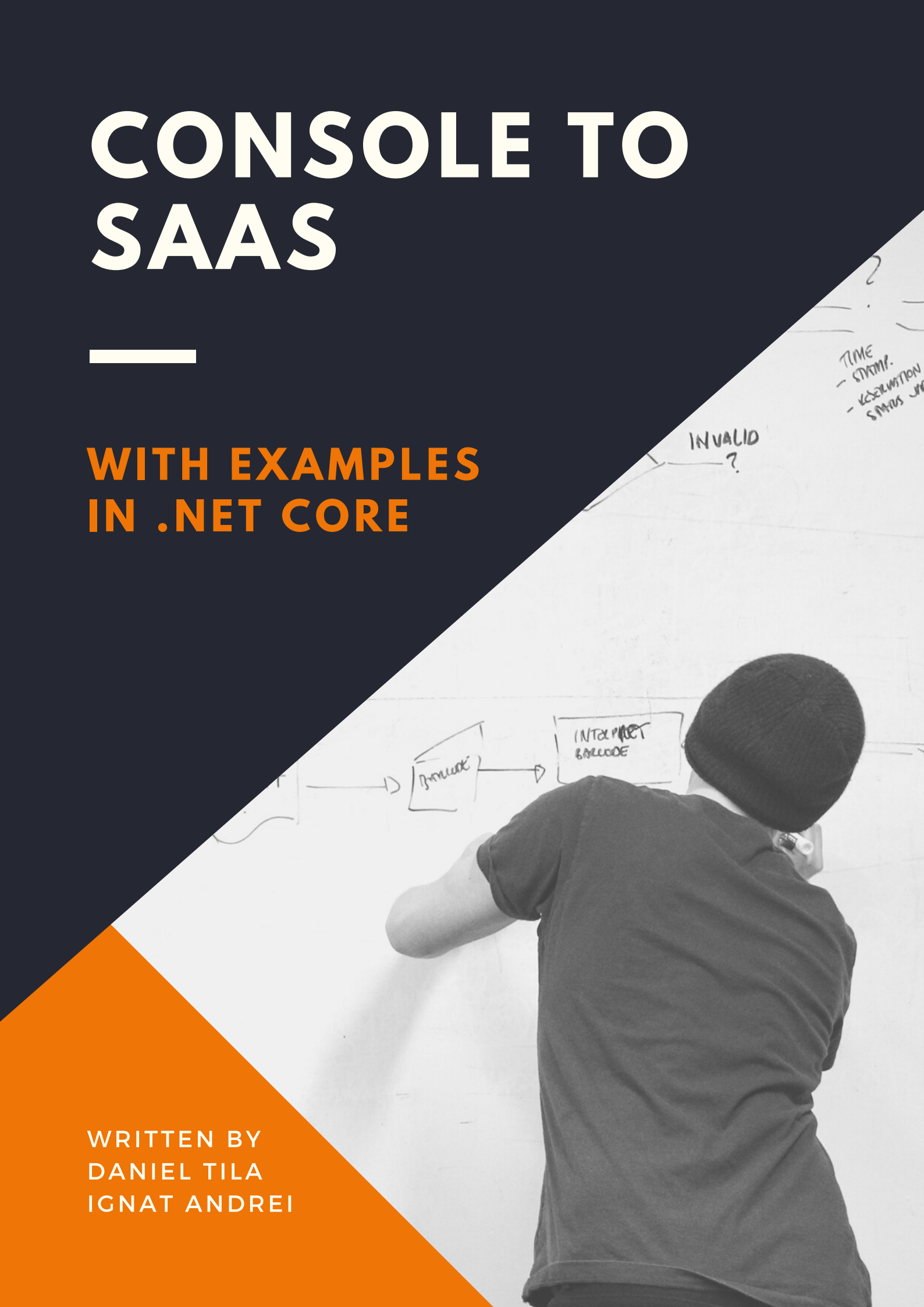
CONSOLE TO SAAS

WITH EXAMPLES IN .NET CORE



WRITTEN BY
DANIEL TILA
IGNAT ANDREI

Console to SAAS

How to transform a Proof Of Concept application to a Software As A Service product

This book will guide you step-by-step how you can build a scalable product from a proof of concept (https://en.wikipedia.org/wiki/Proof_of_concept) to a production ready SAAS (https://en.wikipedia.org/wiki/Software_as_a_service).
Any development done will start from a business need: this will make things clear for the team what is the impact of the delivery.
You will see different arhitecture patterns for separation of concerns (https://en.wikipedia.org/wiki/Separation_of_concerns) and why some of them fit well and some of them not. Everything will happen incrementally and the product development history will be easy to be seen by analyzing commits.
To have our finished product, we will have two development directions:

On premise solution

- how the idea get a shape
- creating the PoC
- bug fixing
- componentization / testing / refactoring
- unit and integration testing
- extending functionalities

As a service

- transforming to a web application
- authentication
- possible arhitectures
- continuous deployment
- tight coupled microservices
- messaging systems / loose coupled microservices

Download the book for free

You can find the code source at
https://github.com/ignatandrei/console_to_saas

You can read online this book at https://ignatandrei.github.io/console_to_saas/

You can read offline by downloading the

1. PDF (https://ignatandrei.github.io/console_to_saas/consoleToSaas.pdf.html)
2. Word (https://ignatandrei.github.io/console_to_saas/consoleToSaas.docx.html)
3. Epub (https://ignatandrei.github.io/console_to_saas/consoleToSaas.epub.html)

4. ODT (https://ignatandrei.github.io/console_to_saas/consoleToSaas.odt.html)

Source codes

You can find source codes at Console2SAAS (https://github.com/ignatandrei/console_to_saas/) or after each chapter.

How the chapters are organized

Each chapter has the following steps:

1. A *description* of the business challenge
2. An *exercise* for the reader
3. A *technical analysis* - what should be modified in order to do the project
4. Some *code* that shows how the challenge is solved (also, link to the full code to be downloaded)
5. *Further reading* - if the reader wants to learn more

What we will build

The application will be a simple one: from multiple word documents we will build a summarizing Excel file. The point here is not to build the application 100% production-ready, but our focus will be on code organization and discuss the common problems while on this journey.

Feedback

Please share feedback by opening an issue or pull request at https://github.com/ignatandrei/console_to_saas/

Authors

Andrei Ignat - http://msprogrammer.serviciipeweb.ro/

Has more then 20 year programming experience. He started from VB3, passed via plain old ASP and a former C# Microsoft Most Valuable Professional (MVP).
He is also a consultant, author, speaker, www.adces.ro community leader and http://www.asp.net/ moderator.

You can ask him any .NET related question – he will be glad to answer – if he knows the answer. If not, he will learn.

Daniel Tila - is a systems engineer master degree in system architecture and 10+ years of experience. I worked on airplane engine software, train security doors to bank payment and printing systems.

I am a certified teaching UiPath Developer & Architect and co-founder at Automation Pill where we help companies to optimize their costs by automating repetitive, mundane tasks. Holding training courses has always attracted me and I really enjoy a curious audience to whom I can interact with

Chapter 1

The boss that is hurrying up

True story: one day my boss came and he asked me to discuss how can we help our colleagues from the financial department since they are thinking about the automation process. We found a spare place and scheduled a meeting where we find out that 3 employees were manually extracting the client information (name, identity information, address) and contract agreement details (fee, payment schedule) to an Excel file and saving the document in a protected area.

Exercise

Make an application that reads all word documents from a folder, parses, and outputs the contents to Excel. Create and make it work, the easy way. You can find sample data in the folder data in the GitHub project.

Technical analysis

After putting some brainstorming and analyzing the constraint we ended up developing a solution that reads from a folder share and continues the process without any human intervention. This would allow us to have all our employees to continue their work simultaneously with less effort than before. The final setup was:

- prototype solution in a console app
- use NPOI (https://github.com/dotnetcore/NPOI) library for handling Word document reading. The decision is simply based on preference. However ClosedXML (https://github.com/ClosedXML/ClosedXML) is another valid option too. Our main focus was to remove any dependency of Microsoft Office, so our program to not require to have it installed while running. Limiting the dependencies is generally a good approach because it makes the testing easier.

We wrapped everything in the same project, just to make sure it works and to keep our focus on delivering. Everything we put in the main file of the console app, and we have chosen C# because we were the most comfortable and were able to fulfill our requirements in a minimum amount of time.

Code

```
string folderWithWordDocs = Console.ReadLine();

//omitted code for clarity
foreach (string file in Directory.GetFiles(folderWithWordDocs, "*.docx"))
{
    //omitted code for clarity
    var contractorDetails = ExactContractorDetails(document.Tables[0]);
    allContractors.Add(contractorDetails);
}
```

We were facing resolving the solution just with a list of sequences of operations without any control flow involved. Great!

Download code

Code at Chapter01 files:1;lines:66 (https://ignatandrei.github.io/console_to_saas/sources/Chapter01.zip)

Further reading

How to create a menu for the console app - https://github.com/migueldeicaza/gui.cs

When reading files on the hard disk, you should understand if you want to enumerate all files or just those that match a path. Read about the difference between IEnumerable and array (e.g. Directory.EnumerateFiles vs Directory.GetFiles) - https://www.codeproject.com/Articles/832189/List-vs-IEnumerable-vs-IQueryable-vs-ICollection-v

Chapter 2

Adapt to the financial department last tweaks

We installed our console app on one of the computers of an employee in the financial department, we set it up and we let it run. After a few days, we had some bug fixing and some new requests that came in. We saw an enthusiastic behavior on our colleagues by using our solution and that was our first real feedback that we were resolving a customer pain. We anticipated that our product will not be perfect (so more feedback will be considered) and to do tracking of what we did, we set up needed a source control. After a few iterations, we had our financial department happy and we are ready with our first MVP.

Problem

Try to refactor the solution from Chapter 1 to extract the business logic (i.e. transforming Word files into an Excel file) into a separate file.

Technical analysis

As a starting point, we decided to go beyond a separate file, hence we ended up to have 2 separate projects:

- a business logic (that can contain the core logic)
- the host (in our case the console app)

We decide to have this approach because it keeps a well balanced effort between value and time, and allows the flexibility to ensure [separation of concerns (https://en.wikipedia.org/wiki/Separation_of_concerns)](https://en.wikipedia.org/wiki/Separation_of_concerns).
We went with creating the **WordContractExtractor** class, where we handle all the Word related file formats. Here we moved the usage of the NPOI library and we make sure that the expected format is according to our requirements.

Hence, our solution looks now like:

- all the file conversion resides in a single place
- the main program file is more simple

This is a simple and long term solution because it is not a time-waster and offers us the possibility to adapt later to possible changes. Moreover using this technique early in the process, helps to isolate the problem better. You have less code to look in one place, and the problems are isolated.

To help the versioning we added a source control. We have chosen GIT since it is simple and allows us to work without an internet connection.

Code

```
var wordExtractor = new WordContractExtractor(folderWithWordDocs);
wordExtractor.ExtractToFile(excelResultsFile);
```

Now, all we need to do is to use the class and pass the needed parameters. In this case, the file path is the only dependent requirement. This path can come from a config file or from a user pick action. The only check we could add is to make sure it is writeable. We can add this check before executing the program or we can just handle separately the exception and display to the user a message. We will see that later in the next chapters.

Download code

Code at Chapter02 files:2;lines:86 (https://ignatandrei.github.io/console_to_saas/sources/Chapter02.zip)

Further reading

What is version control: https://en.wikipedia.org/wiki/Version_control

Create a free version control repository at http://github.com/, https://azure.microsoft.com/en-us/services/devops/, or other online source control.

Read more about how you improve the design of existing code by refactoring: https://www.amazon.com/Refactoring-Improving-Design-Existing-Code/dp/0201485672

Chapter 3

Second internal customer

Sharing code source + settings for client

It has been a few months since our financial department is using the application and they were happy with it. On a morning coffee break where we were discussing the new automation trends, and among them our tool was mentioned. The sales department was willing to see in action, since they had themselves some similar situation.

We configured and deployed on a separate machine for them and we saw great feedback and real usage on a daily basis. They were so excited to see how they decreased the paperwork hours and had more time to pay attention to the client's needs.
They want to use the application as an example of automation and we find it more suitable as product development.

Problem

Make a Console solution that can read a configurable setting (the folder path to be searched) from a file instead of being hardcoded into the application. Do not over-engineer the solution. Create and make it work, the easy way.

Technical analysis

Creating a configuration file with all the settings that a client can particularly change. For now, we have just the directory path, along with the config file. We have added a class that manages the settings operation and uset it in a typed format to have the config related settings in one place.

The reason that we put the setting into a configuration file and not make 2 applications is to have an easy way to maintain the same application for 2 clients. The impact on the solution is minimal. Just create a new class that knows how to read from a config file and retrieve the name of the folder as a property.

Code

```
var settings = Settings.From("app.json");
var directoryToSearch= settings.DocumentSLocation;
```

Download code

Code at Chapter03 files:3;lines:117
(https://ignatandrei.github.io/console_to_saas/sources/Chapter03.zip)

Further reading

How an application can have multiple configuration sources (environment, command line, configuration file, code - in this order) - Read about .NET Core endpoint configuration (https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-2.2)

Hot reload the configuration: read IOptionSnapshot and IOptionMonitor at https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options?view=aspnetcore-3.1

Chapter 4

Transform to product

Making a Graphical User Interface

At our Digital Transformation Center Department, we have recurring meetings with clients that are part of our continuous iterative process to identify things that need to be improved. In one of the meetups that we hosted on our desk, we invited the financial department to leave feedback on the process that we helped. We talked about how the process evolved and how by adopting Digital Solutions, we decreased the manual work.

The presentation ended up with some Q&A and several clients they were interested to have the application installed. At that point, we realized that this is a missing point in the current market and we planned to develop a product around this. At this point, we had a validated idea by a consistend number of users and we decided to evolve based on the client's needs.

Selling the application to multiple clients means to have configurable the word extracting and location of the documents. We ended up creating a GUI to allow the user to configure himself with the automation parameters. This was a valuable feature for our users since most of them could be non-technical.
Transfer the ownership of the paths from the config file to a more appropriate user interface.

Problem

Modify the existing solution to support both Console and GUI.

Technical analysis

For both applications, we have the same functionality: reading files from the hard disk and reading settings. We do not want to copy-paste code and become un-maintainable, so we want to share code source from a console application to the GUI application.

Because the user works now with network folders that cannot be available always, we need to perform some modifications to alert the user for problems. In our case we have the user interface for showing:

- the progress of the operations
- adding logging for knowing what happens if an error occurs
- adding exception handling to not crash the application
- handle edge cases of the user saving configuration (assert that user have not entered any folder, list what documents have been successfully processed, etc)

From a solution with a single project, we have now a solution with 3 projects:

- Business Layer (BL) - the logic for the application - in our case, processing documents from

a folder
- Console (referencing BL)
- Desktop (referencing BL)

The BL has the code that was previously in the console application (loading settings, doing extraction) + some new code (logging, others).

Code

The business logic is now more simple:

```
public void ExtractToFile(string excelFileOutput)
{
string[] files = Directory.GetFiles(_documentLocation, "*.docx");
_logger.Info($"processing {files.Length} word documents");
//code omitted for brevity
```

The Console code now is very simple, just calling code from BL:

```
var settings = Settings.From("app.json");
var extractor = new WordContractExtractor(settings.DocumentsLocation);
extractor.Start();
```

The Desktop is calling the same BL, with some increased feedback for the user:

```
try
{
    string folderWithWordDocs = folderPath.Text;
    if (string.IsNullOrWhiteSpace(folderWithWordDocs))
    {
        MessageBox.Show("please choose a folder");
        return;
    }
    string excelResultsFile = "Contractors.xlsx";
    var wordExtractor = new WordContractExtractor(folderWithWordDocs);
    wordExtractor.ExtractToFile(excelResultsFile);
}
catch(Exception ex)
{
    _logger.Error(ex,$"exception in  {nameof(StartButton_Click)}");
    MessageBox.Show("an error occured. See the log file for details");
}
```

Download code

Code at Chapter04 files:6;lines:287 (https://ignatandrei.github.io/console_to_saas/sources/Chapter04.zip)

Further reading

.NET Core 3.0 Windows application (WPF, WinForms): https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-core-3-0

MultiTier architecture: https://en.wikipedia.org/wiki/Multitier_architecture

Chapter 5

The first real client

Componentization / Testing / Refactoring

A client who was interested mentioned that they were using zip files for backup the documents for company service. They were interested in having our product, but we didn't support zip files. We shared our vision to create a highly customizable product, so implementing reading files from zipping would be an additional feature. He agreed to pay for it and we started the product development.

Problem

Starting from the solution, add support for reading either from a folder, either from a zip file.

Technical analysis

Changing the business core requires refactoring, which usually involves regression testing. Before every refactoring is started, we need to define the supported test scenarios from the business perspective. This is a list that usually includes inputs, appropriate action, and the expected output. The easiest way to achieve this is to add unit tests which do the checking automatically.

Altering pretty much the entire core adds some regression risks. To compensate for that, we will add tests (unit and/or integration and/or component and/or system) which verifies the already old behavior and the new one. The tests will make sure that our core (reading and parsing) outputs the expected output using various file systems. We will add a new project for unit/component test and we will reference the business core project and inject different components (in our case, file system).
This core modification will also impact other projects, including GUI and console applications.
In our case, we want to make sure that the contract parsing success if we use both file systems.

We already identified the operations that need to be abstract, so let's write the interfaces that allow this:

1. An interface for the file system which supports listing files (IFileSystem)
2. An interface with the actual file. This needs to have a Name and to read the content from either a zip file, either a folder into the system (IFile).

Our core program needs to:

1. find Word documents
2. extract information from it

These two operations are part of the actual business flow. We need to abstract the finding operation and reading from a file operation and allow different implementations.

Code

One implementation is from the actual file system (an existing directory) and the other one from the zip archive, hence:

1. DirectoryFileSystem
2. ZipFileSystem

A zip file can have inside folders and files, similar to a directory. Hence, we can do the same operations just like in the actual file system (listing files and reading a file from it).

```
var settings = Settings.From("app.json");
var fileSystem = settings.FileSystemProvider.CurrentFileSystem();
var extractor = new WordContractExtractor(fileSystem);
extractor.Start();
```

Download code

Code at Chapter05 files:12;lines:475 (https://ignatandrei.github.io/console_to_saas/sources/Chapter05.zip)

Homework

There are a few issues that you could encounter to transform this code example in production. The **WordContractExtractor** depends on the IFileSystem. The latter one is the state (it keeps the source files). While it is ok to have a state which is read-only (multiple requests will not alter the state), you could have concurrency problems if the files are written there. In this case, the IFileSystem has only methods for reading. Your homework is to modify the WordContractExtractor to be consistent and use the IFileSystem for writing.

Your task is to:

- add a function for writing in the IFileSystem that receives a Stream and it writes the content to the file system.
- add a property CanWrite to the IFileSystem and ensure writing is possible
- implement in the classes that IFileSystems inherits from the functionality

Further reading

1. Choosing Between an Interface and an Abstract Class - https://medium.com/better-programming/choosing-between-interface-and-abstract-class-7a078551b914
2. How to manage external connections using IDisposable - https://docs.microsoft.com/en-us/dotnet/api/system.idisposable
3. Read about file system abstraction (exists already: https://github.com/System-IO-Abstractions/System.IO.Abstractions
4. How to unit test http://dontcodetired.com/blog/post/Unit-Testing-C-File-Access-Code-with-SystemIOAbstractions)

Chapter 6

Our second external client

Ecosystem / Versioning / CI

Our second client has different types of contracts that need to be considered. Having 2 clients that need to run the same code, but to work differently it is a challenging task. Thus, we could address this problem using two approaches:

1. Create a different branch for each client
2. Have a single branch of code, but allow through configuration to work differently
   There is no perfect or easy solution, so let’s make a comparison and see which one fits better

Organizing source code

There are two approaches to this problem: create separate branches for each client or use a single branch and make the product behave differently.

Create different branches

So, we have our main code in master, and we will create a separate branch for each client. This is an easy task, but let’s see what are the consequences:

Advantages

1. Creating a separate branch is no effort
2. Best way to separate behavior and replicate bugs
3. It is easy to implement a new different feature for each client

Problems

1. ~~If~~ When a bug appears, it should be replicated on every client. This could be done by either have a single master and replicating to each branch
2. writing the bug solving code on every branch
3. Same with a new feature that many clients want.
4. The master branch will not have defined behavior. It will contain all the features merged, thus we should treat it as a new client, but with all features on.

Create a single branch

Another solution is to have a single code branch, but this means to design the application to allow enabling/disabling features by configuration.

Advantages

1. Single branch to maintain
2. Fixing a bug => it is done in a single branch

Problems

1. Re-architecting the application to support different features per configuration.
2. Replicate the client behavior is more difficult
3. Create a feature requires more time to integrate with the configuration system
4. Components per client
5. Incompatible clients requirements ;-)

Product client version

When a client reports a bug, we must know the code that was compiled to be distributed to the client. For this, we should somehow mark the source code before the application distributable to be built - and this can be done by tagging or by branches in the code source.
Also, we can have the application reporting the version of the components.

Further reading

1. To organize to on single source code for multiple clients and different needs, please read Strategy Design Pattern - https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm . Also you may use Plugin architecture - each client is a different plugin into the application .
2. Versioning - show the production version. You can use Semantic Versioning or Calendar Versioning. https://sachabarbs.wordpress.com/2020/02/23/net-core-standard-auto-incrementing-versioning/
3. For Auto-Update application please read how it is done with Click Once, Electron.
4. For deploying / download the whole application please read .NET Core 3 Self Container Application per platform: https://docs.microsoft.com/en-us/dotnet/core/deploying/

Chapter 07

Switching to web - SAAS product

Dealing with multiple clients adds complexity that must be handled, and there are several challenges:

- Keep track of the configuration/version of the app of each client
- Deliver different patches for the different clients.
  Besides this, we want to have a central part where we want to see how clients are using the app, so some analytics were needed.
  Customer issues with the system where it is installed the application (like OS incompatibility, antivirus, dll problems, or others) must be solved.

These challenges we could tackle easier if we could have a central point. Moreover, transforming this project into a SAAS could allow the activation of a subscription model for each individual client.

Problem

Create a website that allows us to import a zip file with multiple documents and output the resulting Excel.

Technical analysis

Create a new web project and use the existing functionality in the `ContractExtractor` project. The application must be converted to a web application (eventually, a mobile application should be ready also).
The web application should allow manage multiple client and be [multi-tenant (https://en.wikipedia.org/wiki/Multitenancy)](https://en.wikipedia.org/wiki/Multitenancy) enabled.

An authentication mechanism should allow having one or more ways to identify users (either username + password, either via a third provider, like Google / Facebook, either via an integrated provider, like Azure Active Directory / Okta). If you want to know how you can setup such a machanism, please read:

1. Identity Server : https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/identity-server
2. 2FA : https://docs.microsoft.com/en-us/aspnet/core/security/authentication/2fa
3. Authentication / Authorization in .NET Core: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/

The zip files can be uploaded and processed on the web. But, for the local files, the architecture should be somehow different: You may think about an agent that works on local, monitors the local file system, and uploads data to the web to be processed. This will allow us to have access to local resources and processes on the web.

Code

The most code will belong to the HTML and the backend infrastructure.

```
<div class="text-center">
    <h1 class="display-4">Welcome to extractor !</h1>
    Please upload your zip file with documents
    <form asp-controller="Home" asp-action="UploadFile" method="post"
          enctype="multipart/form-data">


        <input type="file" name="file" id="fileToSubmit" />

        <input type="submit" value="Generate Summary" />

    </form>
</div>
```

The business layer code , if well written, will be mostly the same

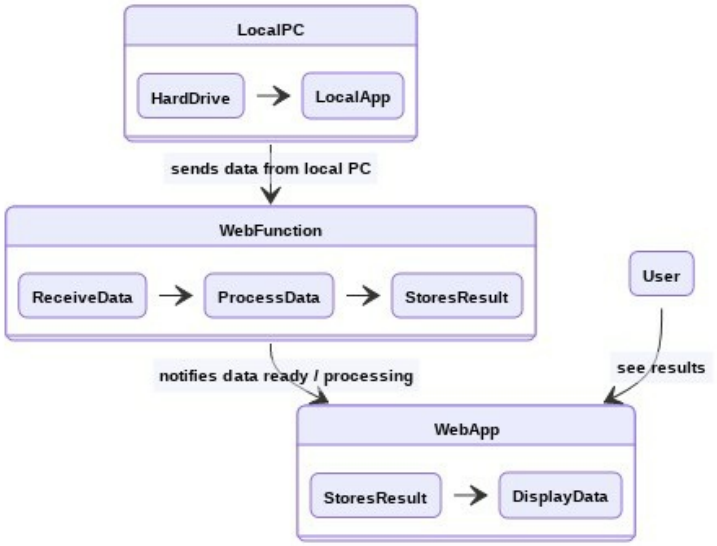
```
var fullPathFile =
Path.Combine(_environment.WebRootPath,Path.GetFileName(file.FileName));
if (System.IO.File.Exists(fullPathFile))
    System.IO.File.Delete(fullPathFile);

using (var stream = new FileStream(fullPathFile, FileMode.Create))
{
    await file.CopyToAsync(stream);
}
var fileSystem = new ZipFileSystem(fullPathFile);
var extractor = new WordContractExtractor(fileSystem);
extractor.Start();
```

Now, all we need to do is to use the class and pass the needed parameters. In this case, the file path is the only dependent requirement. This path can come from a config file or from a user pick action. The only check we could add is to make sure it is writeable. We can add this check before executing the program or we can just handle separately the exception and display to the user a message. We will see that later in the next chapters.

Download code

Code at Chapter07 files:36;lines:34378 (https://ignatandrei.github.io/console_to_saas/sources/Chapter07.zip)

Possible architecture

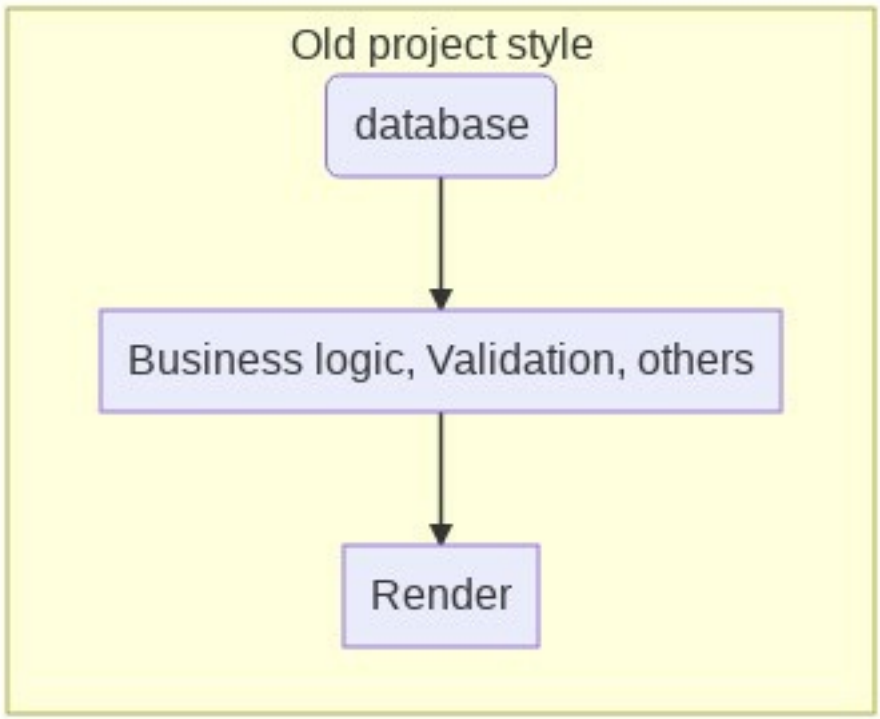


(https://mermaid-js.github.io/mermaid-live-editor/#/edit/eyJjb2RlIjoic3RhdGVEaWFncmFtIFxuICAgIFxuICAgXG4gICAgTG9jYWxQQyAtLT4gV2ViF
IERpc3BsYXlEYXRhXG4gICAgfVxuXG5cdFxuXHRcdFx0XHRcdFx0XHRcdFx0XHRcIsIm1lcm1haWQiOnsidGhlbWUiOiJkZ

Decoupling GUI from backend - allow Mobile, Local, Web to work with same endpoints

A web project is composed of two parts:
The view part (which is what the user sees)
The backend (which prepares the data and make possible the actions)
These two are implemented and rendered at different levels: the view part is rendered in the user browser and can contain any logic that makes sense here (simplest validation, animations), and the backend is what resides on the webserver and processes the request(security, validations, others) from the user.

Previously, we had the same parts in the same project, and the flow of the data it was looking like in this diagram:



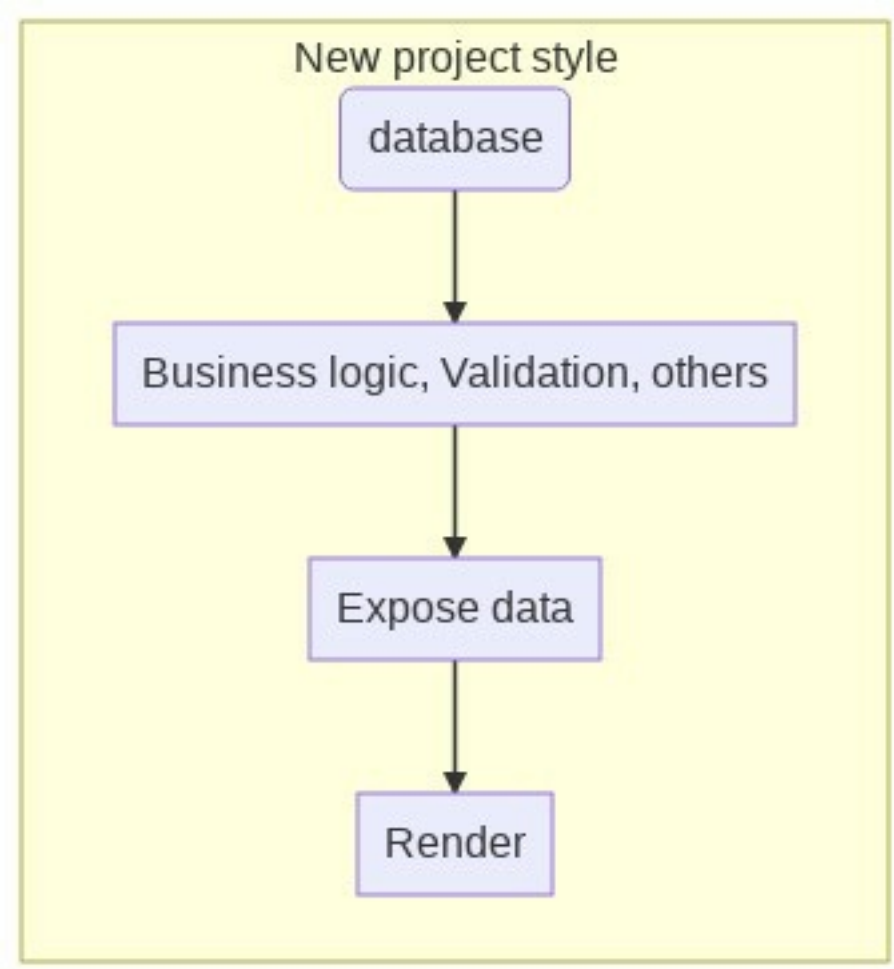
(https://mermaid-

js.github.io/mermaid-live-

editor/#/edit/eyJjb2RlIjoiZ3JhcGggVERcbiBzdWJncmFwaCBsZWdhY3lWaWV3QXBwW09sZCBwcm9qZ

UmVuZGVyMltSZW5kZXJdXG4gICAgZW5kIiwibWVybWFpZCI6eyJ0aGVtZSI6ImRlZmF1bHQiLCJ0aGV

A modern SAAS project has at least 2 projects, and it splits the 2 parts in 2 different projects:
one for data retrieval (the backend)
displays the data on the various GUI (browser, desktop, mobile - frontend).
This is not mandatory, but it’s a powerful mechanism to have smooth and responsive applications.
In this case, the diagram looks like:

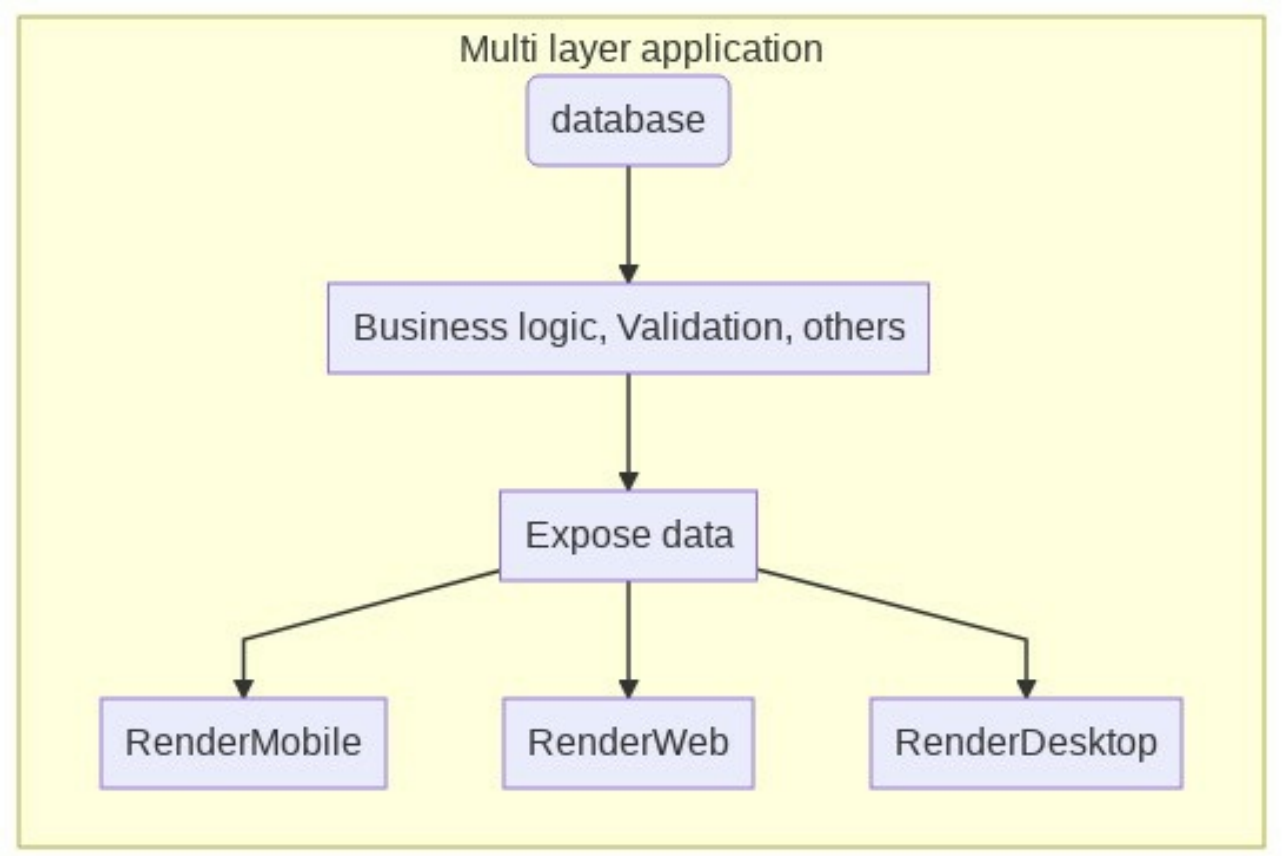


(https://mermaid-

js.github.io/mermaid-live-editor/#/edit/eyJjb2RlIjoiXG5ncmFwaCBURFxuICAgc3ViZ3JhcGggc2VwYXJhdGVQcm9qZWN0c0FwcFQnVzaW5lc3NMb2dpYzFbQnVzaW5lc3MgbG9naWMsIFZhbGlkYXRpb24sIG90aGVyc11cbiAgICBCdXMgVuZGVyMVtSZW5kZXJdXG4gICAgZW5kIiwibWVybWFpZCI6eyJ0aGVtZSI6ImRlZmF1bHQiLCJ0aGV

Moreover, by having this approach, the team members can be split also and the expertise is increased.

A big advantage of keeping the project separate, is that we can link another source that renders the data and displays in a different source: mobile projects



(https://mermaid-js.github.io/mermaid-live-editor/#/edit/eyJjb2RlIjoiXG5ncmFwaCBURFxuICAgc3ViZ3JhcGggbXVsdGlMYXllckFwcFtNdWx0aSBsYUmVuZGVyV2ViXG4gICAgRXhwb3NlRGF0YS0tPlJlbmRlckRlc2t0b3BcbiAgICBlbmRcblxuXG5cblxuXG5cblxuXG5cblxuXG5cblxuXG5

Multi-tenant vs Single-tenant

An application is multi-tenant if it can serve multiple clients at the same time. A single tenant can only serve one client. Modern applications are designed as multi-tenant because they are easier to maintain and scale. For all the advantages and disadvantages please see this link - https://en.wikipedia.org/wiki/Multitenancy.
The choice between these will affect at least the construction of the website and the database structure.

Microservices

The current project structure goes to a monolithic approach: you enhance a web application rich enough to perform multiple tasks. This will be hard to handle multiple clients since you need to have one single server for all the requests. This scaling is named vertical scaling and it is the one way of application operating modes. You can monitor the load of the server and make multiple different servers assigned to different clients (multi-tenant).
The new approach is to scale vertically: you split the application by features (named microservices), and each of these services runs on a smaller machine. It is easy to scale by features (individual

microservice) from the application perspective (one feature could be more heavily used than others), and from the cloud hosting perspective.
How do you split
There is not the best practice to split, it is depending on the use case, but there are 2 popular splitting approaches

- by domain (you group entities that are meant for the same thing. Ex: Orders, Customers, Payment, etc)
- by functionality (OCR, parsing of text, web crawling, etc)
  In our application, we could use some microservices like Authentication and Contracts.
  There is a lot of theory behind microservices with problems/benefits but it is out of the scope of this book.

However, we encourage to learn more by going through a selection of resources:

- https://udidahan.com/?blog=true
- https://www.youtube.com/watch?v=sSm2dRarhPo
- https://docs.microsoft.com/en-us/azure/architecture/patterns/choreography
- https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga

Exposing events to other apps

You may need to link with other services when an event in your application happens (contract is parsed, contract summary is created). This is a problem usually resolved by adopting an event bus architecture (see microservices). While this is not the best way to solve this problem production-ready (it has a lot of security breaches), you could create a service that exposes just some of these events.
One functionality that you may like to implement is to hook to a service which creates a notification when your contract has been created. There are many cloud providers that allow this like IFTTT or Zapier.
One implementation method is using WebHooks and for .NET please see https://docs.microsoft.com/en-us/aspnet/webhooks/

Continuous Integration/Continuous Delivery

Having continuous integration helps to minimize the time to discover code errors. This is useful when the team has any size since you can establish in the integration systems some quality controls rules that need to be passed after each developer does a code modification. The typical steps in a CI system are:

- ensuring application is building successfully
- ensuring tests pass successfully (unit tests, integration tests)
- create the deployment package for the application
- deploy on a UAT environment (continuous delivery)

There are many CI/CD systems (that you can run on your computer or in the cloud) such as:

- https://docs.microsoft.com/en-us/azure/devops-project/azure-devops-project-github
- https://github.com/features/actions

- https://www.jetbrains.com/teamcity/
- http://circleci.com/
- https://travis-ci.org/

Continuous delivery is usually done on a high-risk environment (UAT) and the deployment can be done after each commit of timed based (daily). For production deployment, there are many strategies on how you can do that and is really depending on the target quality and user dynamics. The most known deployment strategies are:

- https://martinfowler.com/bliki/BlueGreenDeployment.html
- https://martinfowler.com/articles/feature-toggles.html
- https://martinfowler.com/bliki/CanaryRelease.html

Conclusions

This book is trying to reproduce as real as possible a use case to get from a prototype to a software as a solution service. This story is not real but is based on our experience and difficulties that we had during this time. The book focuses on following a process and showing some tools to achieve the deliverable and not about implementing the best code practices.

There are more things to consider to have this kind of application in production. We can mention load testing, profiling, and optimization but they are out of the scope of this book.

While we did our best to make this book as useful as possible and if you think this can be improved, we would highly appreciate your feedback and accept pull requests on GitHub at https://github.com/ignatandrei/console_to_saas/actions.

We hoped you have enjoyed reading as much as we enjoyed writing. If you need help with your software implementations, you can reach us for consultancy services at:

- Daniel Tila - contact@dtila.ro
- Ignat Andrei - ignatandrei@yahoo.com

Good luck in your projects!