Unit Test Plan (UTP)

2IPE0 SOFTWARE/WEB ENGINEERING PROJECT

November 8, 2019

GROUP 1

L. van de Beek	1015669
K.K.J. Burgers	1004884
M. Ghanem	1036435
A.N. Groote Woortmann	1000308
M.J.B. Keizer	0988425
B. van Leeuwen	0996101
L.A. Roozen	0948743
W.J.A. van Santvoort	1010775
H.P.A. Swinkels	1005900

TU/e

version: 1.1
supervisor: Serguei Roubtsov
customer: Dimas Satria

Abstract

This document is a description of the unit test plan for CloudFarmer, developed by CloudFarmers. CloudFarmer is web-based application that is developed as part of the Software Engineering Project (2IPE0) at the Technical University of Eindhoven. This document complies with the ESA software standards [1]

Contents

I Document Status Sheet

I.I General

Document title:	Unit Test Plan (UTP)	
Document identifier:	UTP/1.1	
Authors:	L. van de Beek	1015669
	K.K.J. Burgers	1004884
	M. Ghanem	1036435
	A.N. Groote Woortmann	1000308
	M.J.B. Keizer	0988425
	B. van Leeuwen	0996101
	L.A. Roozen	0948743
	W.J.A. van Santvoort	1010775
	H.P.A. Swinkels	1005900
Document status:	Version 1.1	

I.II Document History

Version	Date	Authors	Reason
0.1	2-10-2019	Mohamed	Initial version.
0.2	30-10-2019	Willem	Added 3.1 and 4
1.0	4-11-2019	Han	First complete version
1.1	7-11-2019	Han	Feedback implementation

II Document Change Records

Version	Date	Section	Reason
0.1	2-10-2019	All	Initial version.
0.2	30-10-2019	3 and 4	Added 3.1 and 4
1.0	4-11-2019	3, 5 and 6	Added 3.2, 5 and 6
1.1	7-11-2019	2, 3 and 6	Feedback implementation

1 Introduction

1.1 Purpose

The UTP (the Unit Test Plan) document will contain all the unit tests for testing the CloudFarmer application. To make sure all parts of the software work as intended. The test will be grouped based on the subject the tests are about, this is to make the UTP more structured. The tests results are presented and described in this document.

1.2 Overview

This document consists of six chapters including this introductory chapter. Chapter 2 will be introducing which features need to be tested, together with a description of the criteria which need to be passed to pass each of the unit tests. In Chapter 3, all tests are provided as well as their expected output. Chapter 4 will describe the testing procedure that should be used to execute all the tests, which are described in the previous chapter, correctly. Then chapter 5 will show the coverage results for the tests given in Chapter 3. The last chapter, chapter 6, will provide the test report obtained by completing this Unit Test Plan.

1.3 List of definitions

1.3.1 Definitions

Name	Definition
Active farm	An active farm is the farm to which the map, history, live, field and farm settings view are currently referencing.
Active field	An active field is the field to which the field view is currently referencing.
Active crop field	An active crop field is the crop field to which the field view is currently referencing.
Available farm	A farm of which the user is allowed to see the information.
CloudFarm	Database server for the application.
CloudFarmer	Application as defined by the project group and the customer.

CloudFarmers	The project group tasked to create the aforementioned application.
Crop	The virtual representation of a physical crop.
Crop field	The virtual representation of a physical crop field. A field can contain multiple crop fields.
Crop field information/ the information of a crop field	All the information about a specific crop field; location, crop type, crop year, crop season, area.
Customer	The person who assigned the task to create the application to the project group. In this case Dimas Satria.
Dacom	The company that provided us with their crop database and their API for extracting information from this database.
Data sources	List of all the sources of data that are known by the application.
Equipment	The sensors in a field
Farm	The virtual representation of a physical farm. A user is allowed to have more than one virtual farm.
Farmer (Role)	A person who works on the physical farm.
Farm admin (Role)	A user with unrestricted access to all data associated with a specific farm. (E.g farm information/permissions, field information/permissions ... etc)
Farm information/ the information of a farm	All the information about a specific farm; ID number, name, address, postal code, country, email, phone number, website.
Farm permissions	A set of rules that determine which users are able to read, write or delete farm information.
Field	The virtual representation of a physical field. A farm can have multiple virtual fields.
Field information/ the information of a field	All the information about a specific field; location, field name, area, size in hectares, soil type.
Field permissions	A set of rules that determine which users are able to read, write or delete field information.
General user (Role)	A user that is a member of a farm without permissions.

Member	A user is a member of a farm, if they have a role assigned to them on that farm.
Observation	Metadata.
Observation Data	A general class of data referring to Dacom's data and/or WolkyTolky data and/or data input by the user
Physical crop field	A physical crop field is a physical field that contains at least one type of crop.
Physical farm	A physical farm that has a physical building.
Physical field	A physical field is a patch of dirt, clay, or some type of rock that is owned by a farmer, on which they are able to grow crops.
Researcher (Role)	A person who carries out academic or scientific research.
Selected farm	A farm that the user has selected.
Slug	A set of attributes of an equipment model : brand name, model, model year, series, version
User	A person that interacts with the application.
User role	A role that can be assigned to a user within a farm which defines all the field permissions and farm permissions of the user. These different roles are defined in the definitions and have the tag (role) at the end of them.
Views	The map view, history view, live view, fields view, settings view and personal settings view.
Widgets	A component of an interface that enables a user to perform a function or access a service.

1.3.2 Abbreviations and Acronyms

Acronym	Meaning
CSV	Comma-Seperated Values (file format)
DIPPA	Data Integration Platform for Precision Agriculture
ESA	European Space Agency
ID	Identity
JSON	JavaScript Object Notation
XML	Extensible Markup Language
JWT	JSON Web Tokens
OAuth	Open authorization
PDEng	Professional Doctorate Engineering
PNG	Portable Network Graphics (file format)
SEP	Software Engineering Project
TU/e	Eindhoven University of Technology
URD	User Requirements Document
SDD	Software Design Document
UTP	Unit Test Plan

1.4 List of references

[1] *E.B. for software Standardisation and Control*. ESA software standards, 1991.
[2] *Software Design Document*. Technische Universiteit Eindhoven, 2019.
[3] *User Requirements Document*. Technische Universiteit Eindhoven, 2019.

2 Test Plan

2.1 Test Items

The application tested is CloudFarmer. This application is comprised of the website which users can access as well as the server that contains farm data storage and man-

ages user authentication and authorization as well. Further details about CloudFarmer, including unimplemented parts, can be found in the URD [3] and the SDD [2]

2.2 Features to be tested

All features that have been implemented in CloudFarmer will be tested except the external API's used.

2.3 Test Deliverables

Prior to the execution of the unit test plan, the following items have to be delivered:

- The final version of the CloudFarmer web application.
- The first four chapters of this document.

After executing the unit test plan the following documents have to be delivered:

- The last two chapters of this document.
- Problem reports should any occur.

2.4 Testing Tasks

The following tasks have to be accomplished so that the tests can be executed according to the procedures mentioned in Chapter 4:

- Designing unit tests for all features that will be tested
- Ensuring that all the environment needs mentioned in section 2.5 are satisfied

2.5 Environmental Needs

There are two prerequisites for performing the unit test plan in this document:

- A running development machine with the specifications mentioned in section 4.1 of the SDD [2]
- Completion of the tasks mentioned in the previous subsection.

2.6 Test Case Pass or Fail Criteria

A test case passes if its respective expected output is realised. The overall unit test plan succeeds if every test described in Chapter 3 passes. Otherwise the overall unit test plan fails.

3 Test Case Specification

3.1 User interface

The tests of the user interface can be found in directory */farm-data-storage/stripefarmer/src*.

3.1.1 CropFieldController.test.ts

3.1.1.1 getCropFields()

- returns list of cropfields
- returns list of cropfields after refreshToken

3.1.1.2 getCropField()

- returns cropfield
- returns cropfield after refreshToken

3.1.1.3 addCropField()

- successfully add cropfield
- successfully add cropfield after refreshToken

3.1.1.4 updateCropField()

- successfully update cropfield
- successfully update cropfield after refreshToken

3.1.1.5 deleteCropField()

- successfully delete cropfield
- successfully delete cropfield after refreshToken

3.1.2 EquipmentController.test.ts

3.1.2.1 getEquipments()

- returns list of equipments
- returns list of equipments after refreshToken

3.1.2.2 getEqupiment()

- returns equipment
- returns equipment after refreshToken

3.1.2.3 addEquipment()

- successfully add equipment
- successfully add equipment after refreshToken

3.1.2.4 updateEquipment()

- successfully update equipment
- successfully update equipment after refreshToken

3.1.2.5 deleteEquipment()

- successfully delete equipment
- successfully delete equipment after refreshToken

3.1.3 EquipmentInfoController.test.ts

3.1.3.1 getEquipmentInfo()

- returns equipment information
- returns equipment information after refreshToken

3.1.3.2 addEquipmentInfo()

- successfully add equipment information
- successfully add equipment information after refreshToken

3.1.3.3 updateEquipmentInfo()

- successfully update equipment information
- successfully update equipment information after refreshToken

3.1.3.4 deleteEquipmentInfo()

- successfully delete equipment information
- successfully delete equipment information after refreshToken

3.1.4 EquipmentModelController.test.ts

3.1.4.1 getEquipmentModels()

- returns list of equipment models
- returns list of equipment models after refreshToken

3.1.4.2 getEquipmentModel()

- returns equipment model
- returns equipment model after refreshToken

3.1.4.3 addEquipmentModel()

- successfully add equipment model
- successfully add equipment model after refreshToken

3.1.5 FarmController.test.ts

3.1.5.1 getFarms()

- returns list of farms
- returns list of farms after refreshToken

3.1.5.2 addFarm()

- successfully add farm
- successfully add farm after refreshToken

3.1.5.3 getFarm()

- successfully returns farm
- successfully returns farm after refreshToken

3.1.5.4 updateFarm()

- succeful update farm
- return update farm after refreshToken
- farm does not exist is handled

3.1.5.5 deleteFarm()

- successfully delete farm
- successfully delete farm after refreshToken
- farm does not exist is handled

3.1.6 FieldController.test.ts

3.1.6.1 getFarmFields()

- returns list of fields
- returns list of fields after refreshToken

3.1.6.2 getFarmField()

- returns field
- returns field after refreshToken

3.1.6.3 addFarmField()

- successfully add field
- successfully add field after refreshToken

3.1.6.4 updateFarmField()

- successfully update field
- successfully update field after refreshToken

3.1.6.5 deleteFarmField()

- successfully delete field
- successfully delete field after refreshToken

3.1.7 MappingController.test.ts

3.1.7.1 getDataMaps()

- returns list of datamaps
- returns list of datamaps after refreshToken

3.1.7.2 addDataMap()

- successfully add datamap
- successfully add datamap after refreshToken

3.1.8 RoleController.test.ts

3.1.8.1 getUserRoleOnFarm()

- returns role
- returns role after refreshToken

3.1.8.2 changeUserRoleOnFarm()

- successfully update role
- successfully update role after refreshToken

3.1.8.3 createUserRoleOnFarm()

- successfully add role
- successfully add role after refreshToken

3.1.8.4 getAllUsersOnFarm()

- returns users with roles
- returns users with roles after refreshToken

3.1.8.5 getUserRoles()

- returns roles
- returns roles after refreshToken

3.1.9 TypesController.test.ts

3.1.9.1 getCountries()

- returns list of countries
- returns list of countries after refreshToken

3.1.9.2 getCropTypes()

- returns list of crop types
- returns list of crop types after refreshToken

3.1.9.3 getSoilTypes()

- returns list of soil types
- returns list of soil rypes after refreshToken

3.1.10 UserController.test.ts

3.1.10.1 login()

- handle invalid username or password
- handle successful login
- userId stored in cookies
- accessToken stored in cookies
- refreshToken stored in cookies
- loggedIn() = true

3.1.10.2 logout()

- accessToken deleted from cookies
- refreshToken deleted from cookies
- UserId deleted from cookies
- loggedIn() = false

3.1.10.3 register()

- handle email already exists
- handle successful login
- userId stored in cookies
- accessToken stored in cookies
- refreshToken stored in cookies

3.1.10.4 addFavorite() and deleteFavorite()

- handle delete favorite farm does not exist
- successfully add favorite farm
- successfully delete favorite farm

3.1.10.5 getUser()

- handle user not found error
- return user object
- return user object after refreshToken

3.1.10.6 updateUser()

- handle update user error
- successfully update user
- successfully update user after refreshToken

3.1.10.7 deleteUser()

- handle delete user error
- successfully delete user
- successfully delete user after refreshToken

3.1.11 WolkyTolkyEqController.test.ts

3.1.11.1 getWolkyTolkyEqs()

- returns list of WolkyTolky equipments
- returns list of WolkyTolky equipments after refreshToken

3.1.11.2 getWolkyTolkyEq()

- returns WolkyTolky equipment
- returns WolkyTolky equipment after refreshToken

3.1.11.3 addWolkyTolkyEq()

- successfully add WolkyTolky equipment
- successfully add WolkyTolky equipment after refreshToken

3.1.11.4 updateWolkyTolkyEq()

- successfully update WolkyTolky equipment
- successfully update WolkyTolky equipment after refreshToken

3.1.11.5 deleteWolkyTolkyEquipment()

- successfully delete WolkyTolky equipment cropfield
- successfully delete WolkyTolky equipment after refreshToken

3.2 Data linking service

The tests of the Data Linking Service can be found under the directory */farm-data-storage/microservice/DataLinkingService/src/tests*.

3.2.1 MappingController.test.ts

3.2.1.1 getDataMapIdByEquipmentModelId()

- returns a datamap id of an equipment model

3.2.1.2 getDataColumnsByEquipmentModelId()

- get all the data columns of an equipment model

3.2.1.3 getDataMaps()

- get the json string containing a list of datamaps

3.2.1.4 getEquipmentModelId()

- get the equipment model id of an equipment

3.2.2 UserController.test.ts

3.2.2.1 getAccessToken

- retrieve access token

3.2.3 EnumConverter.test.ts

3.2.3.1 stringToAccessibilityType()

- string: public
- string: private
- string: restricted

3.2.3.2 stringToAccessibilityType()

- enumValue: public
- enumValue: private
- enumValue: restricted

3.2.4 WolkyTolkyDataConverter.test.ts

3.2.4.1 transformLiveData()

- transform the data

3.2.4.2 transformLocationData()

- transform the location data

3.2.5 DataRetriever.test.ts

3.2.5.1 saveData()

- save the retrieved observation data to the database

3.2.6 WolkyTolkyDataRetriever.test.ts

3.2.6.1 getData()

- get data from a WolkyTolky station
- get data from a WolkyTolky station after a certain date

3.2.6.2 getLatestDataDate()

- get the latest date of the observation data from WolkyTolky

3.2.6.3 getLastSaved()

- tests whether a date is retrieved when the retriever last ran

3.2.6.4 run()

- test the run function of the WolkyTolky

3.2.6.5 saveData()

- save the retrieved observation data to the database

3.2.7 WolkyTolkyLiveDataRetriever.test.ts

3.2.7.1 getLatestData()

- get the latest observation data from the WolkyTolky sensor

4 Test Procedure

This chapter describes how to run the unit tests. Since the application itself and the data linking service are not necessarily in the same directory the procedure needs to be executed in a different directory for both services. Except for this the procedure is the same.
The application folder of the user interface is *cloudfarmer* in the root folder. The folder of the data linking service is */microservice/DataLinkingService* from the root folder. To run the unit tests, the following test procedure needs to be followed:

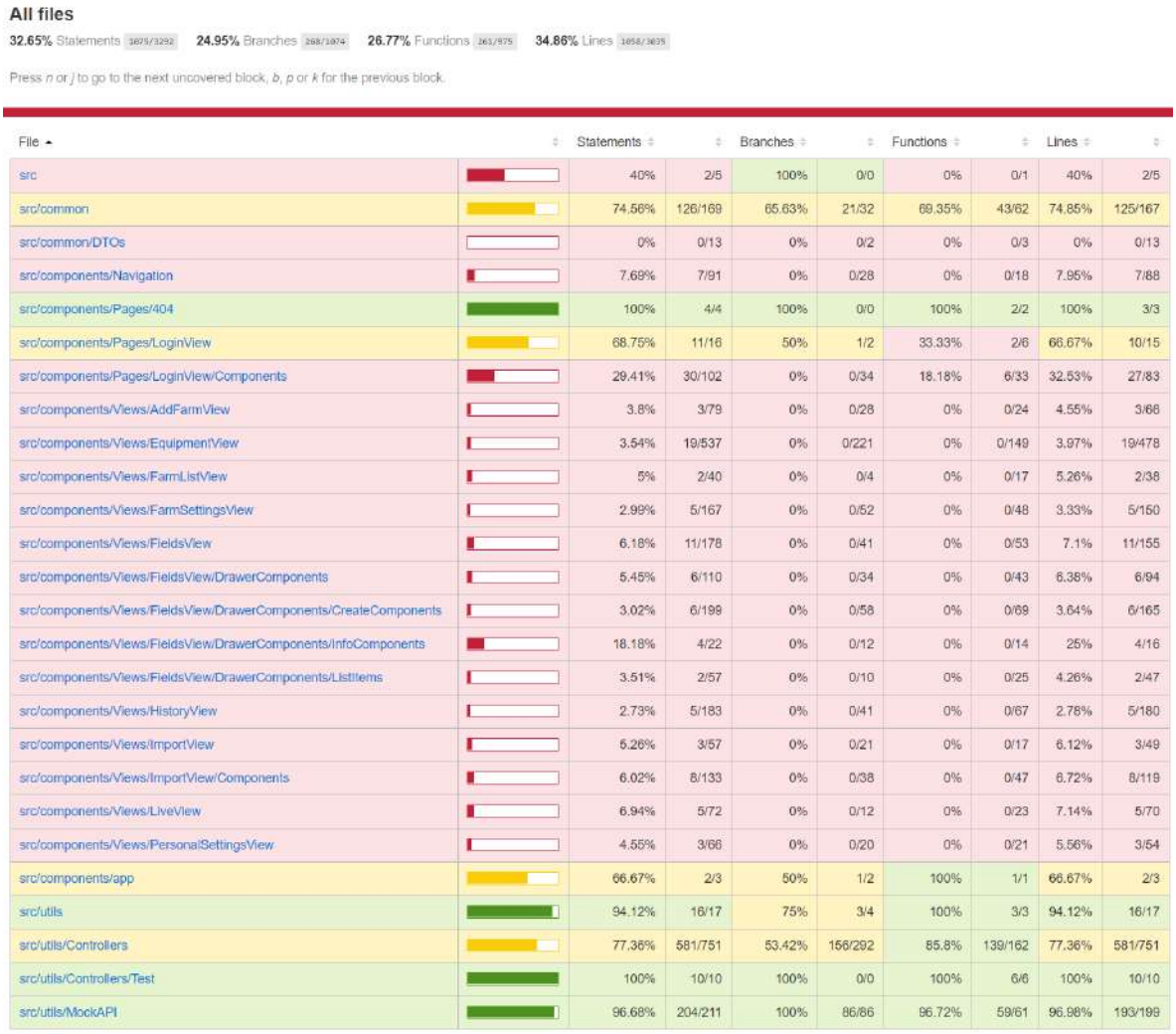
1. Open a terminal in the folder of the desired service to be tested.
2. Execute: `npm test`
3. The tests run and all results are shown in the terminal

5 Test Coverage

This chapter contains screenshots displaying the coverage reports for both the User Interface and the Data Linking Service. The reports follow from the running the aforementioned test procedure.

5.1 User Interface

As can be seen from the picture below, several files have a low coverage and thus are marked red. These files regard the React views of the application. Since Front-end testing with React is difficult, the decision is made to leave this out and make sure that this part was successfully tested during the Acceptance Test Plan (ATP).



All files

32.65% Statements 1075/3292 **24.95%** Branches 268/1074 **26.77%** Functions 261/975 **34.86%** Lines 1058/3035

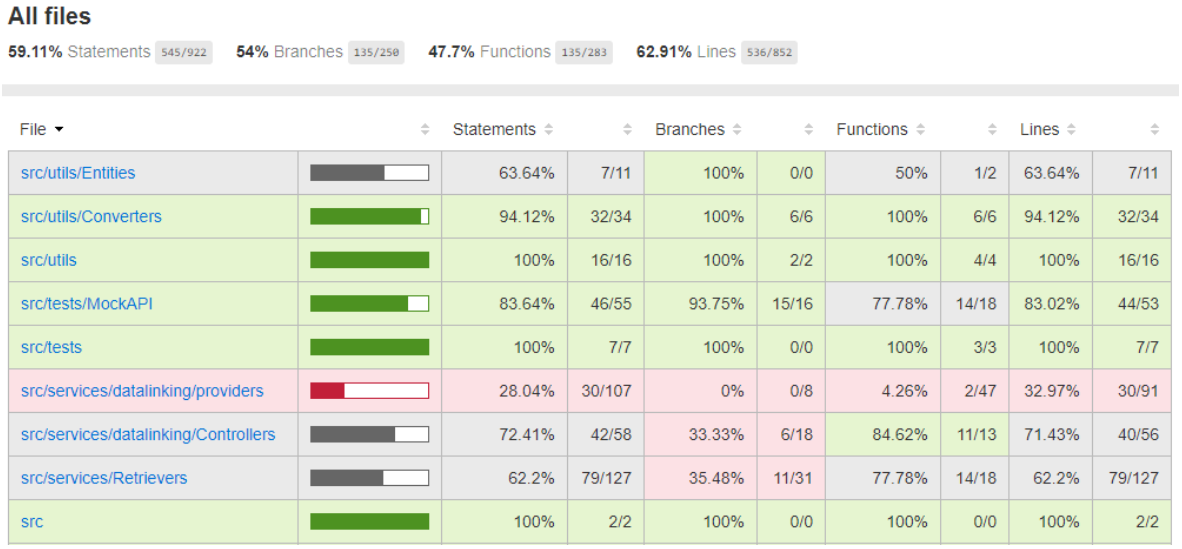
Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
src	40%	2/5	100%	0/0	0%	0/1	40%	2/5
src/common	74.56%	126/169	65.63%	21/32	69.35%	43/62	74.85%	125/167
src/common/DTOs	0%	0/13	0%	0/2	0%	0/3	0%	0/13
src/components/Navigation	7.69%	7/91	0%	0/28	0%	0/18	7.95%	7/88
src/components/Pages/404	100%	4/4	100%	0/0	100%	2/2	100%	3/3
src/components/Pages/LoginView	68.75%	11/16	50%	1/2	33.33%	2/6	66.67%	10/15
src/components/Pages/LoginView/Components	29.41%	30/102	0%	0/34	18.18%	6/33	32.53%	27/83
src/components/Views/AddFarmView	3.8%	3/79	0%	0/28	0%	0/24	4.55%	3/66
src/components/Views/EquipmentView	3.54%	19/537	0%	0/221	0%	0/149	3.97%	19/478
src/components/Views/FarmListView	5%	2/40	0%	0/4	0%	0/17	5.26%	2/38
src/components/Views/FarmSettingsView	2.99%	5/167	0%	0/52	0%	0/48	3.33%	5/150
src/components/Views/FieldsView	6.18%	11/178	0%	0/41	0%	0/53	7.1%	11/155
src/components/Views/FieldsView/DrawerComponents	5.45%	6/110	0%	0/34	0%	0/43	6.38%	6/94
src/components/Views/FieldsView/DrawerComponents/CreateComponents	3.02%	6/199	0%	0/58	0%	0/69	3.64%	6/165
src/components/Views/FieldsView/DrawerComponents/InfoComponents	18.18%	4/22	0%	0/12	0%	0/14	25%	4/16
src/components/Views/FieldsView/DrawerComponents/ListItems	3.51%	2/57	0%	0/10	0%	0/25	4.26%	2/47
src/components/Views/HistoryView	2.73%	5/183	0%	0/41	0%	0/67	2.78%	5/180
src/components/Views/ImportView	5.26%	3/57	0%	0/21	0%	0/17	6.12%	3/49
src/components/Views/ImportView/Components	6.02%	8/133	0%	0/38	0%	0/47	6.72%	8/119
src/components/Views/LiveView	6.94%	5/72	0%	0/12	0%	0/23	7.14%	5/70
src/components/Views/PersonalSettingsView	4.55%	3/66	0%	0/20	0%	0/21	5.56%	3/54
src/components/app	66.67%	2/3	50%	1/2	100%	1/1	66.67%	2/3
src/utils	94.12%	16/17	75%	3/4	100%	3/3	94.12%	16/17
src/utils/Controllers	77.36%	581/751	53.42%	156/292	85.8%	139/162	77.36%	581/751
src/utils/Controllers/Test	100%	10/10	100%	0/0	100%	6/6	100%	10/10
src/utils/MockAPI	96.68%	204/211	100%	86/86	96.72%	59/61	96.98%	193/199

Figure 1: The coverage report for the User Interface.

5.2 Data Linking Service

The test coverage report of the Data Linking Service can be found in the picture below. *src/services/datalinking/providers* shows a low coverage percentage, the files in this folder concern mostly calls to the database. Hence this was difficult to write unit tests for, so all databases calls were tested manually. This also influences the coverage percentage of *src/utils/Entities*, since several classes in this folder are only used in the calls to the database.



All files

59.11% Statements 545/922 **54%** Branches 135/250 **47.7%** Functions 135/283 **62.91%** Lines 536/852

File		Statements		Branches		Functions		Lines	
src/utils/Entities		63.64%	7/11	100%	0/0	50%	1/2	63.64%	7/11
src/utils/Converters		94.12%	32/34	100%	6/6	100%	6/6	94.12%	32/34
src/utils		100%	16/16	100%	2/2	100%	4/4	100%	16/16
src/tests/MockAPI		83.64%	46/55	93.75%	15/16	77.78%	14/18	83.02%	44/53
src/tests		100%	7/7	100%	0/0	100%	3/3	100%	7/7
src/services/datalinking/providers		28.04%	30/107	0%	0/8	4.26%	2/47	32.97%	30/91
src/services/datalinking/Controllers		72.41%	42/58	33.33%	6/18	84.62%	11/13	71.43%	40/56
src/services/Retrievers		62.2%	79/127	35.48%	11/31	77.78%	14/18	62.2%	79/127
src		100%	2/2	100%	0/0	100%	0/0	100%	2/2

Figure 2: The coverage report for the Data Linking Service.

6 Test Reports

In this section, the list of all tests are provided as well as whether or not they passed.

6.1 User interface

6.1.1 CropFieldController.test.ts

6.1.1.1 getCropFields()

- returns list of cropfields PASSED
- returns list of cropfields after refreshToken PASSED

6.1.1.2 getCropField()

- returns cropfield PASSED
- returns cropfield after refreshToken PASSED

6.1.1.3 addCropField()

- successfully add cropfield PASSED

- successfully add cropfield after refreshToken PASSED

6.1.1.4 updateCropField()

- successfully update cropfield PASSED
- successfully update cropfield after refreshToken PASSED

6.1.1.5 deleteCropField()

- successfully delete cropfield PASSED
- successfully delete cropfield after refreshToken PASSED

6.1.2 EquipmentController.test.ts

6.1.2.1 getEquipments()

- returns list of equipments PASSED
- returns list of equipments after refreshToken PASSED

6.1.2.2 getEqupiment()

- returns equipment PASSED
- returns equipment after refreshToken PASSED

6.1.2.3 addEquipment()

- successfully add equipment PASSED
- successfully add equipment after refreshToken PASSED

6.1.2.4 updateEquipment()

- successfully update equipment PASSED
- successfully update equipment after refreshToken PASSED

6.1.2.5 deleteEquipment()

- successfully delete equipment PASSED
- successfully delete equipment after refreshToken PASSED

6.1.3 EquipmentInfoController.test.ts

6.1.3.1 getEquipmentInfo()

- returns equipment information PASSED
- returns equipment information after refreshToken PASSED

6.1.3.2 addEquipmentInfo()

- successfully add equipment information PASSED
- successfully add equipment information after refreshToken PASSED

6.1.3.3 updateEquipmentInfo()

- successfully update equipment information PASSED
- successfully update equipment information after refreshToken PASSED

6.1.3.4 deleteEquipmentInfo()

- successfully delete equipment information PASSED
- successfully delete equipment information after refreshToken PASSED

6.1.4 EquipmentModelController.test.ts

6.1.4.1 getEquipmentModels()

- returns list of equipment models PASSED
- returns list of equipment models after refreshToken PASSED

6.1.4.2 getEquipmentModel()

- returns equipment model PASSED
- returns equipment model after refreshToken PASSED

6.1.4.3 addEquipmentModel()

- successfully add equipment model PASSED
- successfully add equipment model after refreshToken PASSED

6.1.5 FarmController.test.ts

6.1.5.1 getFarms()

- returns list of farms PASSED
- returns list of farms after refreshToken PASSED

6.1.5.2 addFarm()

- successfully add farm PASSED
- successfully add farm after refreshToken PASSED

6.1.5.3 getFarm()

- successfully returns farm PASSED
- successfully returns farm after refreshToken PASSED

6.1.5.4 updateFarm()

- succeful update farm PASSED
- return update farm after refreshToken PASSED
- farm does not exist is handled PASSED

6.1.5.5 deleteFarm()

- successfully delete farm PASSED
- successfully delete farm after refreshToken PASSED
- farm does not exist is handled PASSED

6.1.6 FieldController.test.ts

6.1.6.1 getFarmFields()

- returns list of fields PASSED
- returns list of fields after refreshToken PASSED

6.1.6.2 getFarmField()

- returns field PASSED
- returns field after refreshToken PASSED

6.1.6.3 addFarmField()

- successfully add field PASSED
- successfully add field after refreshToken PASSED

6.1.6.4 updateFarmField()

- successfully update field PASSED
- successfully update field after refreshToken PASSED

6.1.6.5 deleteFarmField()

- successfully delete field PASSED
- successfully delete field after refreshToken PASSED

6.1.7 MappingController.test.ts

6.1.7.1 getDataMaps()

- returns list of datamaps PASSED
- returns list of datamaps after refreshToken PASSED

6.1.7.2 addDataMap()

- successfully add datamap PASSED
- successfully add datamap after refreshToken PASSED

6.1.8 RoleController.test.ts

6.1.8.1 getUserRoleOnFarm()

- returns role PASSED
- returns role after refreshToken PASSED

6.1.8.2 changeUserRoleOnFarm()

- successfully update role PASSED
- successfully update role after refreshToken PASSED

6.1.8.3 createUserRoleOnFarm()

- successfully add role PASSED
- successfully add role after refreshToken PASSED

6.1.8.4 getAllUsersOnFarm()

- returns users with roles PASSED
- returns users with roles after refreshToken PASSED

6.1.8.5 getUserRoles()

- returns roles PASSED
- returns roles after refreshToken PASSED

6.1.9 TypesController.test.ts

6.1.9.1 getCountries()

- returns list of countries PASSED
- returns list of countries after refreshToken PASSED

6.1.9.2 getCropTypes()

- returns list of crop types PASSED
- returns list of crop types after refreshToken PASSED

6.1.9.3 getSoilTypes()

- returns list of soil types PASSED
- returns list of soil rypes after refreshToken PASSED

6.1.10 UserController.test.ts

6.1.10.1 login()

- handle invalid username or password PASSED
- handle successful login PASSED
- userId stored in cookies PASSED
- accessToken stored in cookies PASSED
- refreshToken stored in cookies PASSED
- loggedIn() = true PASSED

6.1.10.2 logout()

- accessToken deleted from cookies PASSED
- refreshToken deleted from cookies PASSED
- UserId deleted from cookies PASSED
- loggedIn() = false PASSED

6.1.10.3 register()

- handle email already exists PASSED
- handle successful login PASSED
- userId stored in cookies PASSED
- accessToken stored in cookies PASSED
- refreshToken stored in cookies PASSED

6.1.10.4 addFavorite() and deleteFavorite()

- handle delete favorite farm does not exist PASSED
- successfully add favorite farm PASSED
- successfully delete favorite farm PASSED

6.1.10.5 getUser()

- handle user not found error PASSED
- return user object PASSED
- return user object after refreshToken PASSED

6.1.10.6 updateUser()

- handle update user error PASSED
- successfully update user PASSED
- successfully update user after refreshToken PASSED

6.1.10.7 deleteUser()

- handle delete user error PASSED
- successfully delete user PASSED
- successfully delete user after refreshToken PASSED

6.1.11 WolkyTolkyEqController.test.ts

6.1.11.1 getWolkyTolkyEqs()

- returns list of WolkyTolky equipments PASSED
- returns list of WolkyTolky equipments after refreshToken PASSED

6.1.11.2 getWolkyTolkyEq()

- returns WolkyTolky equipment PASSED
- returns WolkyTolky equipment after refreshToken PASSED

6.1.11.3 addWolkyTolkyEq()

- successfully add WolkyTolky equipment PASSED
- successfully add WolkyTolky equipment after refreshToken PASSED

6.1.11.4 updateWolkyTolkyEq()

- successfully update WolkyTolky equipment PASSED
- successfully update WolkyTolky equipment after refreshToken PASSED

6.1.11.5 deleteWolkyTolkyEquipment()

- successfully delete WolkyTolky equipment cropfield PASSED
- successfully delete WolkyTolky equipment after refreshToken PASSED

6.2 Data linking service

6.2.1 MappingController.test.ts

6.2.1.1 getDataMapIdByEquipmentModelId()

- returns a datamap id of an equipment model PASSED

6.2.1.2 getDataColumnsByEquipmentModelId()

- get all the data columns of an equipment model PASSED

6.2.1.3 getDataMaps()

- get the json string containing a list of datamaps PASSED

6.2.1.4 getEquipmentModelId()

- get the equipment model id of an equipment PASSED

6.2.2 UserController.test.ts

6.2.2.1 getAccessToken

- retrieve access token PASSED

6.2.3 EnumConverter.test.ts

6.2.3.1 stringToAccessibilityType()

- string: public PASSED
- string: private PASSED
- string: restricted PASSED

6.2.3.2 stringToAccessibilityType()

- enumValue: public PASSED
- enumValue: private PASSED
- enumValue: restricted PASSED

6.2.4 WolkyTolkyDataConverter.test.ts

6.2.4.1 transformLiveData()

- transform the data PASSED

6.2.4.2 transformLocationData()

- transform the location data PASSED

6.2.5 DataRetriever.test.ts

6.2.5.1 saveData()

- save the retrieved observation data to the database PASSED

6.2.6 WolkyTolkyDataRetriever.test.ts

6.2.6.1 getData()

- get data from a WolkyTolky station PASSED
- get data from a WolkyTolky station after a certain date PASSED

6.2.6.2 getLatestDataDate()

- get the latest date of the observation data from WolkyTolky PASSED

6.2.6.3 getLastSaved()

- tests whether a date is retrieved when the retriever last ran PASSED

6.2.6.4 run()

- test the run function of the WolkyTolky PASSED

6.2.6.5 saveData()

- save the retrieved observation data to the database PASSED

6.2.7 WolkyTolkyLiveDataRetriever.test.ts

6.2.7.1 getLatestData()

- get the latest observation data from the WolkyTolky sensor PASSED