MindTrails Documentation

Release 2.0.1

Dan Funk, Sam Portnow, Diheng Zhang

Feb 07, 2022

Contents

Back to the main manu

MindTrails is a library for constructing online Cognitive Behavioral Therapy studies with tools to address major concerns for such studies - such as reducing attrition and secure data handling.

MindTrails provides a core library, and a default implementation for creating new Studies. It provides the following basic tools:

1. Study / Task Management

MindTrails provides an open framework for organizing html web forms and Javascript applications (such as the Project Implicit Player) as *Tasks* within a series of *sessions*. Each session is completed in a single sitting.

Participants are provided with a series of attractive (but highly customizable) views of their progress. And are moved through the tasks rapidly to create a streamlined and pleasant progression.

We chose to use standard HTTP POST as the basis for handling form submissions, as this provided the most open model for extension and expansion. If the tool you want to use for data collection can POST that back to a web server, it should integrate well into this application.

2. Email Reminder System

The system can be configured to remind participants to return to complete later sessions on a regular schedule. Participants can easily opt out of these emails if they choose to do so.

3. Gift Card Support

Integrated with Tango - MindTrails can fully automate the awarding of Gift Cards to help encourage participants to complete sessions.

4. Secure data storage

MindTrails is currently in use at the University of Virginia where it collects Highly Sensitive Data. The system has a robust export system to allow collected information to be pulled from the main web server - creating a clean separation between identifying data (email addresses) and medical or other information.

A seperate application, the MindTrails Data can be used to pull all submitted data from the system on a tight interval (say every 5 minutes) onto a separate server behind a firewall. If the main web server is compromised no medical information will be available.

In addition, email alerts can be configured to notify administrators in cases where sensitive data might be left on a web accessible server for an extended period of time.

5. Simple Data Architecture

Data is collected in a relational database where the records are stored in a table(s)-per-form format that can be easy customized. If the data is not secure, working with a study's collected information is easy and straight forward. If the data is sensitive in nature, it can be exported in the same configurable format. Implementations have full control over the format of the json / excel. There is a companion project that can be used to extract the data to a private server and generate spreadsheets for later analysis - see secure data storage above.

6. A Secure, modern web application Our Security Model is build on the popular Spring Security Framework.We currently use a form based authentication (a web login form) that provides projects against CSRF Attacks, Session Fixation Protection, and Security header integration.

- User Authentication
- Participant Assignments
- Data Collection
- Data Reporting
- Session management

- A new bullet

It is built in Java using the Spring Framework and Spring Boot. There is exceptional documentation on these technologies here: http://spring.io/guides#gs

CHAPTER 1

Getting Started

1.1 Requirements

You must have the following applications installed in order to build and run the server.

- Java 7 JDK - (http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html)
- Node (Server side Javascript - for building the PIPlayer, see http://howtonode.org)
- Bower (Javascript package management tool - just run "npm install bower -global")
- Mysql (Relational Database - http://dev.mysql.com/doc/refman/5.1/en/installing.html)

1.2 Intelij Development

- You need to enable the annotation preprocessor for lombok annotations to work correctly. https://www.jetbrains.com/idea/help/configuring-annotation-processing.html

1.3 Database Setup

Install MySQL, and execute the following commands to establish a user account. You can use a different password if you change the datasource.password setting in src/main/resources/application.properties

```
> CREATE DATABASE pi CHARACTER SET utf8 COLLATE utf8_general_ci;
> CREATE USER 'pi_user'@'localhost' IDENTIFIED BY 'pi_password';
> GRANT ALL PRIVILEGES ON pi.* TO 'pi_user'@'%' IDENTIFIED BY 'pi_password' WITH␣
↪GRANT OPTION;
```

If you are running the tests, that is configured to use a seperate database

```
> CREATE DATABASE pi_test CHARACTER SET utf8 COLLATE utf8_general_ci;
> GRANT ALL PRIVILEGES ON pi_test.* TO 'pi_user'@'%' IDENTIFIED BY 'pi_password' WITH␣
↪GRANT OPTION;
```

The templeton project requires its own database

```
> CREATE DATABASE templeton CHARACTER SET utf8 COLLATE utf8_general_ci;
> GRANT ALL PRIVILEGES ON templeton.* TO 'pi_user'@'%' IDENTIFIED BY 'pi_password'␣
↪WITH GRANT OPTION;
```

The mobile project requires its own database as well

```
> CREATE DATABASE mobile CHARACTER SET utf8 COLLATE utf8_general_ci;
> GRANT ALL PRIVILEGES ON mobile.* TO 'pi_user'@'%' IDENTIFIED BY 'pi_password' WITH␣
↪GRANT OPTION;
```

1.4 Installing Javascript Dependencies

Javascript dependencies, including the PIPlayer are installed using Bower, just run `bower install`

```
> cd core
> bower install
> cd ..
```

Because of the way the PIPlayer script is currently designed, you will need to install the PIPlayer dependencies manually, you can do this by:

```
> cd core/src/main/resources/static/bower/PIPlayer
> bower install
```

Please Note: if you run into problems with PI Player scripts not executing you might try editing the file (This is only required for r34, Templeton has a better configuration)

```
> vim core/src/main/resources/static/bower/PIPlayer/dist/js/config.js
> Set the baseUrl:'../bower/PIPlayer/dist/js',
```

1.5 Running

You can start up the webserver in development mode (meaning hot swappable / auto reloading) with: $ ./gradlew clean bootrun (on windows this is ./gradlew.bat clean bootrun)

You can now visit the website at : http://localhost:9000

1.6 Deploying

You can generate a WAR file suitable for deployment in a web server with $ ./gradlew war Then you will find the war file in ./build/lib/piServer-0.1.0.war

If you need to modify the default configuration for production (and you most likely will) then you should edit the file WEB-INF/classes/application.properties that exists within the WAR file (just edit the file in place) There are three major areas to configure for a production installation:

1. You will need to create/update/modify permissions on a database.
2. You will need an account with Tango (a service to automate the giving of gift cards.)

CHAPTER 2

A word on Sensitive Data

It is critical that we keep the link between a participants personal medical history and their identifiable information (such as email) separate. It is also important that we be able to re-connect this information in the event that we need to contact the particpant because we identify a pattern in the data collected that we much ethically notify the participant about. To keep participant data anonymous (but ultimately linkable) there is the ability to export and then expunge this information from the main web server. A series of REST endpoints exist that allow for listing all data that is available, downloading that data, and then removing the data from the server. These end points include:

GET *[SERVER]/api/export*: Returns a list of all questionnaires, including the number of records, and if the records can and should be deleted after export. For example: *curl -u admin@email.com:passwd localhost:9000/api/export* would return:

```
[
   {
      "name" : "SUDS",
      "deleteable" : true,
      "size" : 9
   },
   {
      "name" : "DASS21_DS",
      "deleteable" : true,
      "size" : 16
   },
   {
      "name" : "Demographic",
      "deleteable" : true,
      "size" : 4
   }
   ...
]
```

GET *SERVER/api/export/NAME*: Returns all the data available on the server at that moment. For example: *curl -u admin@email.com:passwd localhost:9000/api/export/ImageryPrime* would return:

```
[
   {
      "id" : 1,
      "vivid" : 0,
      "situation" : "Waiting at doctors office",
      "prime" : "prime",
      "think_feel" : null,
      "date" : 1441908506000,
      "session" : "PRE",
      "participantDAO" : 5
   },
   {
      "id" : 2,
      "vivid" : 0,
   ...}
   ...
 ]
```

DELETE *SERVER/api/export/NAME/ID*: Removes a record from the Database. For example: *curl -u admin@email.com:passwd -X DELETE localhost:9000/api/export/ImageryPrime/1* would remove the item above. This is secure delete, where the id linking the record to a participant is first overwritten, then deleted.

2.1 Encryption

You can enable encryption of the link between questionnaire data and the participant. When you do so, the system will use a public key to encrypt the participant id when recording the questionnaire.

2.2 Generating a key

(Taken from: http://stackoverflow.com/questions/11410770/load-rsa-public-key-from-file)

1. Generate a 2048-bit RSA private key

```
$ openssl genrsa -out private_key.pem 2048
```

1. Convert private Key to PKCS#8 format (so Java can read it)

```
$ openssl pkcs8 -topk8 -inform PEM -outform DER -in private_key.pem \ -out private_
↪key.der -nocrypt
```

1. Output public key portion in DER format (so Java can read it)

```
$ openssl rsa -in private_key.pem -pubout -outform DER -out public_key.der
```

1. Place the public key in the resources directory of the WAR file, and place the private key on the server running the MTData script.

2.3 Manually Decrypting a link from the command line

if you place the encrypted string in a file, you can decrypt it with

```
base64 -d encrypted.txt  | openssl rsautl -decrypt -inkey private_key.pem
```

If you are copying the key from a file, you can deocde it directly with

```
echo myEncryptedString | base64 -d | openssl rsautl -decrypt -inkey private_key.pem
```

2.4 Testing

```
$ ./gradlew test
```

Test results can be found in . . . PIServer/build/reports/tests/index.html

CHAPTER 3

MTData - package for data handling

We also wrote a python command line tool designed to handle sensitive data for MindTrails library. You can install it on your back end server or laptop, configure the server.config files and keys for decrypting. It comes with tools that take care of most of the data issue.

You can find out more here: MTData

CHAPTER 4

Security Overview

Our Security Model is build on the popular Spring Security Framework. Specifically version 3.2.3 We currently use a form based authentication (a web login form) that provides the following basic projections and features:

- Every URL in the site requires authentication.
- CSRF attach prevention
- Session Fixation Protection
- Security header integration
  - HTTP Strict Transport Security for secure requests
  - X-Content-Type-Options integration
  - Cache Control (can be overridden later by your application to allow caching of your static resources)
  - X-XSS-Protection integration
  - X-Frame-Options integration to help prevent Clickjacking

In production, we will secure the site using SSL encryption; all page requests occur over an HTTPS connection.

CHAPTER 5

Adding a Questionnaire

To Create a new Questionnaire you will to do 4 things:

1. Create an HTML web form to ask your questions.
2. Create a Java Model that represents the data from the form. (I promise this is simple)
3. A repository for storing your form data (Extremely simple)
4. Add details to the Questionnaire controller to allow you to correctly handle the form.

The questionnaire must have a unique name from all other questionnaires. It should not contain spaces, or special characters, thought in a pinch it could use an underscore " _ ". A good convention is camelCase, where you upper case individual terms in your unique name, such as "UniqueName"

You will see references to **myForm** in the steps below. Please replace this with the name of the form you are creating. You may also see **MyForm** at which point you should upper case the first letter.

5.1 Step 1: Create the html form

New forms should be placed in

```
/src/main/resources/templates/questions/**myForm**.html
```

It's a good idea to start with an existing form you like, and modify it. However, there is nothing to prevent you from creating the page you want from scratch. "Credibility" offers a good example of a simple one page form. "Demographics" shows a multi-page form. "DASS21" is a multi-page form with validation.

Be sure to give the HTML tag a unique action.This will be used over again to wire your new questionnaire into the system, so make it unique and descriptive. Making this the same as the file name of the form you are creating is recommended.

```
<form id="wizard" th:action="@{/questions/**myForm**}" method="POST">
```

From here, you just create your HTML form elements. Give thoughtful names to these elements, you will be using them again in the next step.

You can see your form as you develop it. Just execute:

```
gradlew bootrun
```

and visit http:\localhost:9000/questions/**myForm**

Any changes you make will be automatically visible by refreshing the page. You don't need to stop and start the server to see your changes.

5.2 Step 2: Create a Java class for containing your form

This should be located at:

```
/src/main/java/edu/virginia/psyc/pi/persistence/Questionnaire/**MyForm**.java
```

(please note the upper casing of the name)

This file defines how your data will be stored in the database. While this looks an awful lot like programming, it is a very boilerplate format, that can be quickly implemented over and over again.

It should look roughly like this:

```
package edu.virginia.psyc.pi.persistence.Questionnaire;

import lombok.Data;
import javax.persistence.*;

/**
 * The MyForm Web form.
 */
@Entity
@Table(name="MY_FORM")  // 1. The database is case insensitive, so using an _ can add␣
↪a lot of clairty here.
@Data // 2. This will create a getX setX method for all our privately declared values.
↪
public class MyForm extends QuestionnaireData { // 3. Be sure to extend␣
↪QuestionnaireData

   // 4. Define your form fields here using appropriate types.
   //    These should match exactly the "name" attribute on the
   //    the form elements created in Step 1.
   // -----------------------------------------------------
    private int someNumbericInput;
    private String anyTextualInput;
}
```

5.3 Step 3: Define a Java Repository

this file will be located here:

```
/src/main/java/edu/virginia/psyc/pi/persistence/Questionnaire/**MyForm**Repository.
↪java
```

The Repository is VERY simple, and consists completely of only the content shown below. It exists to give us a critical hook into the database if we need to access this data later in a unique way.

```
package edu.virginia.psyc.pi.persistence.Questionnaire;

import edu.virginia.psyc.pi.persistence.ParticipantDAO;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

 public interface **MyForm**Repository extends JpaRepository<**MyForm**, Long> {
     List<**MyForm**> findByParticipantDAO(ParticipantDAO p);
 }
```

5.4 Step 4: Wire up the Controller

There is a Questionnaire controller located at:

```
/src/main/java/edu/virginia/psyc/pi/mvc/QuestionController.java
```

You aren't creating a new file this time, just adding a new method to an existing file.

You need to define an additional method on this controller, that will take the data from the form covert it to our model in step 2, then use the repository in step 3 to store it in the Database. While this sounds complicated, the code is shown in full below.

```
    @Autowired private **MyForm**Repository **myForm**Repository;

    @RequestMapping(value = "**MyForm**", method = RequestMethod.GET)
    public ModelAndView show**MyForm**(Principal principal) {
        return modelAndView(principal, "/questions/**MyForm**", "**MyForm**", new␣
↪**MyForm**());
    }

    @RequestMapping(value = "**MyForm**", method = RequestMethod.POST)
    RedirectView handle**MyForm**(@ModelAttribute("**MyForm**") **MyForm** **myForm**,
                                 BindingResult result, Principal principal) throws␣
↪MessagingException {

        recordSessionProgress(**myForm**);
        **myForm**Repository.save(**myForm**);
        return new RedirectView("/session/next");
    }
```

That is it. When participants fill out your new form, it will be stored in a new␣
↪table named **MY_FORM** in the database. From here we can create various reports␣
↪to present this data which will be covered shortly.

Creating a Basic Application
==========

1. Create an Application class to start the Spring Boot framework.
2. Create an application.properties file that contains some basic settings
3. Generate a public and private key. Hold onto to the private key. (keep it secret,
↪ keep it safe) Place the public key in resources.
4. Create a service that implements the EmailService interface.
```

(continues on next page)
```

(continued from previous page)

```
5.

A Word on Good Practice
===========

The previous section about good practice on mindtrails data has now become a complete␣
↪manual of *[MTData CookBook](http://mtdata.readthedocs.io/en/latest/dataCookBook.
↪html)*.

Resources
==========

  - *[Everything about MindTrails in One Place](http://mtdata.readthedocs.io/en/
↪latest/index.html)*
```