Developper documentation

# Requirements

Best configuration for development:

* 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 (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.**

# 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.

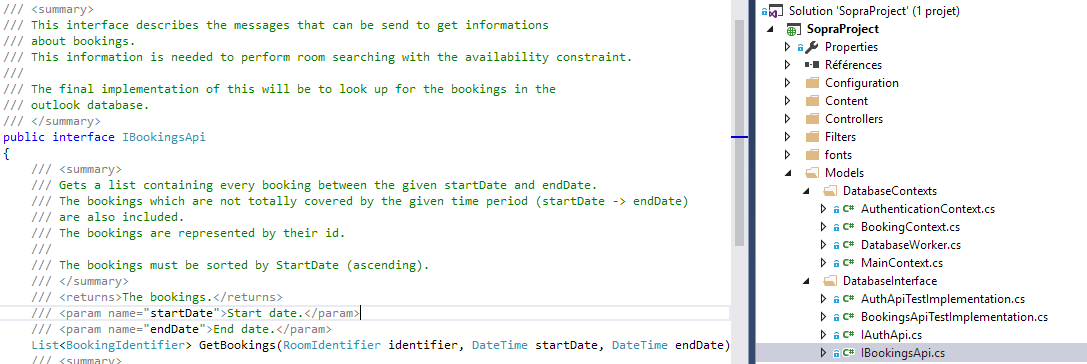


Figure 1. exemple of an interface : IBookingsApi

### 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.

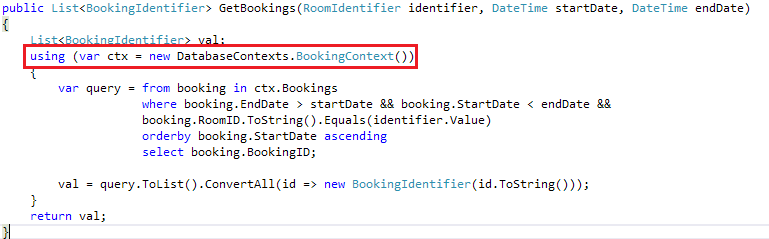


Figure 1. Exemple of operation implementation

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.

# Object API and cache

# Identifiers and coherency

# Controllers

# Deployment on IIS