We2Book : Mini-Program

Operations Manual

Handbook 1

Authors:
Dagfinnur Ari Normann
Elmar Þór Aðalsteinsson
Pálmi Þormóðsson

Contents

List of Tables

List of Figures

1. Introduction

This document is intended for the operational managers and employees of the We2Book mini-program service. It provides step-by-step guidance for future developers approaching the system in a correct and efficient way.

2. The System

The We2Book system is a feature within WeChat called mini-program. It uses RESTful API web service to retrieve data from a PostgreSQL database. The API and the database are hosted by Amazon Web Services in an Elastic Beanstalk cloud environment. The web service is written in the .NET core framework written mostly in C#. The front end of the system is written in the JavaScript, WXML, WXSS, and JSON programming languages, and are all modified forked versions of other well-known languages. JavaScript, or ECMAScript, is for an instance, a modified version of JavaScript 5.

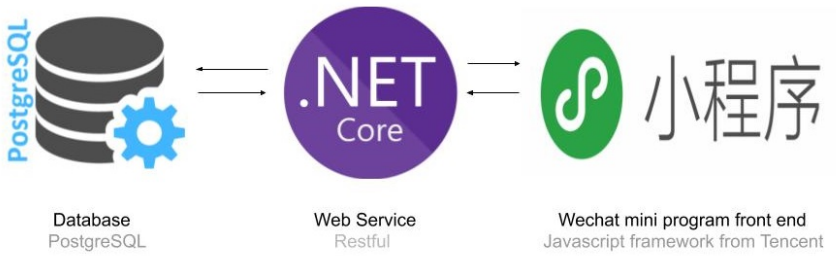

Fig. 1. The Three Layered Structure

3. Development Requirements

3.1 The Setup

The system is maintained on GitHub, developers will need to access it through a secured shell. Developers are required of the WeChat development tool and the .NET core framework.

Table 1. Recommended tools and download links

.NET core	Download Link
Git	Download Link
Wechat development tool	Download Link

How to run the front end:

1. Download Wechat development tool.

2. Clone the repository https://github.com/dagfinnur/EuropePay.git to your computer.
3. Run Wechat developer tool.
4. Choose Import Project.
5. Find directory to the folder you created with Github.
6. Use company AppID, otherwise use Test Account.
7. Choose import.
8. Compile.

How to run back end:

1. Download .NET core.
2. Clone the repository https://github.com/dagfinnur/we2bookAPI.git to your computer.
3. Have Visual Studio installed on machine.
4. Open project in visual studio.
5. To run project go to WebApi folder and run dotnet build and dotnet run.
6. To run tests go to XUnitTest folder and run dotnet tests with dotnet test.
7. To run code coverage go to XUnitTest folder and run dotnet test /p:CollectCoverage=true

3.2 Implementation

The system's development environment consists of a code editor, a compiler and a debugger. It has an inbuilt simulated view of the mini program, giving developers a precise preview of their design on display. Developers can choose between different devices to test their designed view on, which helps with responsiveness.

4. Coding Rules

In order to maintain efficiency and quality of the system, developers need to follow certain set of coding rules. The coding rules that were applied to this system are listed in the subsections below, categorized by its programming language.

4.1 C#

The first and foremost rule when writing in the C# programming language for the system is using PascalCase for class names, public member names, namespaces, method names, and the file names. The second rule is to use camelCase for member names that are not accessible publicly. The third rule is to use descriptive names for every object in the code, avoid jargon names. Developers need to follow the three-layered architecture of the system. No added spaces, to keep the code format clean from unnecessary line spacing. Comment the code where needed, developers may write and place a more detailed description in a comment above a code snippet to clarify its purpose. Use interfaces to provide contracts that objects can use to work together without needing to know anything else about each other, and dependency injection to make the code maintainable and reduce the run time.

4.2 JavaScript

The mini program's interface logic is written in JavaScript. Each page of the interface service consists JavaScript file, the file is placed into its page's folder and the file name should be the same as its folder. Every JavaScript file then consists of a Page() function which is used to register a page to the mini program and its data. The Page() function accepts an object as an parameter which should be used to affect the initial data representation of that page. Developers are condign to make use of the life cycle functions, nested within the Page() function, to trigger functional methods when particular event occurs. Each life cycle functionality can be found in table 2.

Table 2. Life Cycle Functionalities

1.	The **onLoad()** function runs at registration.
2.	When a page loads, the **onShow()** function is called.
3.	The first time a page is displayed, the **onReady()** is called to render its view.
4.	When the jumping from a page to another, the **onHide()** function is called.
5.	**onUnload()** function is called when exit the page view

The pages are displayed in a structural order by using components. Components are delivered from the database through the system's web service compartment, accessed through an API. The structure of each page must be placed within a wrapper called data, placed under the Page() function. The components control what kind of information each data contains, and can be manipulated using the life cycle functions. In order to change data asynchronously, we make use of event logic within the life cycles, triggered by the event handler from the WXML.

4.3 WXML

WXML is a WeChat programming markup language representing the system's structure using the components gathered from JavaScript. The components can be manipulated by

JavaScript event calls withing the life cycle functions. WXML code is similar to HTML code, the main differences between those two are the tags used to construct the page view.

5. WXSS

The WXSS file is the style sheet for the mini program, it has almost all of the features that CSS has to offer. Styles written in app.wxss are of global value, meaning that it represents the style of all the pages in the program. Styles written in a particular page acts only for that particular page and overwrites the selectors from the global style sheet if it is configured again. Selectors in WXSS are different from the selectors in CSS and make use of different size units for elements. Size units in WXSS use rpx (responsive pixel) as a unit. It allows to adjust pixels according to the width of the screen. It's fine to use px as a unit, if developers are used to the CSS styling.

6. Web Service Structure

The web service is constructed into three layers and follows the standard RESTful format. The repository layer is placed at the bottom, error handling and data from the database is retrieved from there with Entity framework. The service layer is in the middle of the project layered architecture, the data is sent to the middle layer where calculations are performed. Calculated data retrievals are sent from the service layer to the top layer, called the presentation layer. As developers we interact with the presentation layer in order to fetch data, but keep in mind the orderly construct of our layered architecture if catastrophic failure is in bound or maintenance of the system is needed.

6.1 Debugging

Developers may write unit tests to debug certain functions or smaller units within the web service. Unit tests are performed in the XUnit folder where xUnit and Moq frameworks are made of use to test the project's code.

6.2 Project's Folders

When an MP is created, the IDE automatically generates files within folders that are required for the program to run smoothly, including few files that can be used purposely to add functionalities to every page of the MP and thereby excludes the need to write exhaustive functions repeatedly. Functionalities of this kind are widely known as global functions and are written in files called app.js, app.json, and app.wxss, which are automatically generated by the IDE. Developers can construct the overall look and feel of their MP by configuring these files. Other files generated on creation are jsconfig.json, project.config.json, and sitemap.json. They may be configured to set up the routing, network timeout exceptions, window preferences, debugging methods, and the core compiling methods of the

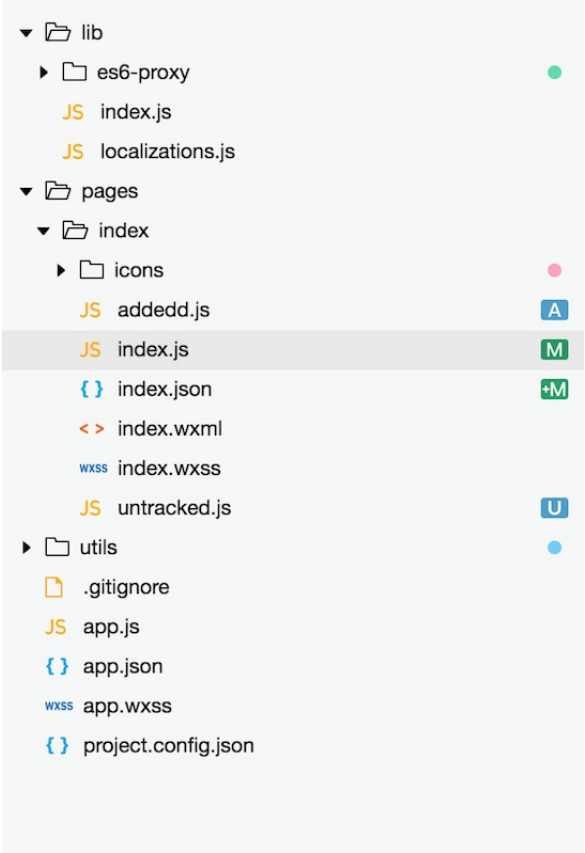

Fig. 2. WeChat Tree Structure

program. The IDE automatically integrates those functions on the creation and when a new page is added to the MP.

7. Deployment

The system makes use of a service called CircleCI, CircleCI is linked to the projects GitHub repository. Each time a new version of the API web service is pushed onto the API GitHub repository, CircleCI runs predefined unit tests made by the developers to debug certain functions or smaller units within the web service. The API cloud service is hosted on a virtual engine provided by AWS service, Elastic Beanstalk. CircleCI is also linked to this virtual environment as an IAM user, which is a user role configuration within AWS to define its role and permissions. A bucket of the virtual engine had to be created in order to grant IAM users the correct permissions. When CircleCI has performed the tests, it starts zipping the project version files needed for deployment. After zipping the required version files, CircleCI deploys a zip folder onto the AWS Elastic Beanstalk virtual machine. CircleCI plays a crucial role in the testing- and deployment phase of our system, whereas developers are only required to add, commit, and push files as usual to the system's GitHub repository.

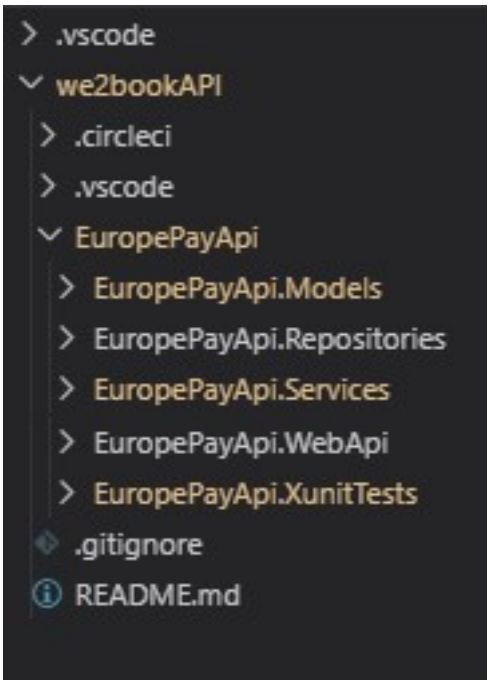

Fig. 3. Web Service Tree Structure

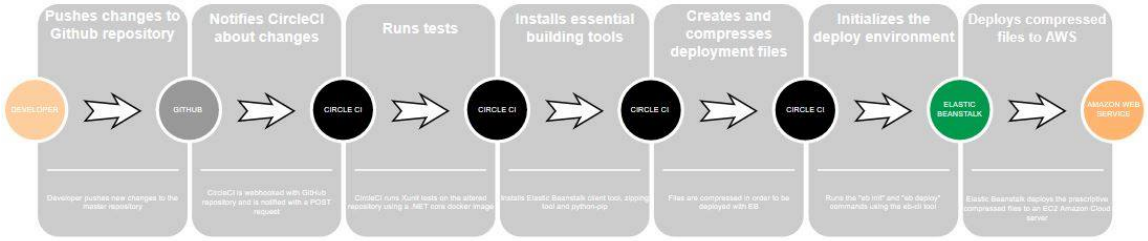

Fig. 4. Continuous integration and deployment process