POLITECNICO
MILANO 1863

Computer Science and Engineering

Software engineering 2 Project

DREAMS

ITD

Implementation and Testing Document

Versione 1.0 – 06/02/2022

Pistone Santi Pier – 996419

Zouzoua Axel Israel Ble – 968931

Link to the software: https://github.com/santipistone/PistoneBle3

Sommario

Introduction

This document contains all the details about the implementation and the testing of DREAMS.
Not all the parts of the have been taken into consideration; indeed the aim of this document is to show what could be a possible implementation of the software and the corresponding achieved results.
For the scope, the attention is put on the Policy Maker.

The corresponding RASD and DD documents can be found on the same Git hub page.

Scope

This work is focused on how the software should present as an application for tablet devices but for a lack of medium it is show from the web point of view.

What is discussed in this document is referred to the parts of the system regarding the Policy Maker.
The developed parts are:

- PolicyMaker Application that is the Client services for the policy maker
- parts of the Server dedicated to the Policy Maker and their interfaces, that is:
    - Autentication Manager (PolicyAuthentication Management)
    - Supervision Services Manager: DirectConnection Manager (PolicyChatGrourGesture interface)
    - Production Manager : Analytics Manager (Analytics Management interface)
    - Suggestions & Help Manager : HelpTicket Manager (PolicyTicket Management interface)
- Data base: only a “snippet” to favor the simulation and to give an idea of a plausible expansion.

Implemented Requirements

The requirements implemented in this work according to the RASD are the following:

• [R3]: A policy maker that uses the system should be logged

• [R4]: A Policy Maker obtains the DREAMS special home page after he has logged-in

• [R17]: The system allows a policy maker to visualize all the production analytics in output from the classification algorithm

• [R22]: The system allows only a policy maker to establish a Direct Connection; A Direct Connection can be established only between two farmers at a time

• [R24]: The system allows a policy maker to visualize all the generic production data of each of the farmers under his supervision

• [R25]: A policy maker can comment on farmers performances grades calculated by the algorithm

• [R26]: When a group chat is created, the system automatically names it with the pair of farmers names

Developement Frameworks

The frameworks used in the development of this work are specified in the corresponding DD document.
It is important to consider that it is intended to be just a simulation, therefore the specifications regarding the frameworks about the Mobile Application mentioned in the DD document are not respected.

Adopted Programming Languages

The programming languages used are:

- Htlm, css & js to manage the Client part
- Blade php to code the Server
- And MongodB which is NoSQL type database

Those languages are used for the sake of good order, simplicity and clarity.
Further expansions and development of the system could integrate more sophisticated programming languages.

Middlewares

Using a web-based architecture of type Client – Server and Laravel MVC, the middlewares are already integrated in the system.

APIs

Laravel includes Eloquent, an object-relational mapper (ORM) that makes it enjoyable to interact with your database. When using Eloquent, each database table has a corresponding "Model" that is used to interact with that table. In addition to retrieving records from the database table, Eloquent models allow you to insert, update, and delete records from the table as well.

The only API used is the original Laravel API to allow the communication between Laravel and MongoDB. It is also provided with an external library that extends the original Laravel classes, so it uses exactly the same methods: it is a package that adds functionalities to the Eloquent model and Query builder for MongoDB.

Information about this library can be found in the references section.

Code Structure

The structure of the code is composed of three principal elements each dealing with the Client GUI, the Server and the Database, more precisely:

The **Client** that is Policy Maker Application is composed of four html pages:

- login.blade.php: supports the graphic interface for the client's authentication
- policy.blade.php: supports the graphic interface for the home page of the application
- analytics.blade.php: supports the graphic interface to allow the user to visualize the analytics results

- ticket.blade.php: supports the graphic interface to allow the policy maker in managing the tickets he has received from the farmers
- direct.blade.php: supports the graphic interface for the chat service;

A css page “dreams.css“ that cures the styling for all the html section.

A js script “dreams.js”, instead, cares about making the tools on the GUI interactive, and manages the interaction -switching- between the different sections of the application.

Server: the structure is based on two controllers: Db Controller and Page Controller.
Db controller oversees managing the queries that is the interactions between the database and the server itself. This controller uses four different models:

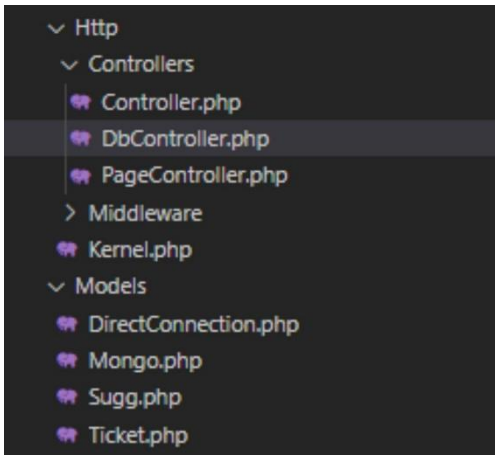
- mongo: deals with interactions between the document called “user”
- ticket: deals with interactions between the document called “ticket”
- suggestions: deals with interactions between the document called “suggestions”
- direct connection: deals with interactions between the document called “direct connection”

Page Controller manages the results returned to the views

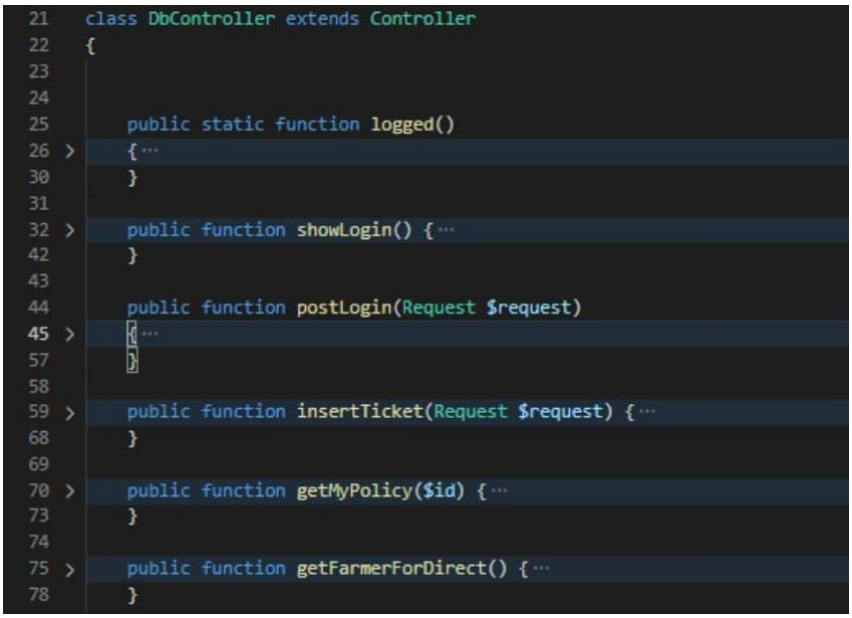
Each of these models interfaces directly with the database.
the Controllers and the Models work together to manage the views of the software.

Data base: MongoDB

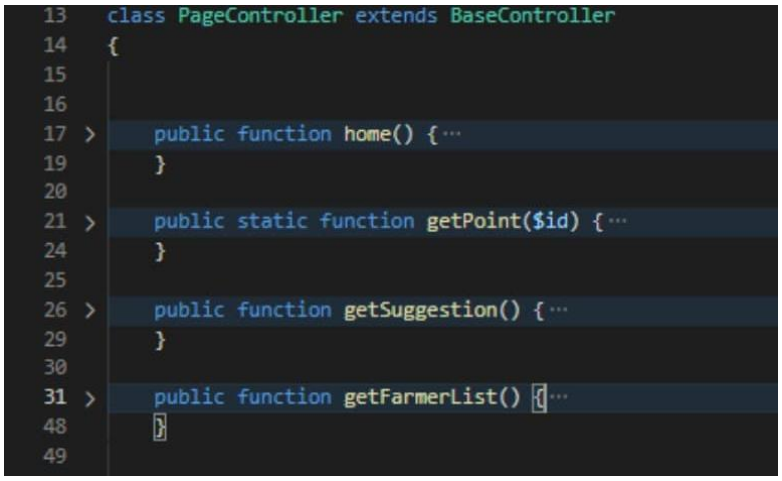
Controllers and Models



Db Controller



Page Controller



```
class PageController extends BaseController
{


    public function home() {···
    }

    public static function getPoint($id) {···
    }

    public function getSuggestion() {···
    }

    public function getFarmerList() {···
    }

```

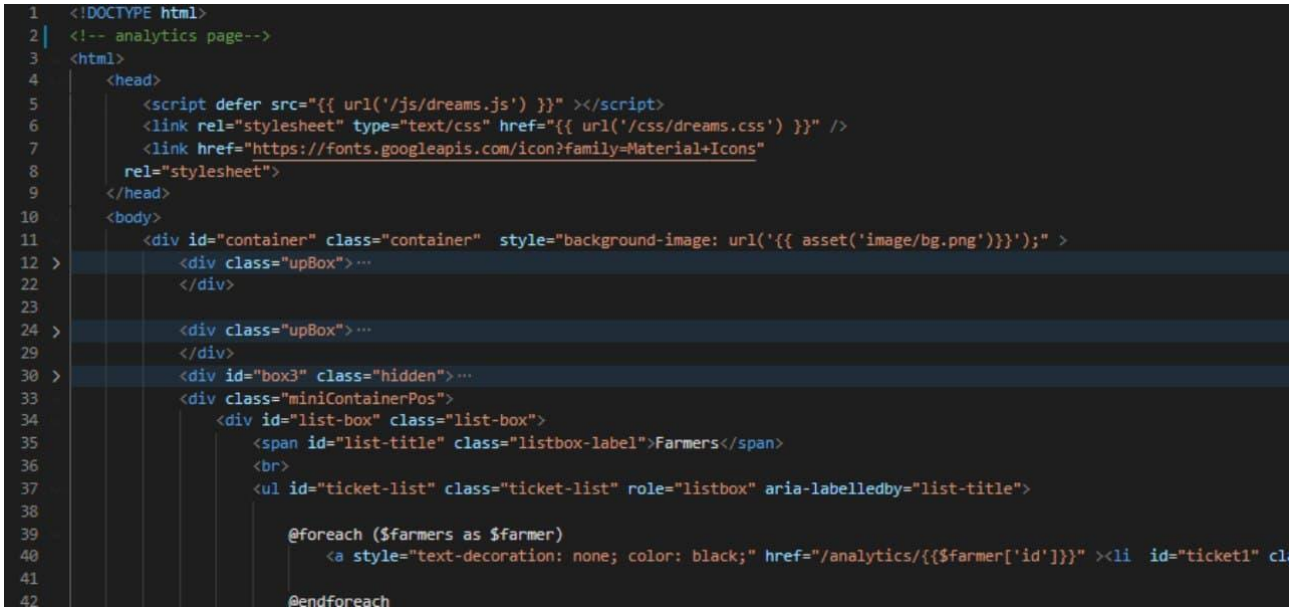
login.blade.php

```
<!DOCTYPE html>
<!-- login page -->
<html>
    <head>···
    </head>
    <body>
        <div id="container" class="container" style="background-image: url('{{ asset('image/bg.png')}}');">
        <span id="logo" class="logo-login">
                DREAMS
                <button id="logo-button" type="button" class="logo-button-login">
                    <span class="material-icons">grass</span>
                </button>
            </span>
            <div class="login-box">
                <form class="form-box" action="{{url('login')}}" method="POST">
                    @csrf
                    <label for="usrname">Username:</label>
                    <input required="" type="text" name="email" class="password" required>
                    <label for="psw">Password:</label>
                    <input id="psw" type="password" name="passw" class="password" required>

                    <input type="submit" value="Submit" class="password">
                </form>
                <button id="eye-button" class="password-eye">
                        <i class="far fa-eye" id="togglePassword";></i>
                    </button>
              </div>
```

policy.blade.php

```
<!DOCTYPE html>
<!-- policy maker home page -->
<html>
    <head>···
    </head>
    <body>
        <div id="container" class="container"  style="background-image: url('{{ asset('image/bg.png')}}');" >
            <div class="upBox">
                <span id="logo" onclick="location.href='home';" class="logo">
                    DREAMS
                    <button id="logo-button" type="button" class="logo-button">
                        <span class="material-icons">grass</span>
                    </button>
                </span>

            </div>
            <div class="upBox">···
            </div>
            <div id="box3" class="hidden">···
            <div class="miniContainerPos">
                <div style="font-weight: bold;" id="AnalyticsBlock" class="analyticsContainer" onclick="location.href='analytics';">
                    Analytics
                </div>
                <div style="font-weight: bold;" id="DirectBlock" class="sections" onclick="location.href='direct';">
                    Direct Connection
                </div>
```

analytics.blade.php



```
<!DOCTYPE html>
<!-- analytics page-->
<html>
    <head>
        <script defer src="{{ url('/js/dreams.js') }}" ></script>
        <link rel="stylesheet" type="text/css" href="{{ url('/css/dreams.css') }}" />
        <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
      rel="stylesheet">
    </head>
    <body>
        <div id="container" class="container"  style="background-image: url('{{ asset('image/bg.png')}}');" >
            <div class="upBox">···
            </div>

            <div class="upBox">···
            </div>
            <div id="box3" class="hidden">···
            <div class="miniContainerPos">
                <div id="list-box" class="list-box">
                    <span id="list-title" class="listbox-label">Farmers</span>
                    <br>
                    <ul id="ticket-list" class="ticket-list" role="listbox" aria-labelledby="list-title">

                        @foreach ($farmers as $farmer)
                            <a style="text-decoration: none; color: black;" href="/analytics/{{$farmer['id']}}" ><li  id="ticket1" cl

                        @endforeach
```

ticket.blade.php

```
<!-- ticket page -->
<html>
    <head>
    <script defer src="{{ url('/js/dreams.js') }}" ></script>
    <link rel="stylesheet" type="text/css" href="{{ url('/css/dreams.css') }}" />···
    </head>
    @if (App\Http\Controllers\DbController::logged() == 1)
    <body>
        <div id="container" class="container" style="background-image: url('{{ asset('image/bg.png')}}');">
            <div class="upBox">
                <span id="logo" class="logo">···
                </span>

            </div>
            <div class="upBox">···
            </div>
            <div id="box3" class="hidden">···
            <div class="miniContainerPos">
                <div id="list-box" class="list-box">
                    <span id="list-title" class="listbox-label">
                        Tickets
                    </span>
                    <br>
                    <ul id="ticket-list" class="ticket-list" role="listbox" aria-labelledby="list-title">
                        @foreach ($tickets as $ticket)
                        @if ($ticket['open'] == 1)
```

direct.blade.php

```
<!DOCTYPE html>
<!--direct connection page-->
<html>
    <head>
    <script defer src="{{ url('/js/dreams.js') }}" ></script>
    <link rel="stylesheet" type="text/css" href="{{ url('/css/dreams.css') }}" />
        <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
      rel="stylesheet">
    </head>
    @if (App\Http\Controllers\DbController::logged() == 1)
    <body>
        <div id="container" class="container" style="background-image: url('{{ asset('image/bg.png')}}');">
            <div class="upBox">···
            </div>
            <div class="upBox">···
            </div>
            <div id="box3" class="hidden">···
            <div class="miniContainerPos">
                <div id="list-box" class="list-box">
                    <span id="list-title" class="listbox-label" style="margin-left: 195px;">
                        Direct Connection
                    </span>
                    <br>
                    <ul id="ticket-list" class="ticket-list" role="listbox" aria-labelledby="list-title">
                        @foreach ($dm as $mex)
                        <a style="text-decoration: none; color: black;" href="/direct/{{(string)$mex['id']}}" ><li  id="ticket1" clas
```

dreams.js

```
2  > window.onload=function(){···
25   }
26
27   function openLogin(event) {
28     const l1 = document.querySelector("#box3");
29     if (l1.classList.contains("hidden")) {
30         l1.classList.remove("hidden");
31     } else {
32         l1.classList.add("hidden");
33     }
34   }
35
36
37
38   const fk = document.querySelector("#account");
39   fk.addEventListener("click", openLogin);
40
41
```

Testing Information

The testing has been done from different points of view:

- **Acceptance testing:** The whole system works as intended.
- **Integration testing:** it has been done as indicated in the DD document.
- **Unit testing:** each component has been tested before being integrated with the others.
- **Functional testing:** since just part of the system regarding only one actor, the policy maker, it was not really possible to simulate a real business scenario. But to ensure that the considered functional requirements had been implemented correctly, a fixed simulation has been implemented in the database.
- **Regression testing:** it is done by verifying the functioning of the system step by step; yet as a new functionality, tool or component is added, the whole system responds correctly.
- **Usability testing:** the user application component is really simple. All the tasks a user could have the necessity to complete considered in this work can be carried out simply and correctly, with a good response from the system.
  In this case, especially, all the tasks leading to the achievement of the goals G1 and G7 (more details in the RASD document) are well supported by the system.

Other types of testing such as Performance testing, Crowd Testing and Stress testing have not been made for they are out of the results wanted to achieve in this work.

Installation Instructions

To have the possibility to interact with the work, it is required to execute the instructions correctly as follows:

Installation:

- Download + install php: in php install php composer
- Download and install php composer: No developer; browse php folder and choose php application; add path
- Downlaod + installation Laravel project folder
- Download MongoDB Compass
- Create MongoDB server via atlas
- Connect to MongoDB Server (Atlas) via laravel:open file config/database.php, and modify here:

```
'mongodb' => [
                'driver' => 'mongodb',
                'dsn' => env('DB_URI', 'mongodb+srv://<username>:<password>@<database_url>'),
                'database' => '<database_name>',
        ]
```

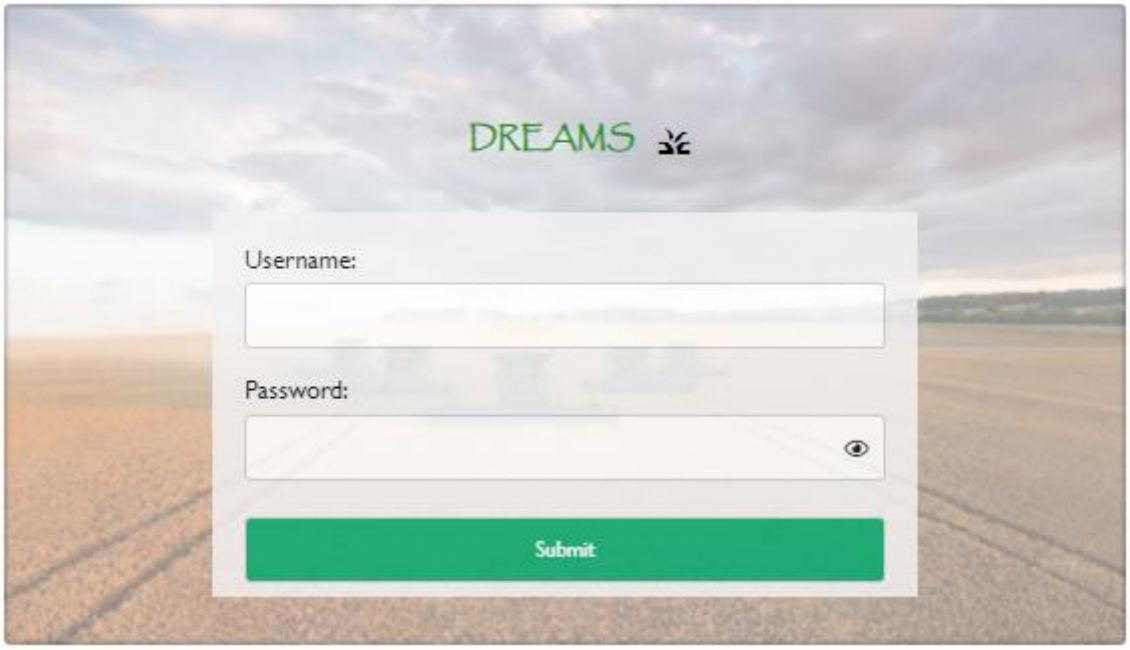
- Upload MongoDB document in your own MongoDB server.
- Download + install certificate SSL to connect via Laravel to MongoDB server.
- Start laravel server via command:

```
php artisan serve.
```

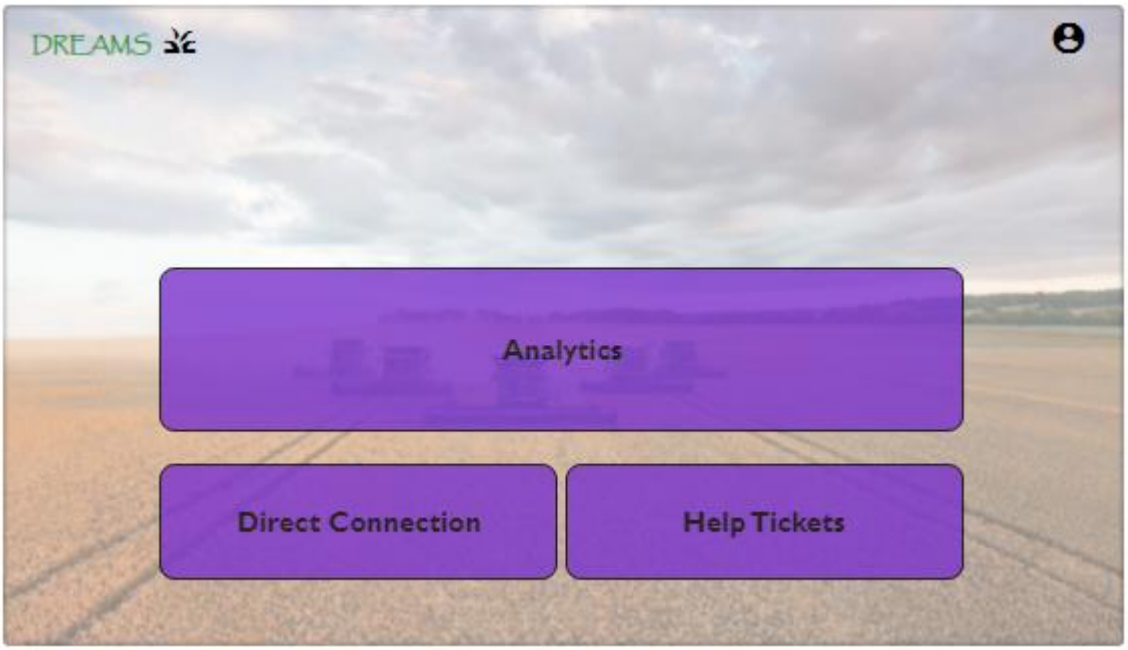
Then to interact with the application connect on localhost:8000 from the web browser.

Application picture

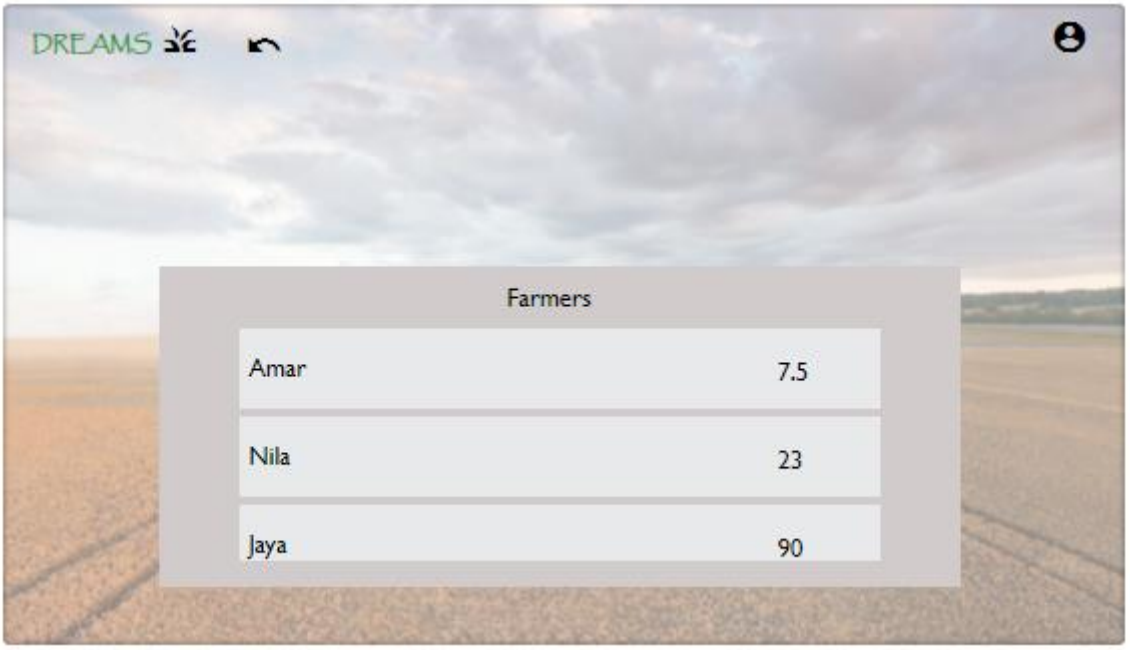
Login



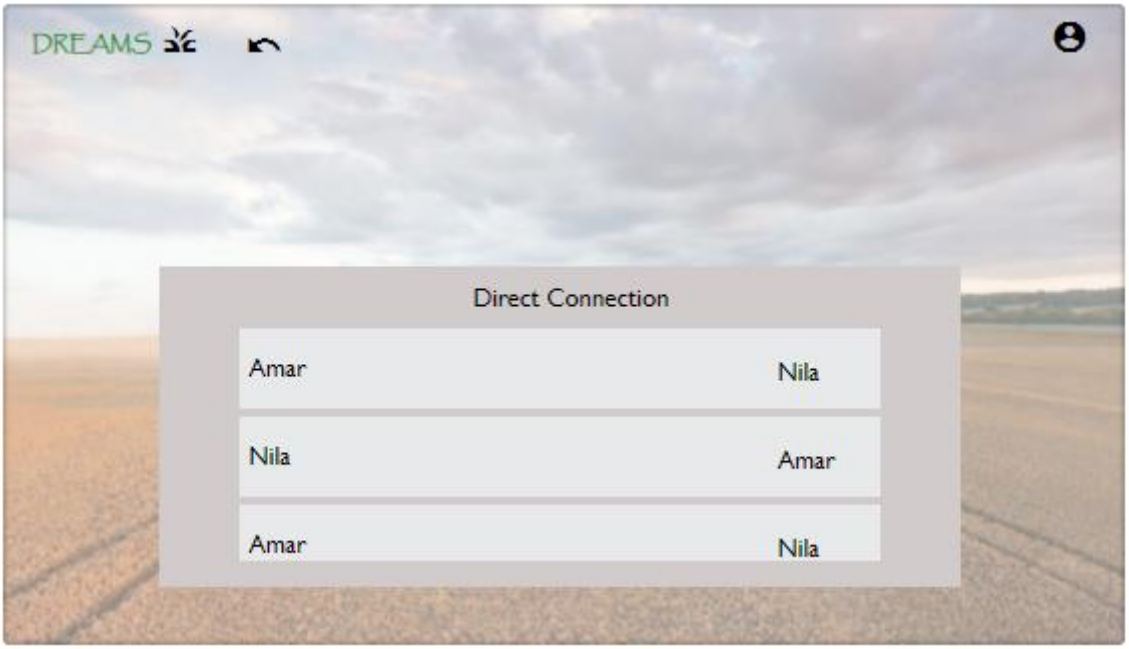
Home



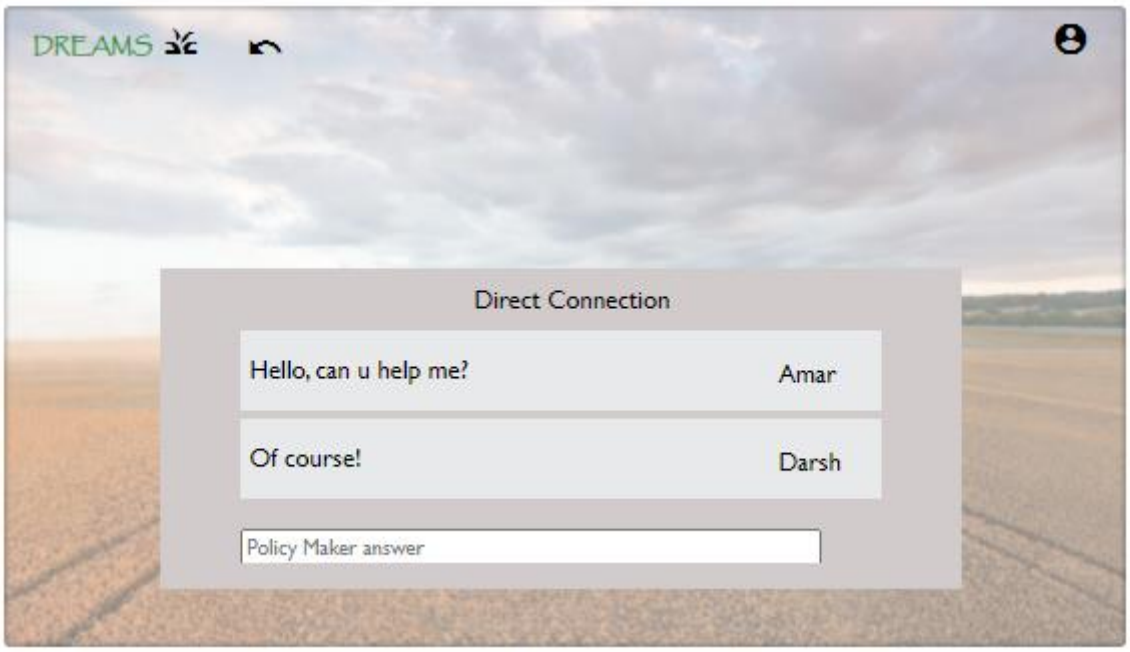
Analytics



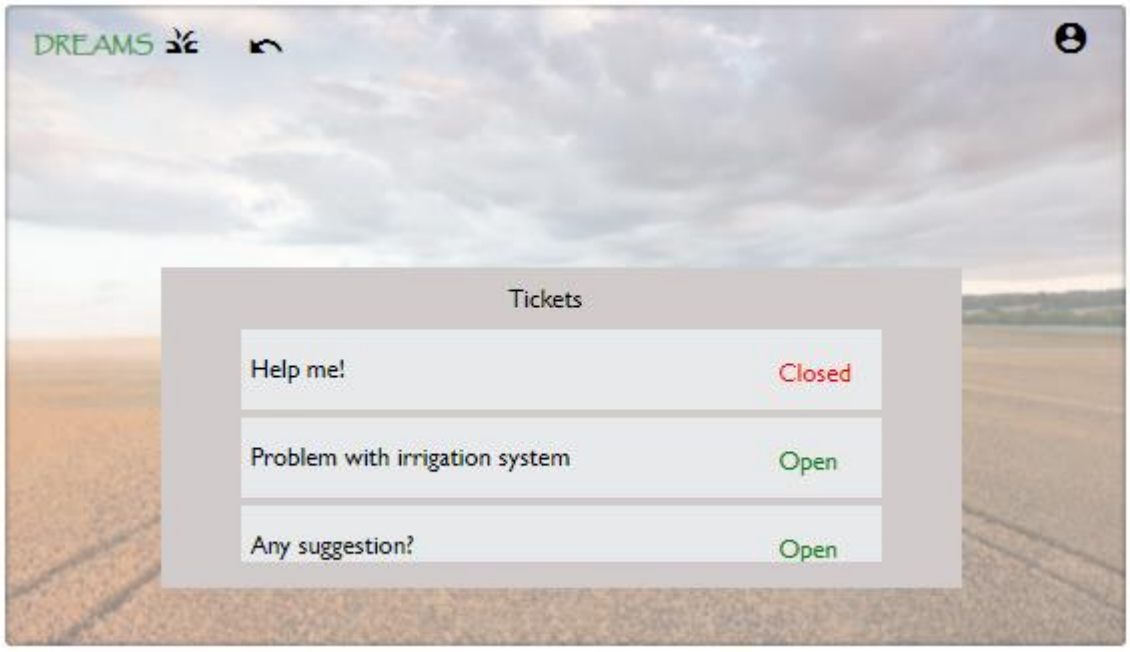
Direct Connection



Opened Direct Connection



Ticket List



Effort Spent

Student 1

Topic	Hours
General Reasoning	**8:00h**
GUI Coding	**10:00h**
Server Coding	**3:00h**
Database	**1:30h**
Testing	**5:20h**
Document	**3:00h**

Student 2

Topic	Hours
General Reasoning	**8:00h**
GUI Coding	**3:00h**
Server Coding	**10:00h**
Database	**8:30h**
Testing	**5:20h**
Document	**3:00h**

References

https://github.com/jenssegers/laravel-mongodb