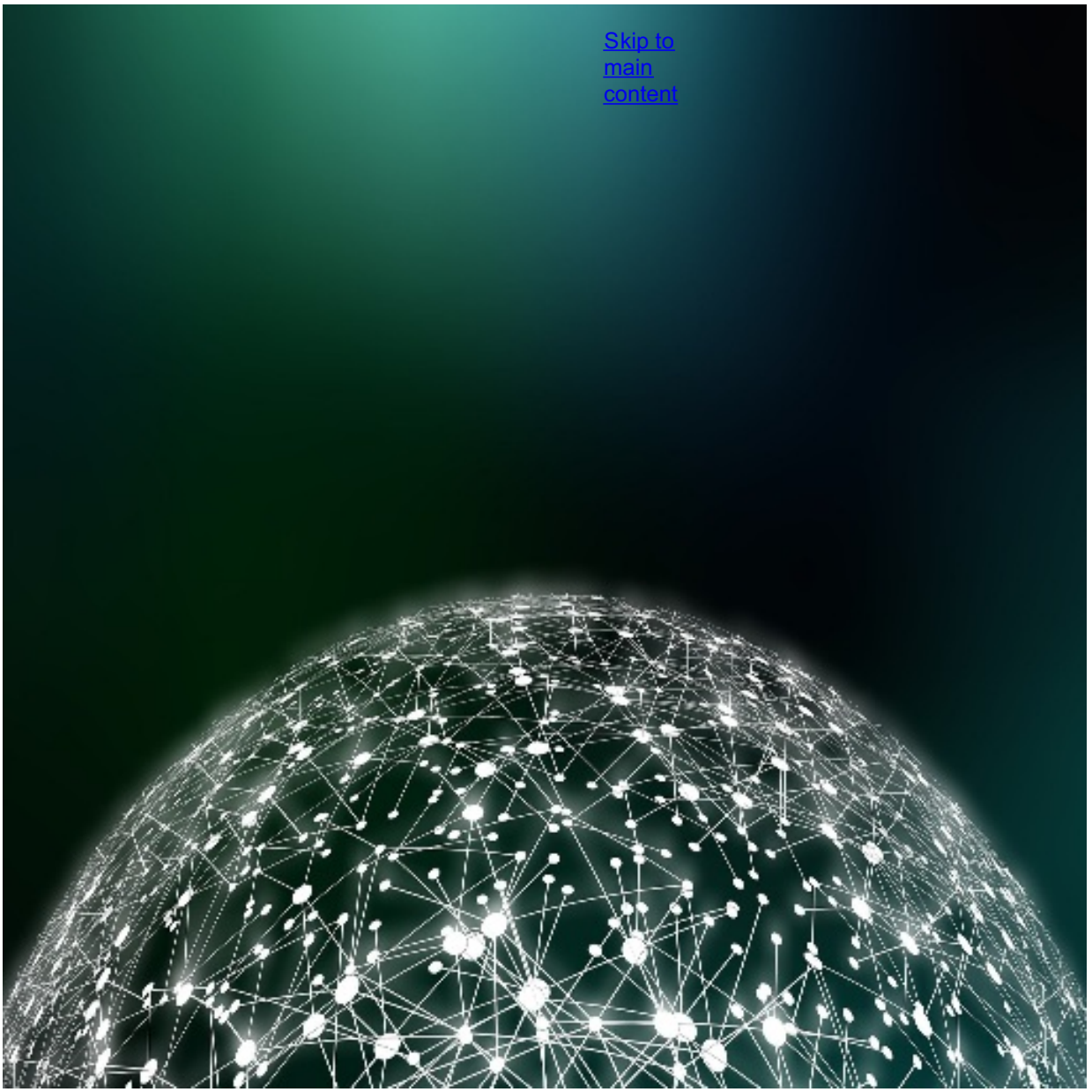

Smart Contracts

Below you will find the smart contracts necessary for the successful implementation of the publication platform and for a functioning submission process. For reasons of capacity and overview, these have been divided into different contracts to ensure better performance:

The individual contracts were written with the Solidity programming language in version 0.8.2.

- Contoller
- Meta Data
- Validation

The individual contracts are linked to each other by means of inheritance.

Controller

The Controller Contract is used to control all functions. For example, the creation of submissions or the validation of input data.

The controller contract inherits from the contract data to get access to the different data. Furthermore, the controller accesses the validation library in order to be able to check the inputs for formal correctness. All submitted submissions receive an individual ID to be able to assign them exactly and are stored in a dynamic array.

For each submitted submission, it is checked directly for formal correctness and will not be submitted if it is incorrect.

```
pragma solidity ^0.8.2;
pragma experimental ABIEncoderV2;
import "./Data.sol";
import "./Validation.sol";

//Authors: Alexander Albert, Maximilian Bonkosch, Marco Spraul

/** @title Controller inerhits from data.sol */
contract Controller is Data {

 // Import Validation library for structs
 using Validation for Student;
 using Validation for Thesis;
 using Validation for Examiner;
 address submissionAddress;


 // counter to give every Submission an unique id
 uint public SubCount = 0;

 // all submissions are saved in an dynamic array
 Submission[] public submissions;


 /// @dev function to validate the arguments typed in by the student
 /// @param student (Data.sol) ,
 /// return bool for validation was sucessfull (=true) or not sucessfull (=false)
 function validateStudent (Student memory student) public pure returns (bool){  //
   // Student memory student = Student("String Student", "String Student", "String Student",
   return student.testStudent();
 }


 /// @dev function to validate the arguments typed in by the student
 /// @param examiner1 is a struct of Data.sol
 /// @return bool for validation was sucessfull (=true) or not sucessfull (=false)
 function validateExaminer (Examiner memory examiner1)public pure returns (bool){
   return examiner1.testExaminer();
 }

 /// @dev function to validate the arguments typed in by the student
 /// @param thesis1 is a struct of Data.sol
 /// @return bool for validation was sucessfull (=true) or not sucessfull (=false)
 function validateThesis (Thesis memory thesis1)public pure returns (bool){
   return thesis1.testThesis();
 }
```

```
/// @dev function to create a submission and assign the submission to an ID (SubCount) wi
/// @param student Is a struct of Data.sol which contains the data of the student
/// @param examiner Is a struct of Data.sol which contains the data of the examiner
/// @param thesis Is a struct of Data.sol which contains the data of the thesis
/// @return uint the ID is assigned to the submission
function createSubmission(Student memory student, Examiner memory examiner, Thesis m
  if(validateStudent(student)&&(validateExaminer(examiner))&&(validateThesis(thesis))){
    submissions.push(Submission(SubCount, student, examiner, thesis, false, false, false));
    SubCount ++;
    return (SubCount, true);
  }else return (SubCount, false);
}

/// @dev function receive all submissions that have been created
/// @param -
/// @return Submission[] memory All Submissiosn that have been created
function getSubmissions() public returns(Submission[] memory) {
  return submissions;
}

/*
/// @dev function to show whether the examiner accepted the responsibility for the the thesis
/// @param _id ID which is assigned to a specific submission
/// @return Submission memory Submission which is assigned to the ID
function acceptSubmission(uint _id) public returns (Submission memory){
  submissions[_id].accepted = true;
  return submissions[_id];
}

/// @dev function to show wheter the examiner has corrected the submission
/// @param _id ID which is assigned to a specific submission
/// @return Submission memory Submission which is assigned to the ID
function correctSubmission(uint _id) public returns (Submission memory){       //anpassen
  submissions[_id].corrected = true;
  return submissions[_id];
}
/// @dev function to show wheter the examiner has verified the submission
/// @param _id ID which is assigned to a specific submission
/// @param date To set a timestamp
/// @return Submission memory Submission which is assigned to the ID
function verifySubmission (uint _id, string memory date) public returns (Submission memory
  submissions[_id].verified = true;
  submissions[_id].thesis.uploaddate = date;
  return submissions[_id];
}


/// @dev function to upload a an updated version of the thesis, no overwriting of the orgional
/// @param _id ID which is assigned to a specific submission
/// @param _adr Address of the Person who wants to change the IPFS Hash
/// @param ipfs Contains the new ipfs string from the updated thesis
// @return
function changeIpfsHash(uint _id, address _adr, string memory ipfs,string memory date)pub
  require ((submissions[_id].student.student_address ==_adr || submissions[_id].examiner.ex
  submissions[_id].thesis.ipfs_hash[1] = ipfs;
  submissions[_id].thesis.lastupdated = date;
}


/// @dev function to change the E-mail of the students in case of student have a new mail or
/// @param _id ID which is assigned to a specific submission
/// @param _adr Address of the student who wants to change the E-Mail
/// @param mail Contains the Mail Address
function changeStudentMail(uint _id, address _adr, string memory mail) public {
  require((submissions[_id].student.student_address ==_adr ) && Validation.testString(mail)=
  submissions[_id].student.email_student = mail;
}

*/
```

```
function createSub() public {
        Student memory student = Student("String Student", "String Student", "String Student",
        Examiner memory examiner = Examiner("String Examiner", "String Examiner", "String E
        Thesis memory thesis = Thesis("String Thesis", "String Thesis", "String Thesis", "String
87']);
        createSubmission(student, examiner, thesis);
    }

}
```

Meta-Data

The Meta-Data Contract is divided into four diffrent structs: Studen, Exminer, Thesis and Submission

This contract contains the data required for the transaction of uploading a publication. On the one hand, these are data of the submitter, such as his name or his ipfs address. On the other hand structure contains the data of the examiner. The third structure contains the data required for the thesis and a fourth, which is called Submission. This contains all the data of the student, the examiner and the thesis as well as an id for assignment and various bools for querying the assessment status of the thesis.

```
pragma solidity ^0.8.2;
// import "./Submission11.sol";
//Mutter

contract Data{


 struct Student {
   string first_name_student; //!
   string last_name_student; //!
   string email_student; //!
   string field_of_study;
   string study_interests;
   string researchgate_profil;
   string google_scholar_profil;
   address student_address; //!
 }

 struct Examiner {
   string first_name_e; //!
   string last_name_e; //!
   string email_e; //!
   string institution_name; //!
   string department_name;
   string instituion_postal_address;
   string instituion_webite;
   string position_examiner;
   string examiner_profile_website; //!
   address examiner_address; //!
 }

 struct Thesis {
   string thesis_title; //!
   string thesis_topics; //!
   string applied_research_methods;
   string type_of_thesis; //!
   string thesis_language; //!
   string institution_name_thesis;
   string department_name_thesis;
   string instituion_postal_address_thesis;
   string instituion_website_thesis;
   string additional_comments;
   string uploaddate; //!
   string lastupdated; //!
   string[2] ipfs_hash;
 }

 struct Submission {
   uint id;
   Student student;
   Examiner examiner;
   Thesis thesis;
   // examniner accepts allocation
   bool accepted;
   // examiner finished correction
   bool corrected;
   // examiner marks submission as verified
   bool verified;
 }

}
```

Validation

The validation was implemented as a libary. This has the great advantage that no storage space is required, thus reducing gas costs.

The libary contains functions, which are used to validate the to check the input of the

submitter in the function "Sumbit Your Thesis" in the frontend.

The various strings and the ipfs hash run through the function "testString", in which the individual letters are checked for correctness. In addition, the length of the address and the ipfs hash is checked for correctness in the function "testLengthAddress".

The logic of the "testString" in the current version is adapted to the German and English alphabets and also includes umlauts. Other alphabets such as Danish umlauts could be implemented later and integrated in a future version adapted to the respective country.

```
pragma solidity ^0.8.2;
pragma experimental ABIEncoderV2;
import "./Data.sol";


library Validation  {


 function testStudent(Data.Student memory student) public pure returns (bool) {

   if(!testString(student.first_name_student)) {
     return false;
   }

   if(!testString(student.last_name_student)) {
     return false;
   }

   if(!testString(student.field_of_study)) {
     return false;
   }

   if(!testString(student.study_interests)) {
     return false;
   }

   if(!testString(student.researchgate_profil)) {
     return false;
   }

   if(!testString(student.google_scholar_profil)) {
     return false;
   }

   if(!testLengthAddress(student.student_address)) {
     return false;
   }

   else {
     return true;
   }
 }


 function testExaminer(Data.Examiner memory examiner) public pure returns (bool) {

   if(!testString(examiner.first_name_e)) {
     return false;
   }

   if(!testString(examiner.last_name_e)) {
     return false;
   }

   if(!testString(examiner.email_e)) {
     return false;
   }

   if(!testString(examiner.institution_name)) {
```

```
    return false;
  }

  if(!testString(examiner.department_name)) {
    return false;
  }

  if(!testString(examiner.instituion_postal_address)) {
    return false;
  }

  if(!testString(examiner.instituion_webite)) {
    return false;
  }

  if(!testString(examiner.position_examiner)) {
    return false;
  }

  if(!testString(examiner.examiner_profile_website)) {
    return false;
  }

  if(!testLengthAddress(examiner.examiner_address)) {
    return false;
  }

  else {
    return true;
  }
}


function testThesis(Data.Thesis memory thesis) public pure returns (bool) {

  if(!testString(thesis.thesis_title)) {
    return false;
  }

  if(!testString(thesis.thesis_topics)) {
    return false;
  }

  if(!testString(thesis.applied_research_methods)) {
    return false;
  }

  if(!testString(thesis.type_of_thesis)) {
    return false;
  }

  if(!testString(thesis.thesis_language)) {
    return false;
  }

  if(!testString(thesis.institution_name_thesis)) {
    return false;
  }

  if(!testString(thesis.department_name_thesis)) {
    return false;
  }

  if(!testString(thesis.instituion_postal_address_thesis)) {
    return false;
  }

  if(!testString(thesis.instituion_website_thesis)) {
    return false;
```

```
    }

    if(!testString(thesis.additional_comments)) {
      return false;
    }

    if(!testString(thesis.uploaddate)) {
      return false;
    }

    if(!testString(thesis.lastupdated)) {              // IPFS Hash hinzufügen
      return false;
    }

    if(!testLengthHash(thesis.ipfs_hash[thesis.ipfs_hash.length-1])) {
      return false;
    }

    else {
      return true;
    }
  }


  function testString(string memory text) public pure returns (bool) {
    bytes memory b = bytes(text);
    bool correct = true;
    for(uint i; i<b.length; i++){
      bytes1 char = b[i];
      if(
        !(char >= 0x30 && char <= 0x39) && //9-0
      !(char >= 0x41 && char <= 0x5A) && //A-Z
      !(char >= 0x61 && char <= 0x7A) && //a-z
      //,ä,Ä,ö;Ö,ü,Ü,ß,.,_,@,-
      !(char ==0xC3)&&
      !(char ==0xA4)&&    //ä
      !(char ==0x84)&&    //Ä
      !(char ==0xB6)&&    //ö
      !(char ==0x96)&&    //Ö
      !(char ==0xBC)&&    //ü
      !(char ==0x9C)&&    //Ü
      !(char ==0x9F)&&    //ß
      !(char ==0x2E)&&    //.
      !(char ==0x5F)&&    //_
      !(char ==0x40)&&    //@
      !(char ==0x2D)      //-
      ) {
        correct = false;
      }
    }
    return correct;
  }


  function testLengthAddress(address addr) public pure returns (bool) {   // Teste Länge der S
e
    bytes memory a = abi.encodePacked(addr);
    if(a.length != 20) {
      return false;
    } else {return true;}
  }


  function testLengthHash(string memory hash) public pure returns(bool){  // Teste Länge des
    bytes memory h = bytes(hash);
    if(h.length == 20) { return false;
    } else {return true;}
  }
}
```

// ## Migration

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract Migrations {
address public owner = msg.sender;
uint public last_completed_migration;

modifier restricted() {
require(
msg.sender == owner,
"This function is restricted to the contract's owner"
);
_;
}

function setCompleted(uint completed) public restricted {
last_completed_migration = completed;
}
}
```

Edit this page

Previous
« Possibilities of PEER
Next
Estimated Gas Cost »

Controller
Meta-Data
Validation

Docs

Overview

Getting Started

Frontend

Blockchain

Further Information

More

GitHub