Developer documentation

This document aims at providing guidelines for the developers who want to make the server production-ready. This document is not intended to be exhaustive.

Requirements

Best configuration for development / production:

- Windows 7/8/8.1/10 operating system.
- Visual Studio 2015 with support for C# and ASP.NET and IIS host.
- .NET Framework 4.5.

Alternative configuration for test (multiplatform):

- Windows, Linux, MacOS operating systems.
- Monodevelop with xsp4 host.
- Mono / .NET Framework 4.5.

When switching from one configuration to another, you need to replace the visual studio project file (.csproj), the Web.config files, the packages.config file.

The project is shipped with 2 versions of each of these files : (named xxx.Windows and xxx.Linux).
The “Windows” file works on VS2015, IIS host on Windows only !!
The “Linux” file works on Monodevelop, xsp4 host on **all operating systems**.

There is a python script included in the project to switch configuration : config.py.

How to switch configuration with the python script

Scenario: switching from the current configuration to the “Windows” configuration.
Solution: run config.py, and type “restore”, then “Windows”, then type “ok” and press enter. This will replace all the current configuration files (say “xxx”) by their Windows version “xxx.Windows”.

Scenario: saving your configuration to “Windows” configuration, to commit it.
Solution: Run config.py, type “save” then “Windows”, then type “ok” and press enter.

Requirements of the production server and deployment are detailed in the Deployment section.

I. Implementing database context and interfaces

Database Interfaces

Specifications

First, look at the files included in *Models/DatabaseInterface*.
This folder contains C# interfaces which describe the operations that must be implemented (*IAuthApi.cs*, *IBookingsApi.cs*, etc…).
Each of the operations are documented within the code.

```
/// <summary>
/// This interface describes the messages that can be send to get informations
/// about bookings.
/// This information is needed to perform room searching with the availability constraint.
///
/// The final implementation of this will be to look up for the bookings in the
/// outlook database.
/// </summary>
public interface IBookingsApi
{
    /// <summary>
    /// Gets a list containing every booking between the given startDate and endDate.
    /// The bookings which are not totally covered by the given time period (startDate -> endDate)
    /// are also included.
    /// The bookings are represented by their id.
    ///
    /// The bookings must be sorted by StartDate (ascending).
    /// </summary>
    /// <returns>The bookings.</returns>
    /// <param name="startDate">Start date.</param>
    /// <param name="endDate">End date.</param>
    List<BookingIdentifier> GetBookings(RoomIdentifier identifier, DateTime startDate, DateTime endDate)
    /// <summary>
```

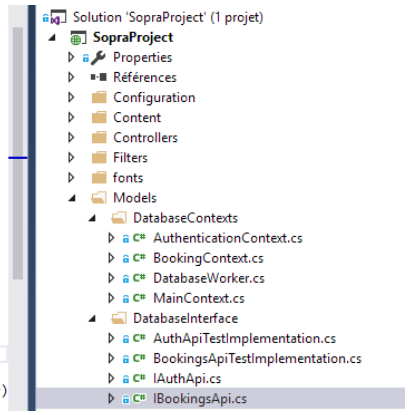

Figure 1. exemple of an interface : IBookingsApi

Implementation

Test implementations for all the interfaces can be found in the same folder (*Models/DatabaseInterface*). They rely on Database contexts, which establish direct connections to the databases.

```
public List<BookingIdentifier> GetBookings(RoomIdentifier identifier, DateTime startDate, DateTime endDate)
{
    List<BookingIdentifier> val;
    using (var ctx = new DatabaseContexts.BookingContext())
    {
        var query = from booking in ctx.Bookings
                    where booking.EndDate > startDate && booking.StartDate < endDate &&
                    booking.RoomID.ToString().Equals(identifier.Value)
                    orderby booking.StartDate ascending
                    select booking.BookingID;

        val = query.ToList().ConvertAll(id => new BookingIdentifier(id.ToString()));
    }
    return val;
}
```

Figure 2. Exemple of operation implementation

You can look at the implementation to better understand what the operations must do.

Database contexts

The database contexts are objects which expose database functionality.

Understanding the current implementation

The current implementations of the Database contexts use a MySQL database backend, and the EntityFramework relational mapper to map objects to classes, but many database engines are supported.

Database contexts implementations can be found in the *Models/DatabaseContexts* folder.
Example:

```
public class BookingContext : DbContext
{
    /// <summary>
    /// Bookings database set.
    /// </summary>
    public DbSet<BCBooking> Bookings { get; set; }
    public BookingContext() : base("bookingContext")
    {
        /// …
    }
}
```

Two things are relevant in this code:

- The `BookingContext` inherits from `DbContext` (EntityFramework).
- `"bookingContext"` is a connection string. The connection string refers to an entry in the server configuration (see below), indicating how to connect to the database.
- `DbSet<BCBooking>` represents a set of BCBooking objects. The BCBooking object will be "analysed" by EntityFramework and mapped to database tables in the backend engine corresponding to the connection string.

About the connection string:
We can find it in the Web.config file:

```
  <connectionStrings>
    <!-- … -->
    <add name="bookingContext" providerName="MySql.Data.MySqlClient"
connectionString="server=localhost;port=3306;database=booking;uid=sopra;password=sopra
" />
  </connectionStrings>
```

Implementing new contexts

If you want to implement new contexts, you can either:

- Use EntityFramework (like in the current implementation) to create the database contexts, if your database backend is supported. In that case, just look at the current implementation to see how it is done, and have a look at the *EntityFramework* documentation.
- Make your own context (ex: manually connect to a server and make queries) and use it in your interface implementation.

II. Identifiers and coherency

Identifier are a way to add type information about database identifiers.
They are used, for instance, to ensure that using an room identifier to identify a site will lead to a compilation error !

They are also used so that an object which has **different identifiers in two databases** can be represented by **one identifier object**, holding identifier data for all databases!

Example of identifier:

```
/// <summary>
/// Represents a Site identifier.
/// </summary>
public class SiteIdentifier : Identifier<string>
{
    public SiteIdentifier(string id) : base(id) { }
}
```

The underlying identifier type (`string` here) could also be an `int`.

Complex identifiers

If you want to hold 2 identifiers (one from the Exchange DB and one from the custom SQL DB) in this object, simply add a property holding the secondary identifier.
Keep note that you must be able to create an identifier instance with a primary identifier only (e.g. **DO NOT** use something like `Identifier<Tuple<string, string>>`)!

Here is an example of complex identifier :

```
public class ComplexIdentifierExample : Identifier<string>
{
    public int SecondaryIdentifier
    {
        get
        {
            // Your transformation logic here !
            // You can make database request to look up correspondances in a table
for instance.
            // Here we just convert the primary identifier to an int !
            return Int32.Parse(this.Value);
        }
    }
    public IdentifierExample(string primaryId) : base(primaryId) { }
}
```

III. Getting a production version

1. Requirements

The following components are required to put the server into production :

- Windows server
- IIS Express 10.0
- .NET Framework 4.5
- Visual Studio 2015 Community Edition
- git

2. The way to a production version

Getting the code

Here is how to get the code for the release version:

```
git clone https://github.com/Scriptopathe/project-sopra
git checkout 1-0-stable
```

Setting up the production database

In order to get the project working in your environment, you must:

- Write your own implementation of:
    - `SopraProject.Models.DatabaseInterface.IBookingsApi`
    - `SopraProject.Models.DatabaseInterface.IAuthApi`
    - `[optional] SopraProject.Models.DatabaseInterface.ISitesApi`
    - `[optional] SopraProject.Models.DatabaseInterface.IUserProfileApi`

  `[optional]` means that the provided implementation may be suitable for production.
- Register your implementations in `SopraProject.Models.ObjectApi.ObjectApiProvider`

```
private ObjectApiProvider()
{
    BookingsApi = new BookingsApiTestImplementation();
    AuthApi = new AuthApiTestImplementation();
    UserProfileApi = new UserProfileApi();
    SitesApi = new SitesApi();
}
```

- Change the connection strings in the *Web.config* file if necessary.

```
<connectionStrings>
    <add name="mainContext" providerName="MySql.Data.MySqlClient"
connectionString="server=localhost;port=3306;database=sopra;uid=sopra;password=
sopra" />
    <add name="authContext" providerName="MySql.Data.MySqlClient"
connectionString="server=localhost;port=3306;database=auth;uid=sopra;password=s
opra" />
    <add name="bookingContext" providerName="MySql.Data.MySqlClient"
connectionString="server=localhost;port=3306;database=booking;uid=sopra;passwor
d=sopra" />
</connectionStrings>
```

3. Security in the production version

The login and password are transmited as **clear text** when trying to log in.

Before putting the server in production state, you **MUST** be sure your IIS Server has SSL enabled, and ensure that any request with SSL disabled will result in an error/redirect.

In the 1-0-stable configuration, SSL is enabled and any request without SSL will result in a redirect. However, this redirect does not work correctly in DEBUG mode (e.g. when debugging with Visual Studio) but is **designed for production**.

For instance, while debugging using Visual Studio, the server will redirect requests from http://localhost:57270/ to https://localhost:57270/. However, the server will be listening for SSL connections on port 44300.

Test the SSL enabled server in debug mode

Just go to https://localhost:44300/ instead of https://localhost:57270/.

4. From test server to production server

For our demo, we put the server on Microsoft's Azure Cloud platform. We will not cover this process as you will be deploying on your own servers.

Unfortunately, we were not able to try to deploy our server on a Windows Server environment. However, this environment is actually the closest to the development environment (same hosting server : IIS).

Here are some resources that might help you deploying the project on your production server :

- **Installing IIS (if needed) :** http://www.howtogeek.com/112455/how-to-install-iis-8-on-windows-8/?PageSpeed=noscript
- Deploying to IIS : http://www.asp.net/mvc/overview/deployment/visual-studio-web-deployment/deploying-to-iis (skip database related stuff).
  http://docs.asp.net/en/latest/publishing/iis.html