PSExampleApp: Configurable and Open-Source App for Sensor Applications

Last document update: June 28, 2023

Application Note

Contents

1 Introduction

The PSExampleApp is a simple open-source application that guides a user to do a measurement with a PalmSens device to measure concentrations in a liquid sample. The app can be re-configured without code for use with (bio)sensors measuring a specific analyte using a linear calibration curve.

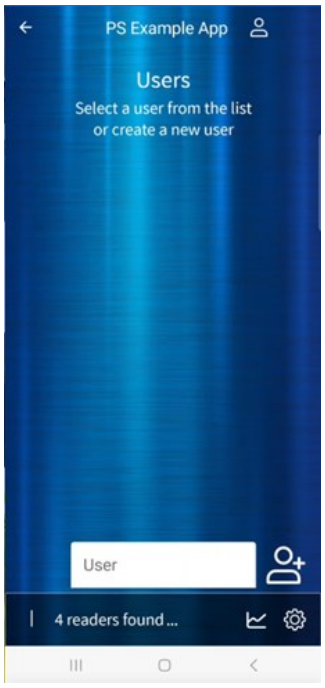
The application provides clear instructions, so that the user follows a simple flow to connect to a device and run a measurement with just a few taps.

The app's default configuration is set for detection of Heavy Metals using ItalSens screen-printed-electrodes for heavy metal detection.

It can be download from the stores (Apple Appstore and Google Playstore) and configured for your own need or serve as an example for your own app.

In the next chapter an overview will be given of the functionality of the application. It will describe the flow that the user goes through to do a measurement. The succeeding chapters will go more in depth. First it will describe how to configure the app by using the admin options and the last chapter serves as a technical guideline for customizing and developing the application from code.

2 Application Flow



2.1 User login, start measurement and connecting

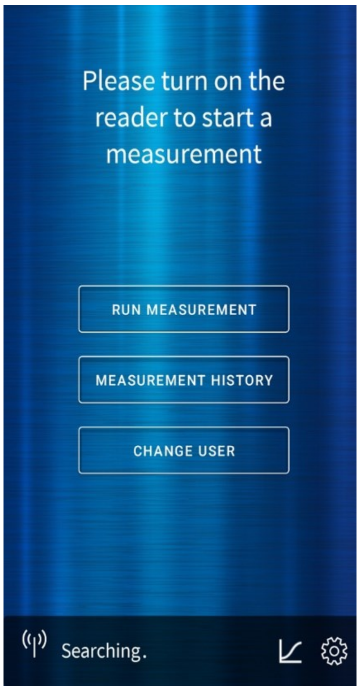
When a user starts the application for the first time, a screen will be shown where a user must be created. The application has a simple user management functionality where a user can be created by providing a username. New users can be added, or existing users can be changed (logged in) or removed at will. Every time the app starts it will login in with the last selected user. The user functionality mainly exists to couple a measurement to a user. This way multiple users can use the app on the same device without seeing each other's measurements. Users can be deleted by swiping the user name to the right in the user selection list. Note that deleting a user means that all measurement done by this user will be lost.

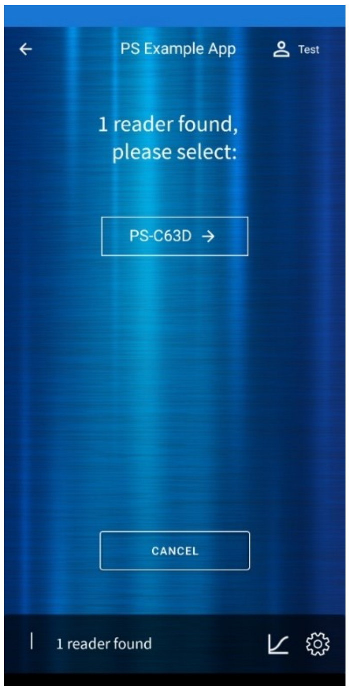
In the future different user roles can be added. This way you can have an admin user that has access to more advanced options.

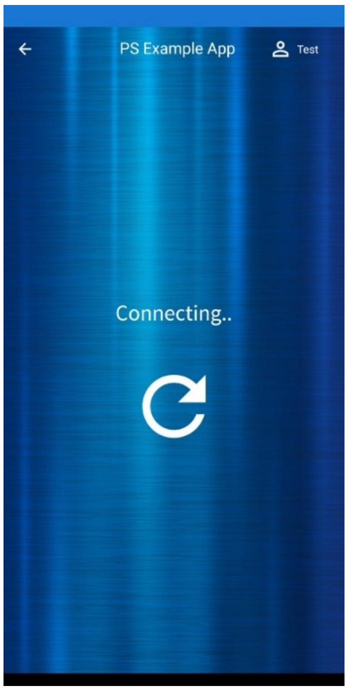
From the Home screen the user can start a measurement. Meanwhile in the background the Bluetooth scanner starts scanning for available instruments. The following instruments from PalmSens BV are supported:

- Sensit Smart
- Sensit BT
- EmStat3/3+ Blue
- EmStat Go
- PalmSens4

When user decides to run a measurement, he will be presented with a list of devices that are found. A user can select a device from the list to connect.



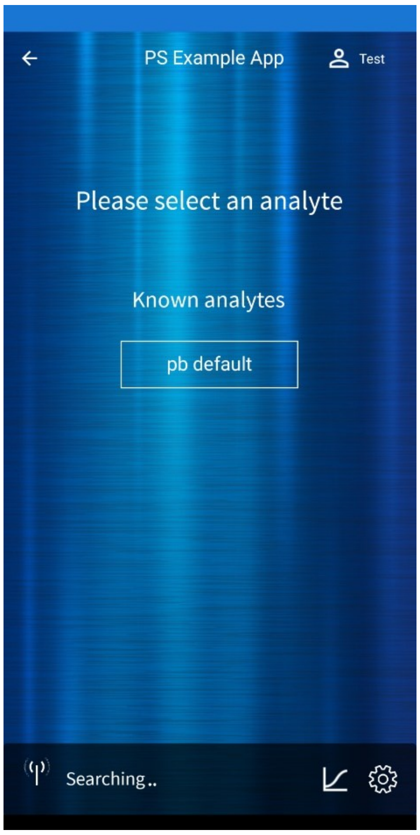




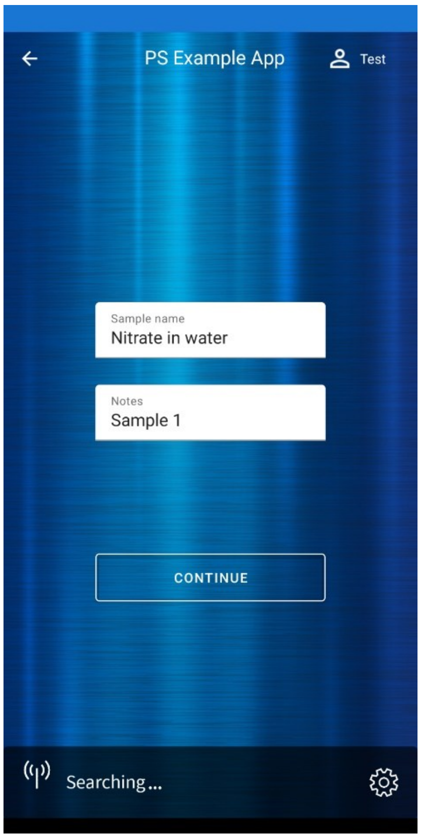
2.2 Select analyte, insert sensor

After the application has successfully connected to a device a user has to select an analyte to continue. The application comes with a few default analytes. However, a user can upload a custom analyte with its custom configuration. This process is described in chapter 3.2.

After selecting an analyte and setting a measurement name the measurement will start.

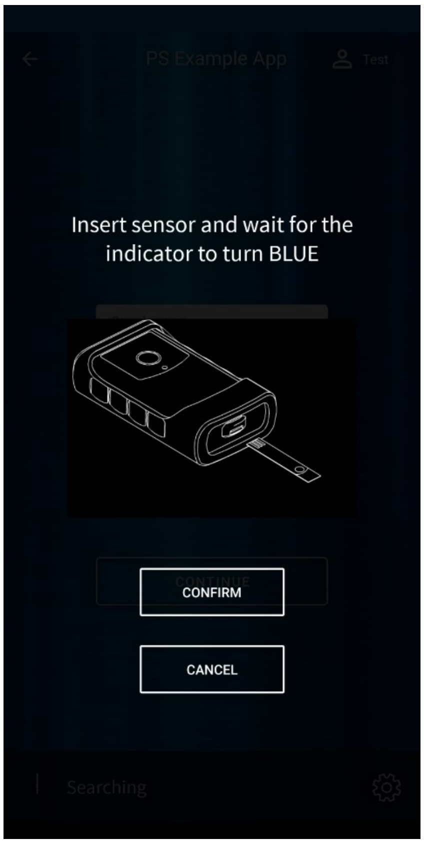


Please select an analyte



Sample name: Nitrate in water

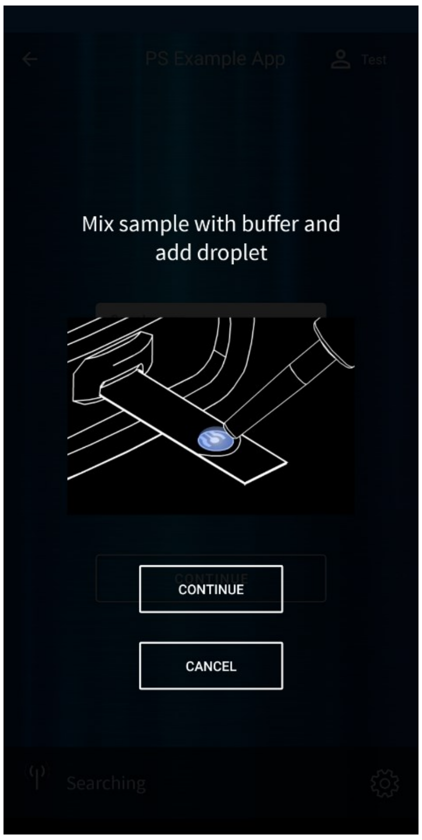
Notes: Sample 1

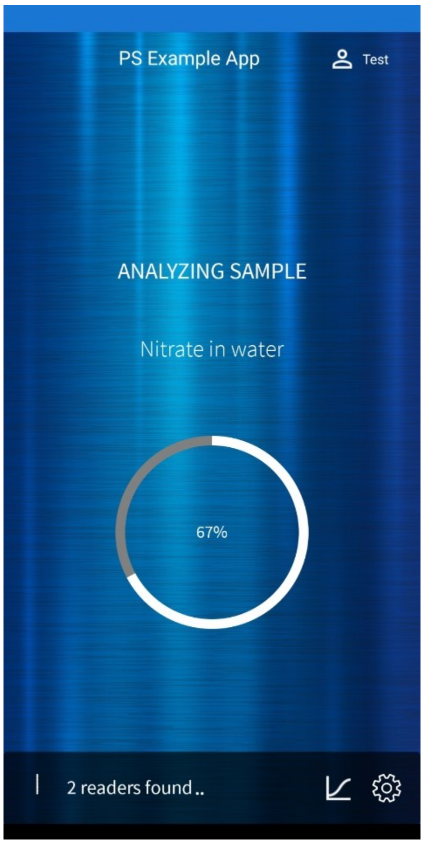


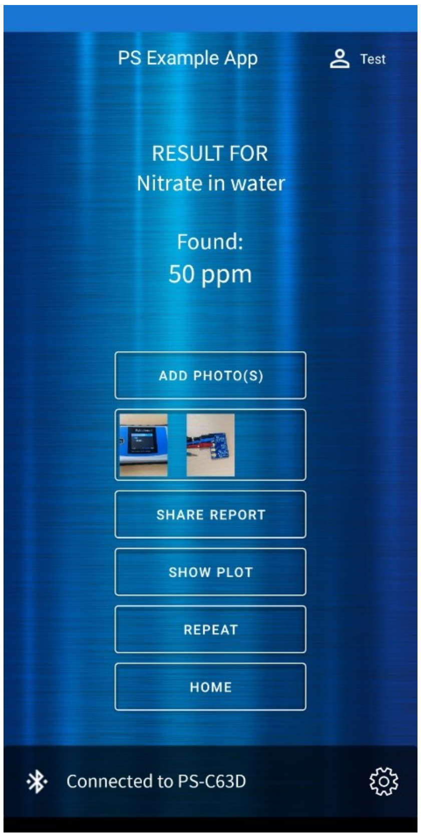
Insert sensor and wait for the indicator to turn BLUE

2.3 Apply a droplet, perform a measurement, and show the result

When the measurement is completed, the user can continue to the measurement finished screen. In this screen a user can take a maximum amount of three pictures that will be attached to the measurement. A user can also share a report. This will generate a PDF report with the measurement result and the pictures attached. This report can be shared through email, Google Drive, Whatsapp etc.





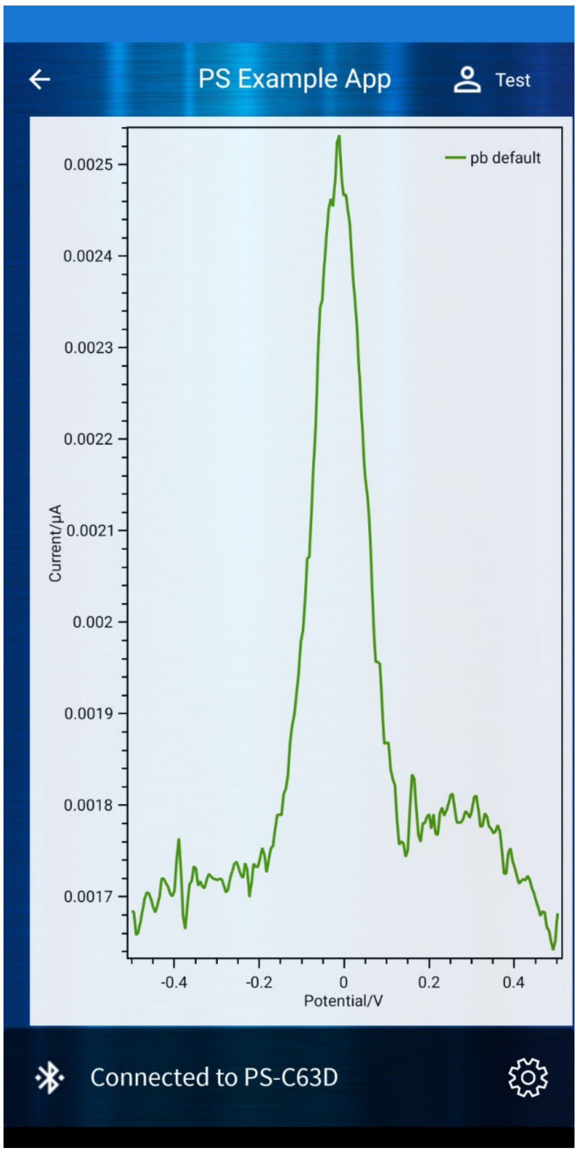


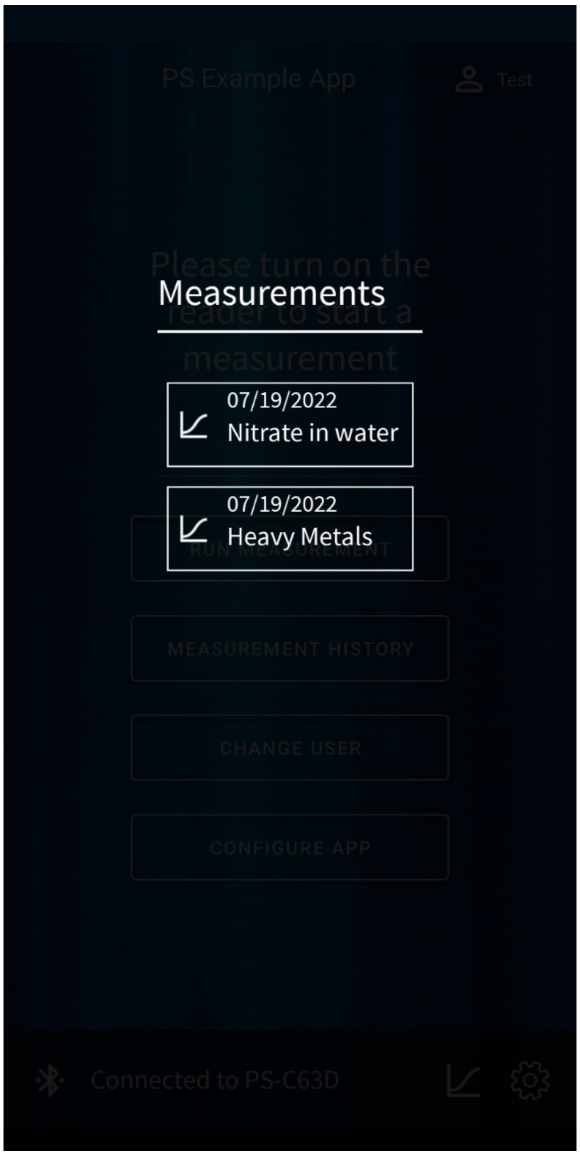
2.4 Show plot and overview of measurement history

From the measurement finished screen the user can also get a display of the plot with the measurement data. Aside from this a user can start a new measurement or go to the home screen.

A user can open a list of previous measurements from the status bar. If a measurement is selected from this list, then the user will be presented with a view that is similar to the measurement finished view with the same functionality.

Measurements can be deleted from the list by swiping it to the right. All measurement data and photo’s will be deleted.

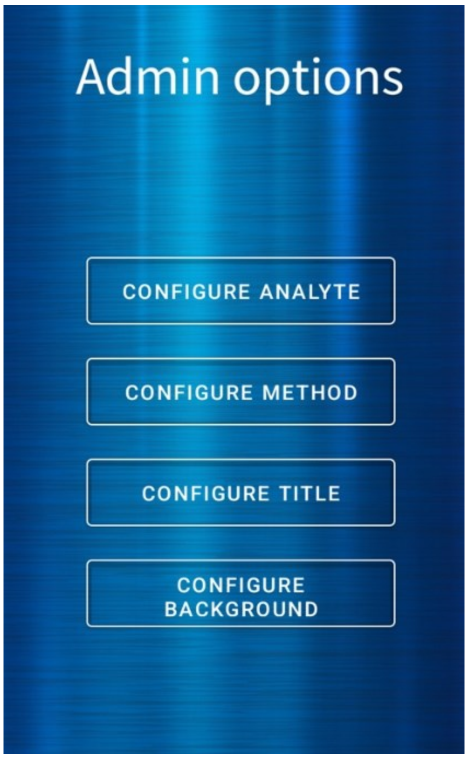




3 App Configuration

The application can be configured to fit the styling of the user.

To access the app configuration menu, you have to put a user in admin mode. This can be done from the user settings (cogwheel). When the admin mode is toggled on then from the home screen you can configure the application with the following options:



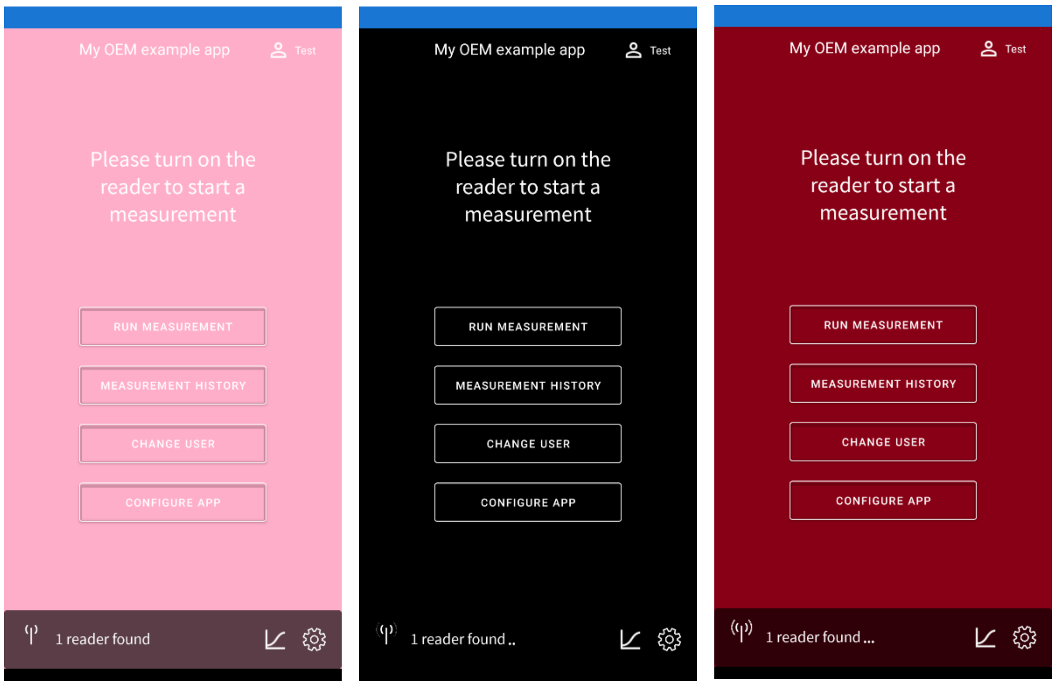
3.1 Custom style

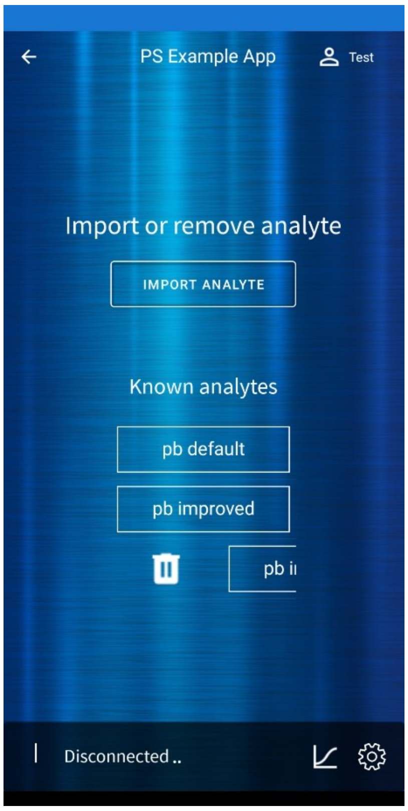
You can change the following styling of the application:

- Title that is shown in the top screen
- The background image of the application

Both options can be changed from the admin options. These changes will only take effect after the application has been restarted. When changing the background consider that the styling of the application is based on a dark background (for example white buttons with white frames). Changing it to a light background will make it hard to read for the user.

Also note that the background will be stretched according to the rotation of the screen and the screen dimensions. Abstract backgrounds are recommended.





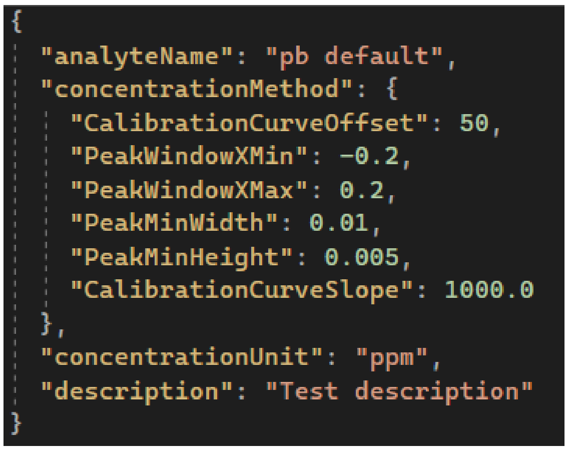
3.2 Custom analyte

The heavy metal application comes with its own default analytes that a user can use. This analyte calculates the concentration from the measurement data.

A user can add their own analyte configuration from the Configure Analyte menu. In this menu you can import an analyte by selecting a file from either the device itself or from another source like email or Google Drive.

The file that is selected must be a json file with the following properties:

In the example here you see the properties with its values. If you make a custom analyte, then you have to use the same properties, but you can change the values.



```
{
  "analyteName": "pb default",
  "concentrationMethod": {
    "CalibrationCurveOffset": 50,
    "PeakWindowXMin": -0.2,
    "PeakWindowXMax": 0.2,
    "PeakMinWidth": 0.01,
    "PeakMinHeight": 0.005,
    "CalibrationCurveSlope": 1000.0
  },
  "concentrationUnit": "ppm",
  "description": "Test description"
}
```

3.2.1 Analyte parameters in json file explained

- **analyteName**: name of the analyte that the app uses.
- **PeakWindowXMin**: potential in Volt where to start searching for peaks.
- **PeakWindowXMax**: potential in Volt where to stop searching for peak
- **PeakMinWidth**: minimum width in Volt of a peak at the minimum peak height. Peaks wider than the minimum width and higher that the minimum height are considered a peak.
- **PeakMinHeight**: minimum peak current in µA, compared to the level baseline. Peaks wider than the minimum width and higher that the minimum height are considered a peak.
- **CalibrationCurveOffset**: offset, used to convert peak current in µA to the concentration unit, using a linear formula concentration = slope * current (µA) + offset.
- **CalibrationCurveSlope**: slope, used to convert peak current in µA to the concentration unit, using a linear formula concentration = slope * current (µA) + offset.
- **Concentration unit**: unit of the concentration to detect, like for example parts per million (ppm).
- **Description:** a small explanation of the test.

3.2.2 Converting the measured current to concentration

The goal of each measurement is to find the concentration of a certain analyte. The relation between the measured current and the concentration has to be determined beforehand, by creating a calibration curve.

A calibration curve is determined by taking samples of a known concentration, and measure the current. An example curve is shown in Figure 1, with a linear fit. The best fit for a linear curve might as well be a second order polynomial or exponential function.

The PSTrace software for Windows as provided by PalmSens can be used for doing concentration determinations and calculating a linear calibration curve.

For the sake of simplicity, this example application has default support for a linear calibration curve. By modifying the code, a programmer can extend the types of calibration curves that are supported.

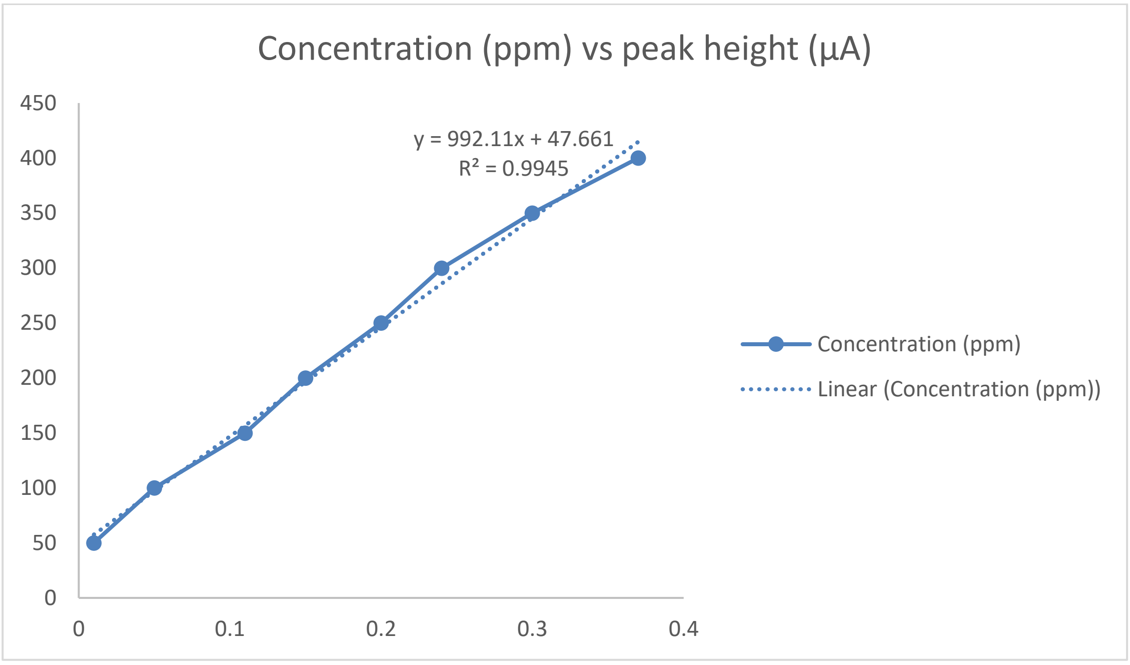


Figure 1: example calibration curve for the concentration, giving a slope of 992 ppm per µA and an offset of 48 ppm.

The measured current is in this example app converted to a concentration using a linear function:

$$y = ax + b \tag{3—1}$$

The linear function for the concentration is:

$$Concentration = slope * current + offset \tag{3—2}$$

3.2.3 Example calculation A

- the concentration unit is ppm,
- the slope is 992 ppm per µA,
- with an offset of 48 ppm.

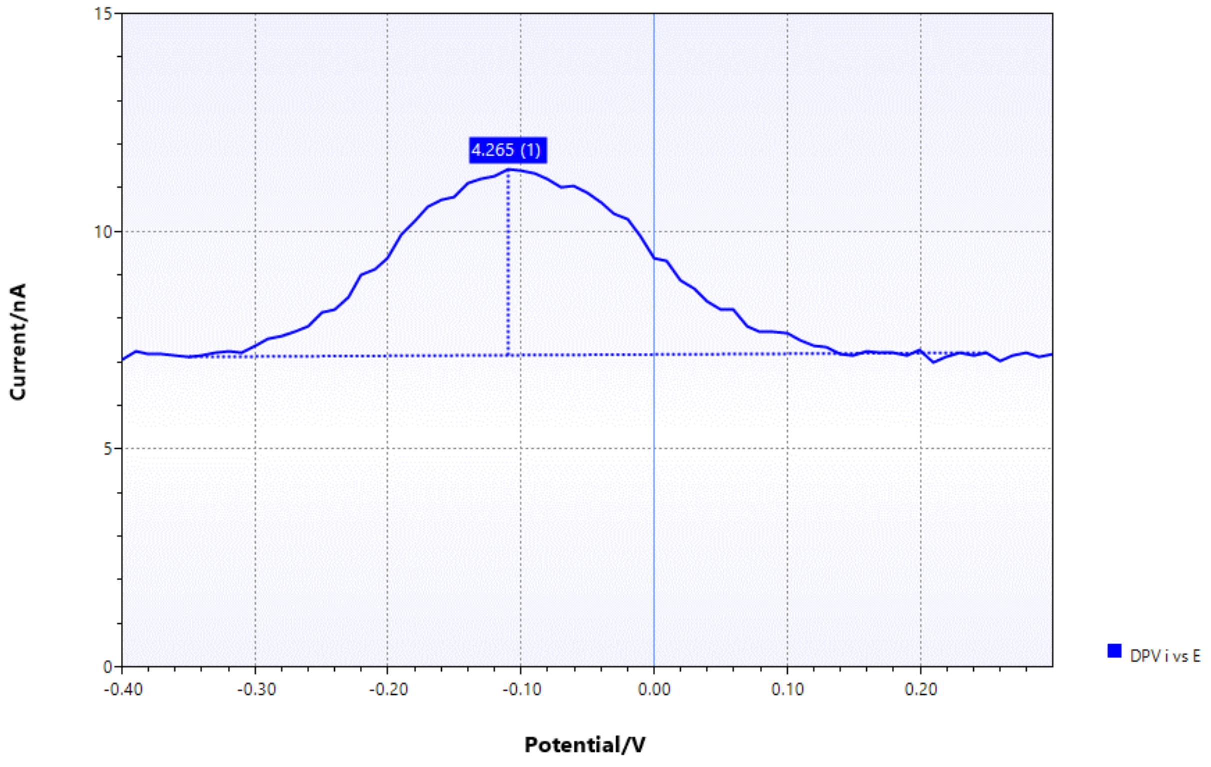


Figure 2: measured current in PSTrace, with a baseline of ~7 nA and a peak of ~4 nA.

A measured peak current of 0.004 µA will result in:

$$concentration = 992 * 0.004 + 47 = \ 51\ ppm \qquad (3—3)$$

3.2.4 Example calculation B

- the concentration unit is ppm,
- the slope is 992 ppm per µA,
- with an offset of 48 ppm.

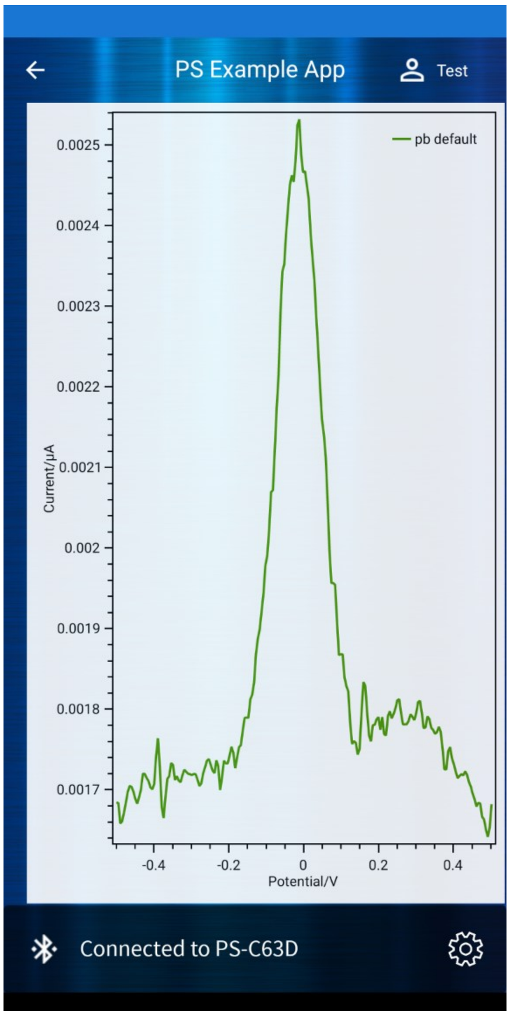


Figure 3 baseline of 17 nA, peak of 8 nA using the example application.

A measured peak current of 0.008 µA will result in:

$$concentration = 992 * 0.008 + 47 = \ 55\ ppm \qquad (3\text{—}4)$$

3.3 Custom method

The PSExample application uses a PalmSens method (.psmethod) to execute the measurement that is used to detect analytes, such as heavy metals. The result of this measurement is used by the analyte to calculate the concentration.

The PSExample application comes with a default .psmethod-file which is a Differential Pulse Voltammetry, used for Heavy Metal Detection. However, an admin user can change this method by selecting the configure method option in the app configuration menu. Currently the analysis is set up for Differential Pulse Voltammetry and Square Wave Voltammetry, the analysis will need to be modified when using a different technique.

When a user selects this option, they can select a file from the device or other sources like email or Google Drive. The file has to be of type .psmethod. When a psmethod file is selected the current psmethod file is overwritten. This means only 1 psmethod file can be used at a time.

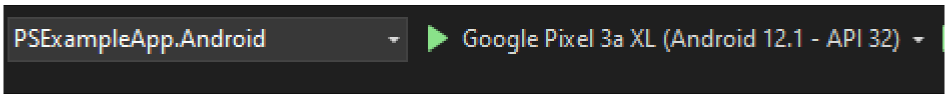
PalmSens provides software called PSTrace to generate psmethod files on a computer, or PSTouch to generate psmethod files on an Android device. For more information please check palmsens.com/software/.

This example application is made to work with voltammetric methods. The concept also works for other electrochemical techniques and impedance spectroscopy. However, an experience software developer will have to adjust the data processing according to the electrochemical method used.

4 Use Visual Studio to modify the code

This example app is a great start for a custom application to control a PalmSens potentiostat. To extend this example app:

- A C# programmer needs Visual Studio and the Xamarin SDK extension by selecting Mobile development with .NET.
- Download the code from GitHub. To avoid build errors, make sure the path to the extracted code is as short as possible, for example c:\code\.
- Open the PSExampleApp.sln in visual studio.
- Connect an Android phone and enable debugging mode in the developer settings of the Android smartphone.
- Click on the green play button to start building the app:



The app is based the PalmSens .NET SDK for Android. Documentation on the SDK can be found here.

5 Application structure

The application is set up with Xamarin. Xamarin is a cross platform framework that makes it possible to create applications that is compatible with Android and IOS.

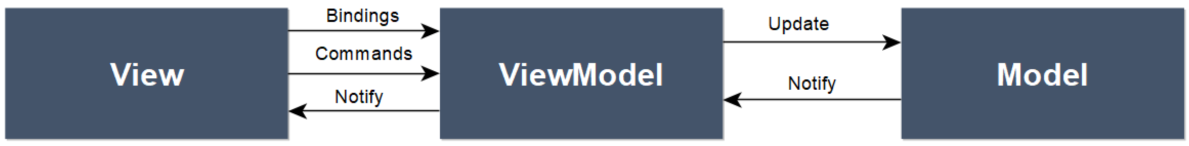
The language that Xamarin uses for the front-end is XAML. The backend is written in C#.

This chapter will describe the application structure from a developer point of view.

5.1 The Model-View-ViewModel (MVVM)

The application uses MVVM to structure the flow of control. MVVM has 3 kind of components:

- View: This is the part that the user sees. With the application this means the part that is written in XAML.
- ViewModel: The part that is connected to the view which sends updates of the user interaction to the model
- Model: The part that contains the business logic and responsible for the data



With the MVVM structure the front-end part written in XAML connects with the viewmodel through Bindings and commands. The commands register user interaction and sends it to the viewmodel. The bindings make sure that the data on the view gets updated when that data is changed in the view model.

By using bindings and commands the view doesn't have to contain any code-behind which means this doesn't have to be unit tested. Also, the view model doesn't have any connection with the view. This means that it's loosely coupled. The view model just updates its properties and through bindings on the view side the view will be updated.

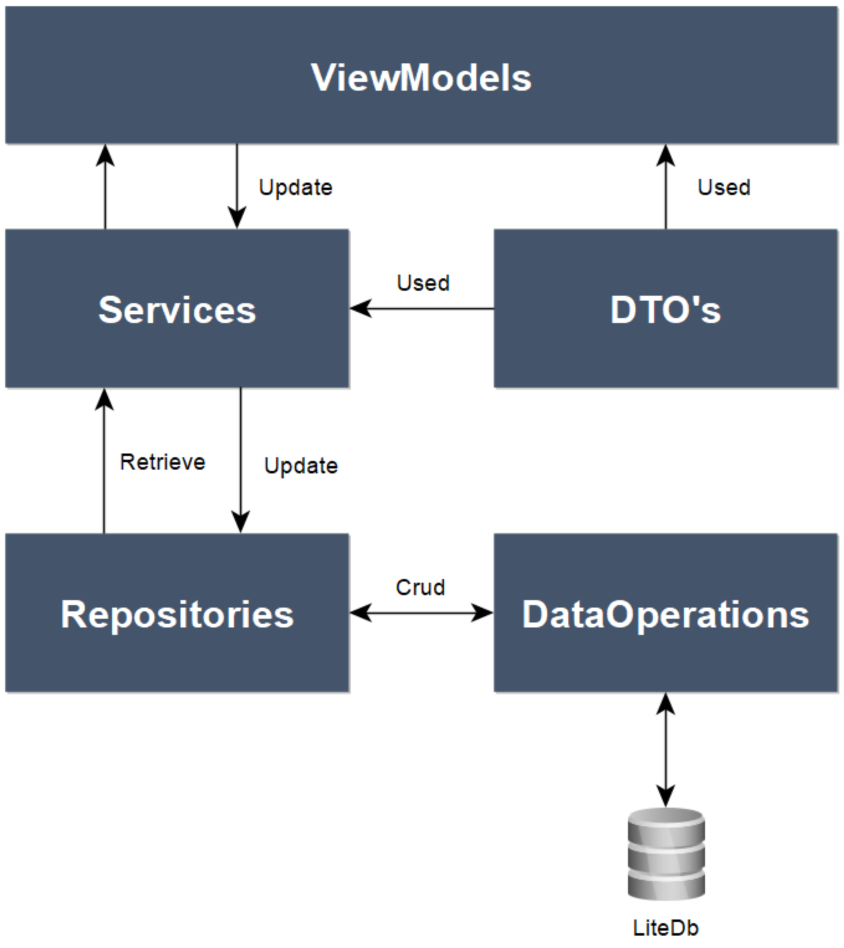
The model receives updates from the view model and process these updates through the business logic. The result will be sent as notifications to the view model mainly through events. This will have the same effect as the relation between views and viewmodels the model doesn't know anything about the view model. This decreases the coupling and makes the application more modular.

For more information about MVVM please visit the following link: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

5.2 Services, DTO’s and repositories

Even though in the last chapter the model is represented as a single component. This component consists of several types of classes:

- Services: These contain the business logic and can be called from the view model
- Repositories: Repositories are connected to the services. Repositories have method which are responsible for manipulating or retrieving data
- DataOperations: The class that has the database configuration
- DTO’s: Data transfer objects (DTO) are classes responsible for carrying data between processes. These classes are shared by the viewmodel and model.



The view models are connected to services. However, services are not aware of the view models classes. They only send requested data back or send notifications through events. Services can connect to other services to handle business logic or connect to one or more repositories to handle data related requests (saving, retrieving etc.).

Repositories have CRUD (create, read, update, and delete) methods for data handling. Repositories don’t have any connection to services or other repositories. They do have a connection to the DataOperation class which has database operations

For now, repositories handle only database related actions. In the future this can also be data related calls to export data (for example HTTP calls).

The services, repositories and data operations classes have a one-on-one interface. The communication between the model classes and viewmodel goes through interfaces. This serves as an extra layer of abstraction. For example, if we switch to a different kind of database then we only have to change the concrete data operations class.

5.3 Project's structure

The heavy metal application is a Visual Studio solution which consist of the following projects:

PSExampleApp.Forms

This project has the view, viewmodels and front-end related helper classes like converters.

PSExampleApp.Core

This project has the services, repositories and data operations classes and their respective interfaces. It also has helper classes only used by the model classes. The services in this project are all cross platform.

PSExampleApp.Common

This project contains classes that are used by both the Core and Forms projects. For now, it's only the DTO classes and some helper classes that are used by both projects.

PSExampleApp.Android

The Android solution that is created based on the Xamarin projects. It has some specific android configuration in the main activity class. And implementations of platform specific services.

PSExampleApp.IOS

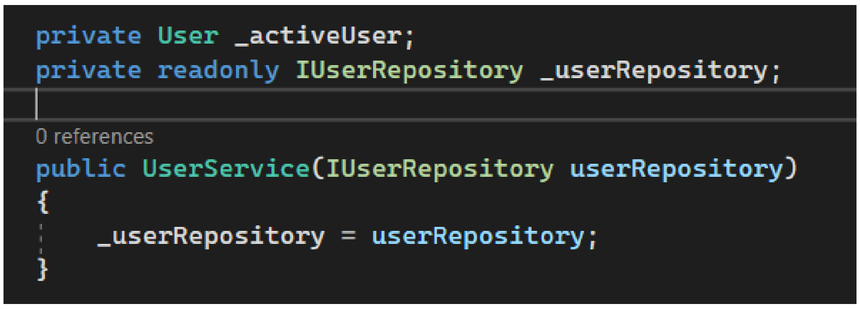
The IOS solution that is created based on the Xamarin projects. Like the Android project it has IOS specific configuration and services.

PalmSens.Core.Simplified.XF.Application & Infrastructure

These two projects contain interfaces for platform specific services that must be implemented in the platform specific projects. It also contains classes that are used by those services.

5.4 Dependency injection

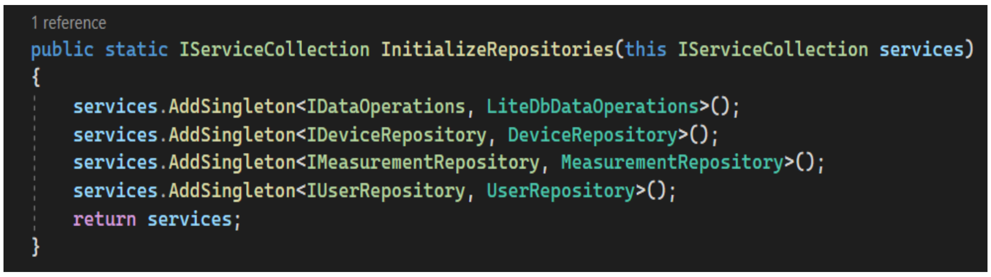
The application makes use of dependency injection containers. This mean that all services, repositories and view models are configured in the startup class of the .Forms project. To use a service or repository you can simply add it to the constructor:



```
private User _activeUser;
private readonly IUserRepository _userRepository;

0 references
public UserService(IUserRepository userRepository)
{
    _userRepository = userRepository;
}
```

The advantages of using dependency injection with IOC containers is that first, you have to declare outside dependencies in the constructor of the class. Which makes it less error prone. Also, it's easier for unit testing by using mocks for the outside dependencies. Another pro is this way makes sure you only use 1 instance of a class is used.

If you create a viewmodel, repository or service that needs to be registered then you can add it to the *DIContainers* class in the .Forms project. You can add the class to 1 of the existing methods:



```
1 reference
public static IServiceCollection InitializeRepositories(this IServiceCollection services)
{
    services.AddSingleton<IDataOperations, LiteDbDataOperations>();
    services.AddSingleton<IDeviceRepository, DeviceRepository>();
    services.AddSingleton<IMeasurementRepository, MeasurementRepository>();
    services.AddSingleton<IUserRepository, UserRepository>();
    return services;
}
```

You can add the class with its representing interface to the list. After that you can use the class from constructors.

For more information about dependency injection please visit the following link:
https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

5.5 Navigation

The navigation of the application app is done with a NavigationPage. This page has a navigation stack that keeps track of which page to display. This means that you have a home view which is the main page. If a user navigates to other pages, then that page is being pushed on the stack. If the user is using the back button or any other button that goes back a page, then that page is being popped of the stack.

For the heavy metal app there is a custom navigation dispatcher to support pushing and popping of pages asynchronously in view models instead of using code behind or view code in the view model.

To use this, you can call the static NavigationDispatcher class in the app and call the push or pop method:

```
await NavigationDispatcher.Push(NavigationViewType.SelectAnalyteView);
```

You can use the NavigationViewType to select the view where you want to navigate to. If you use the Pop method, then you don't have to specify a page.

5.6 Connection

The heavy metal app can connect to a PalmSens device either through Bluetooth or USB. For Bluetooth the reader should be on. When it's on it will detect PalmSens devices nearby. This detection will run in the background and will reset after every 5 seconds. When a device is selected then it will connect and disable the device detection.

The device service in the heavy metal app handles all device related functionality for the application and serves as a mediator between the heavy metal application and the PalmSens simple wrapper.

5.7 PS simple wrapper

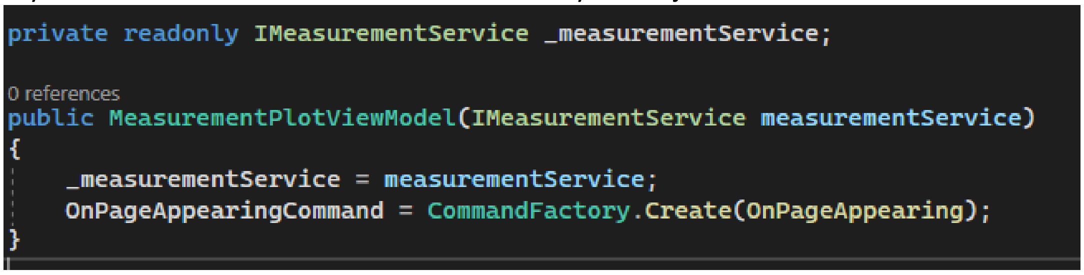
For connecting to devices, the heavy metal app uses the PalmSens simple wrapper. This is a wrapper class around the PalmSens core library. The simple wrapper is also used for SDK users to make applications for PalmSens devices.

5.8 Adding views and view models

In case of adding your own custom views and view models you have to follow the following steps:

1. Add a view by creating a xaml ContentPage.
2. Add a corresponding view model by creating a C# class
3. Set the viewmodel as binding context in the code behind of the contentpage
   ```
   BindingContext = App.GetViewModel<SelectDeviceViewModel>();
   ```
4. Add the view to the NavigationDispatcher by adding the view to the NavigationViewType enum and add the translation from enum to the actual page in the PageSelector method within the NavigationDispatcher. This way a view model can use the NavigationDispatcher to navigate to the view
   ```
   case NavigationViewType.SelectAnalyteView:
       return new SelectAnalyteView();
   ```
5. Last step is to add the view model to the InitializeViewModels method in the DIContainers class. This makes sure the App.GetViewModel method in step 3 can find the viewmodel. The viewmodel is added with a transient lifecycle this means it only exists when the corresponding view exists.
   ```
   1 reference
   public static IServiceCollection InitializeViewModels(this IServiceCollection services)
   {
       services.AddTransient<MeasurementPlotViewModel>();
       services.AddTransient<SelectMeasurementViewModel>();
       services.AddTransient<MeasurementFinishedViewModel>();
   ```

6. If you want to use services within a view model you can just add them to the constructor



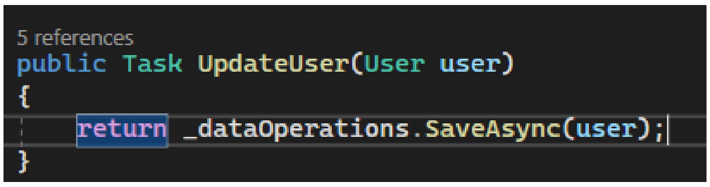
```
private readonly IMeasurementService _measurementService;

0 references
public MeasurementPlotViewModel(IMeasurementService measurementService)
{
    _measurementService = measurementService;
    OnPageAppearingCommand = CommandFactory.Create(OnPageAppearing);
}
```

5.9 Database

The PSExampleApp uses LiteDb to store objects and data. LiteDb is a no-sql database that is embedded. This means that the database is installed on the device that uses the application.

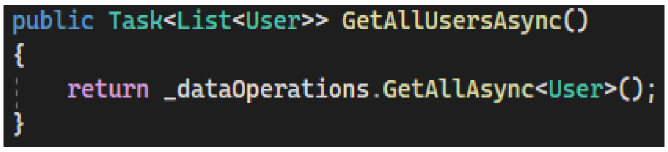
The way LiteDb works is that it has collections of C# classes that are saved. This means if you want to save data to the database you have to structure this data in a C# class. With this class you can call the IDataOperations interface. This interface has all the methods to save, load and delete data from the database.

The following example is a save method from the user repository



```
5 references
public Task UpdateUser(User user)
{
    return _dataOperations.SaveAsync(user);
}
```

The user information structured in the User class is saved to the database. If you want to load the data, you can use the load methods of the IDataOperations interface. You must put the class you want to load between the <T> brackets



```
public Task<List<User>> GetAllUsersAsync()
{
    return _dataOperations.GetAllAsync<User>();
}
```