Automated Generation and Dynamic Rendering of Web-based Data Collection Systems

Ziyang Zhang
Royal Bank of Canada
Toronto, Canada
zhangzyster@gmail.com

Tsung Ting Chen
Royal Bank of Canada
Toronto, Canada
tsungtingjohn.chen@rbc.com

Kapil Vigneswaren
Royal Bank of Canada
Toronto, Canada
kapil.vigneswaren@rbc.com

Fatima Hussain
Royal Bank of Canada
Toronto, Canada
fatima.hussain@rbc.com

Salah Sharieh
Royal Bank of Canada
Toronto, Canada
salah.sharieh@rbc.com

Alexander Ferworn
Department of Computer Science, Ryerson University
Toronto, Canada
aferworn@ryerson.ca

***Abstract*—Development of web based data collection and resource management system is a time consuming, resource demanding process, as it contains multiple layers of back-end and front-end services. And the process is often prone to errors. Efforts have been made in research communities as well as in the software industry to automate this process. However there is no solution that produces a complete and ready-to-use system. In this paper we propose a novel approach to automate the development process with code generation and dynamic form rendering using the latest technology stacks available. By abstracting the data schema definition and configuration away from the actual implementation, code generation reduces development time dramatically, standardizes system architecture and is less prone to human errors. Our approach is also capable to generate a ready to use system including front-end and back-end components.**

***Index Terms*—automation, code generation, template, dynamic form rendering, web service, data collection system, resource management system**

I. Introduction

Almost every business process and online transaction involves collection of data with a predefined schema. The data collection form usually triggers some Create, Read, Update, Delete (CRUD) operations on some resources via web services [1]. For example, the open account operation provided in many mobile/online banking solutions involves a form where the users input their personal and financial information, after which the software validates data formatting, sends data to the bank's back-end services, and presents the user with a response. The users can also later view their existing accounts or make updates to them by using the same system. Such a data collection and resource management system usually consists of a User Interface (UI) that users interact with directly (which is often referred to as the front-end), a web service (often referred to as the back-end) that communicates with the UI, and a database behind the web service to store data.

To interact with end user, the front-end needs to run on user devices, presenting a UI either in the form of a Graphical User Interface (GUI) or Command Line Interface (CLI). A GUI can be in various forms but usually is in the form of a desktop or mobile application interface that is native to the target user platform. However, developers often need to provide multiple versions targeting all popular platforms. In recent years, web GUI have become more common compared to their counterpart , due to their ability to run in various browsers and to provide cross-platform accessibility.

A web service communicates with its clients via the Web, usually in the form of Remote Procedure Calls (RPC) or through Hypertext Transfer Protocol (HTTP) [1]. RESTful web service, which is an architecture for implementing web services with HTTP and JavaScript Object Notation (JSON), has become the de facto industry standard for web services due to its scalability in cloud environments and inherent capability of inter-operability with JavaScript-based front-end UIs [2].

Each data collection system has several features categorized on the basis of data collection, transmission and security. Cloud, portable devices (mobile phone) and Global Positioning System (GPS) are some of the technologies that are widely used to support data collection system. Paper-and-Pencil Interviewing (PAPI), Computer-Assisted Telephone Interviewing (CATI), Computer-Assisted Personal Interviewing (CAPI), and Computer-Assisted Web Interviewing (CAWI) are existing methods for data collection. CAPI is the most widely used choice. The integration of several technologies or methods increase the functionality on implementation. A data collection system also needs a database to store the data. Various solutions exist to meet different needs for data storage in terms of availability, data consistency, and data structure, etc. However, whichever database solution one chooses, the logic to interact with database always needs to be implemented.

The creation of such a data collection system involves development, testing and integration of these multiple components, therefore the system and its components are prone to human error. At the same time, a business may also need a number of data collection systems targeting various users and processeswhich results in a lot of repetitive developer work,

978-1-7281-2530-5/19/$31.00

and further leads to higher development cost, inconsistency, as well as complexity of system maintenance.

Automation could potentially solve the problems caused by repetitive work. The common parts in data collection systems, such as the architecture and interactions of components, the styling of RESTful APIs and UIs, can be put into a common place that only needs to be developed and maintained once. This paper presents our approach to automate the development of such systems through code generation and dynamic form rendering.

A. Related Work

There has been research done for the automation of business processes [3], in various perspectives. Data collection is considered as one of the most important parts in any business process, as it is used for making important decisions which are crucial for the overall performance of any business. In this spirit, [4] presents a systematic review of various data collection techniques and systems. [5] proposes and develops a tool to collect data from spreadsheets. This tool uses spreadsheet software as its GUI, and adds an integration facility to pipe data into SQL databases.

Code generation has been a widely used technique in the software industry [6] [7]. The CLI of Angular framework [8], for instance, uses code generation to initialize its TypeScript based front-end projects. In this spirit, [9] creates a template-based code generation framework in $C^{\#}$ to reduce development costs for back-end web services with an SQL database. They apply the framework to a shelf tracking system in a retail setup. Their framework reduced the development cost and also increased the code reuse capability and management capability of their system. However, this approach deals with back-end services only and lacks support for front-ends.

In [10], a code generating framework is proposed, that can generate source code from configuration files during the development phase. Configuration files are essential for software development, especially when development is on a large scale. The authors developed a framework, that is free of human errors and can generate type-safe codes from a configuration file. The authors of [11] propose an automated cross-platform GUI code generation framework. They utilize image processing and deep learning classification techniques for GUI code implementation between two mobile platforms (Andriod and iOS). Their framework takes UI pages of one platform as an input and outputs the GUI code for the other or target platform. This framework not only improves the efficiency of mobile development, but also shortens the development life cycle. The authors of [12] present a framework for automatic RTOS portability, by integrating model-based design into embedded software development. They focused on modeling the interaction between software and hardware for generating low-level code. This enables automatic portability for the hardware-related parts of the OS (i.e., context switching, memory management, security aspects, etc,) without the requirement of knowledge of the target architecture. Automatic code generation guarantees that the model is correctly translated to machine language, avoiding manual coding, which is usually prone to human errors.

OpenAPI (formerly known as Swagger) [13] is another area, where code generation has been intensively used to generate documentation, for test clients as well as back-end web services. The authors of [14] explored the possibility of generating front-end web UI with OpenAPI specifications.

B. Our Work

There are various challenges associated with the formation of data collection and resource management systems, including high development costs, low code reuse, and complexity of maintenance of multiple systems etc. To address all these challenges, we propose a framework of code generation and dynamic form rendering. Our framework has the ability to build an entire full stack solution including both back-end services and front-end UIs.

In this paper we demonstrate the proposed framework through a real world example in our business process. Our contributions are in 3 areas:

- the way we abstract the resource data schema into configuration file allows the framework to generate full stack solution, not limited to single component as reported in related work.
- we use the latest technology, such as Angular and GoLang to create the best performance and extendability to the framework.
- the integration of dynamic form rendering in the front-end further reduces the complexity of the code generation framework.

The rest of the paper is organized as follows: in section II, we present our model of the code generation framework with dynamic form rendering. In section III, we discuss and analyze our proposed framework in terms of complexity and efficiency. Conclusions are drawn in section IV.

II. Proposed Framework

In this section, we present our framework of code generation and dynamic form rendering. This section is organized as follows: in subsection II-A, we formulate the working example to illustrate the process. Afterwards, we present the abstractions for the configuration file in subsection II-B. Template based code generation and system architecture is covered in II-C, while subsection II-D presents the dynamic form rendering procedure (a specific technique to render forms in web GUI from a configuration).

The algorithm for our framework is as follows:

- Configuration File: Design the data schema.
- Code Generation: Invoke the code generation tool to generate source code for front-end Web UI, CLI, and the back-end web service.
- Build and Deploy: Build and deploy the generated components with necessary testing.

A. Example Use Case

Consider an enterprise comprised of several technology groups and running numerous processes. One of the business intake enables the ability of other technology groups to register their APIs with a central API registry gateway as a proxy. The services to be provided include the CRUD operations of their proxies.

We use the API registration platform as a use case to discuss our proposed framework. The operations of the Proxy CRUD service are illustrated in Table I. A user of this service will be able to create a new proxy, update an existing proxy with appropriate parameters, read (list) or delete proxies belonging to that user.

TABLE I: Operations in the Proxy CRUD Example

Operation	Parameters	Response
Create	Access Token Proxy Name Line of Business Target URL(s) Expected TPS*	(Created object)
Read	Access Token	(Object List)
Update	(Same as Create)	(Updated object)
Delete	Access Token Proxy Name	

*TPS: Transaction Per Second.

B. Configuration File

The configuration file contains the abstracted data schema and the environment setups of the system to be generated. YAML (Yet Another Markup Language) format is used for easy human readability. Listing 1 shows the configuration file for the example proxy CRUD service.

Listing 1: Configuration File Example

```
application:
  name: appname
  description: 'Managing your proxy
  in the API Registry'
  version: 1.0.0
resource:
  name: proxy
  pluralName: proxies
  schema:
    - property: name
      isIdentifier: true
      type: string
      pattern: ^[a-zA-Z][a-zA-Z0-9-]{0,31}$
      description: >-
        Name of the proxy.
        Should only contain letters, digits
        and hyphens. Maximum 32 characters.
    - property: lob
      type: string
      enum:
        - DepartmentX
        - TechnologyGroupY
        - BusinessTeamZ
      description: Line of business.
    - property: target
      type: string
      description: 'Target URL to proxy to.
      Should be a valid URL.'
    - property: tps
      type: integer
      minimum: 1
      description: 'Expected volume of the
      proxy in TPS (Transaction Per Second).'
environment:
  databaseType: SQL
  databaseURL: DB_URL
  databaseCredential: DB_USER_PASSWORD
  webserverHost: WEB_SERVER_HOST
```

Below, we describe various sections of the configuration file listed above.

- The *application* section defines the application name, version, and description. The *environment* section defines the environment variables the generated application will need at runtime.
- The *resource* section defines the data schema of the resource being managed. The data type definition is compatible with the OpenAPI specification [13] so that we can leverage the opensource tools of back-end server generation.
- The *isIdentifier: true* line will signal the code generation framework to use the *name* property as the unique identifier of the proxy resource. It will become the primary key in the generated database schema and also appear as a path parameter in the URL of the backend web service.

As shown in the example configuration file, we only define the resource data schema, without defining the operations. In this way we leave the formation of CRUD operations to the code generation framework so that they conform to the RESTful service standard no matter what data schema the resource has.

C. Template-based Code Generation

After defining the configuration file, it is fed to the code generator for the production of 3 components based on predefined templates, as illustrated in Figure 1. In our first implementation, the Go Template package, a standard library for Golang (Go Programming Language), is used as our templating engine. The code being generated is also in Golang for the back-end web server and the CLI, which will be compiled into binaries for deployment. For the web GUI, an Augular project is generated, which can then be built into static webpages.

The generated components are then built and deployed into a cloud environment, with a database component available (as defined in the configuration file). The runtime system architecture is illustrated in Figure 2.

The RESTful web server component stores data in a database and serves endpoints to manage the resource defined in configuration file. Table II shows the generated endpoints, while Listing 2 shows the generated SQL database schema for the example use case.

Listing 2: Generated Database Schema for the Example Use Case

```
CREATE TABLE IF NOT EXISTS apollo_proxies (
        name VARCHAR NOT NULL,
        lob VARCHAR NOT NULL,
        target VARCHAR,
```

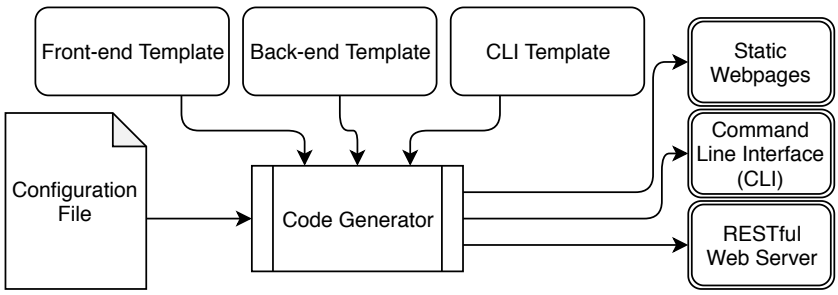


Fig. 1: Code Generation Process

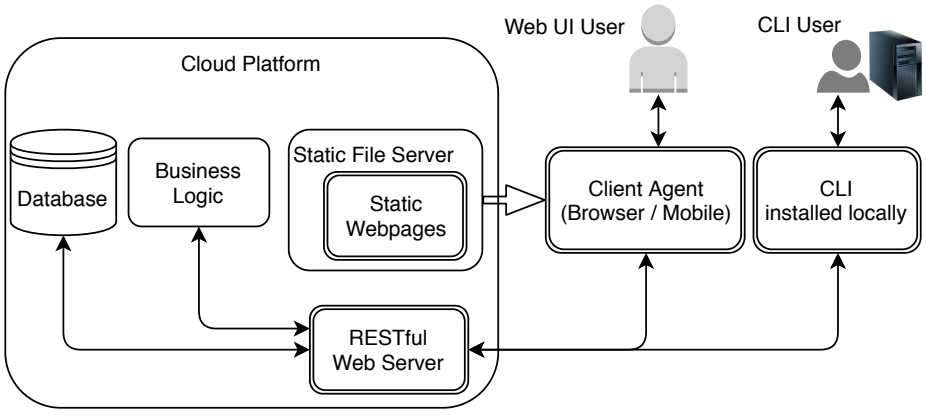


Fig. 2: Run-time Architecture of the Generated Systems

TABLE II: Generated Endpoints for the Example Use Case

Endpoint	Method	Operation
/appname/v1/proxies	POST	Create Proxy
	GET	Read Proxy List
/appname/v1/proxies/{name}	POST	Update Proxy
	DELETE	Delete Proxy

```
    tps INTEGER,
    createdBy VARCHAR,
    createdAt BIGINT,
    lastModifiedBy VARCHAR,
    lastModifiedAt BIGINT,
    PRIMARY KEY (name)
);
```

Note that necessary lifecycle metadata columns (*createdBy*, *createdAt*, *lastModifiedBy*, *lastModifiedAt*) are added to the database schema by the code generation framework.

Additional logic can be added manually in the generated web server so that whenever a resource is being created, modified or deleted, an event will emit that triggers some business logic processes in another service.

The generated CLI is a tool that can be deployed to the client side, to be used by either human users or automated scripts. Listing 3 shows the usage for the demo example. The CLI is generated so that it will take in parameters as command line options according to the defined data schema and send them through RESTful API calls to the generated web server.

Listing 3: Generated CLI Usage for the Proxy CRUD Example

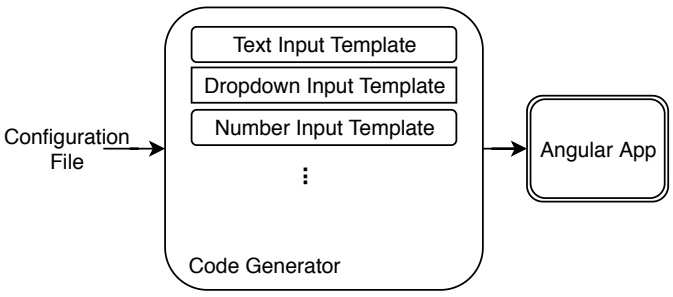
```
$appname --help
Usage: appname <Command> [options]
Commands:
proxy-create     Create proxy
  Options:
    --name   Name of the proxy.
    --lob    Line of business.
    --target Target URL to proxy to.
    --tps    Expected volume of the proxy in TPS (
        Transaction Per Second).
proxy-list       List proxies
proxy-delete     Delete proxy
  Options:
    --name   Name of the proxy.
proxy-update     Update proxy
```

Finally, the front-end web GUI is a static webpage package generated and built as an Angular project. Even though code templates greatly reduce the development cost, another layer of abstraction with dynamic form rendering in the front-end can further reduce the complexity of code generation. This will be illustrated in the next subsection.

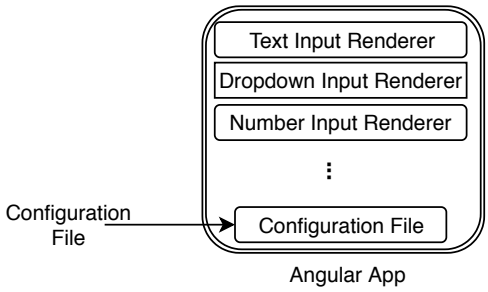
D. Dynamic Form Rendering with Angular for Front-end

A web GUI application usually contains source code files in multiple languages, for example in HTML and CSS for the content and layout of web page and in TypeScript or JavaScript for the controller logic. Front-end frameworks like Angular provide two-way data binding between the controller and the UI presentation.

The data binding in Angular is realized in the form of a HTML template that is linked to a property in the controller code. This would make the front-end template files in our



(a) Generating Web GUI without Dynamic Form Rendering



(b) Web GUI with Dynamic Form Rendering

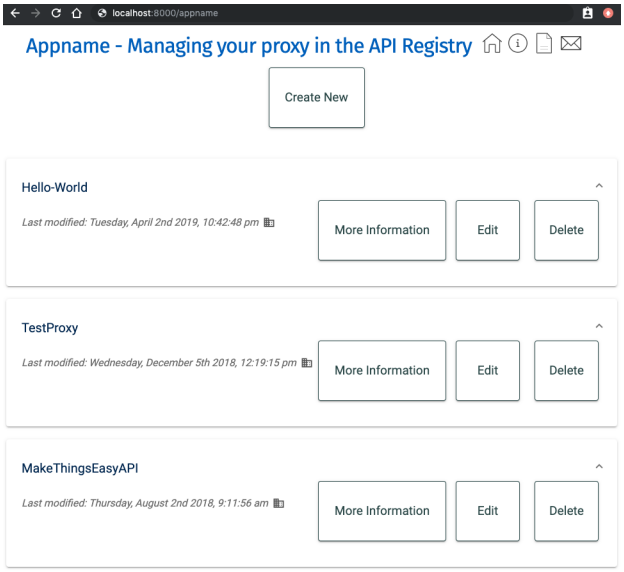
Fig. 3: Simplifying Web GUI with Dynamic Form Rendering

framework contain another layer of templating, resulting in difficulties in readability and maintainability of the template files.

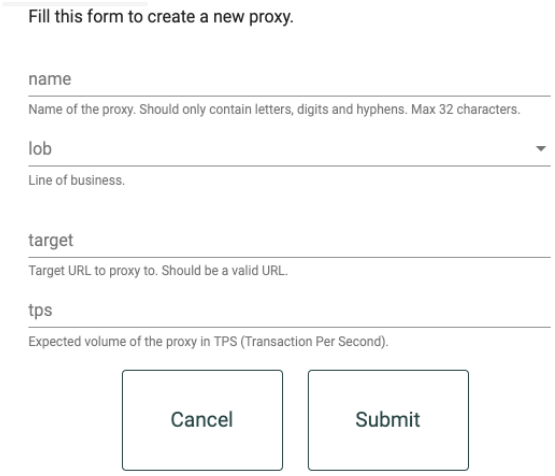
However, since the web GUI runs in a client browser, and is serving one user at a time, we can user dynamic form rendering to sacrifice some performance and achieve higher maintainability and a cleaner code base.

Figure 3 shows the simplification dynamic form rendering brings into the web GUI: the Angular app now is a standard app without the need for generating or templates and can be built separately. It loads the configuration file at run time and renders appropriate forms based on the data schema definition.

In our first implementation, we adopted the Angular Material package for the styling of forms. Figure 4 shows the final web GUI for our demo example.



(a) GUI for Listing Resource



(b) GUI for Creating New

Fig. 4: Rendered Web GUI for the Proxy CRUD Example

III. Results

In this paper a new framework is proposed to generate full stack web-based data collection and resource management applications. It uses template-based code generation and dynamic form rendering to abstract the data schema definition away from applications. A real world example in our business domain is used to illustrate how the proposed framework works, from defining the configuration file, generating the web server, CLI and web GUI components, to further simplifying web GUI architecture with dynamic form rendering.

The proposed framework significantly reduced the development complexity and cost of web based resource management systems. It makes the future development of similar systems a trivial work by changing the definitions in the configuration file. For example, by changing the resource type and data schema to those of a product catalog, a new system to manage product catalog can be easily generated and built.

IV. Future Work

Our first implementation took care of the core functionality of code generation and was used to demonstrate the benefit of our framework. However, many more features could be added in the future to further improve the framework.

One of the most obvious potential extensions is to add more supported data types. Our current implementation only recognize primitive data types, such as string, number and boolean type, to be used as the type for the data collection fields. It would be beneficial to support other types of data. For example, an image data type could be added to enable the input and display of images; an array type could also be added enable list of items of other primitive types.

The integration with identity and access management (IAM) systems is also a potential area of improvement. The configuration file could be extended to define ways to authenticate users, as well as the access controls on resource CRUD operations. This extension would increase the security level of the generated system and make it possible for use in real production environments.

Another area of possible future work is the abstraction of UI styling. Though with dynamic form rendering, it's fairly easy to change the web GUI styling, it still requires source code changes to the framework to adjust the look and feel of the UI. It would be more flexible if the styling can be controlled by a configuration file as well.

REFERENCES

[1] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs: Services for a Changing World*. O'Reilly Media, 2013.

[2] E. Cerami, *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*. O'Reilly Media, 2002.

[3] X. Guang-cai, W. Zhi-feng, Z. Xin-jia, and J. Guo-jun, "Realization of business process automation based on web services and ws-bpel," in *ICSSSM11*, June 2011, pp. 1–5.

[4] Miswar, Suhardi, and N. B. Kurniawan, "A systematic literature review on survey data collection system," in *2018 International Conference on Information Technology Systems and Innovation (ICITSI)*, Oct 2018, pp. 177–181.

[5] F. Nurdiantoro, Y. Asnar, and T. E. Widagdo, "The development of data collection tool on spreadsheet format," in *2017 International Conference on Data and Software Engineering (ICoDSE)*, Nov 2017, pp. 1–6.

[6] J. Arnoldus, M. van den Brand, A. Serebrenik, and J. Brunekreef, *Code Generation with Templates*, ser. Atlantis Studies in Computing. Atlantis Press, 2012. [Online]. Available: https://books.google.ca/books?id=UvC0MJHSqjkC

[7] M. Qadir, "Role of automation in computer-based systems."

[8] N. Murray, F. Coury, A. Lerner, and C. Taborda, *Ng-Book: The Complete Guide to Angular*. CreateSpace Independent Publishing Platform, 2018. [Online]. Available: https://books.google.ca/books?id=tI5UswEACAAJ

[9] K. Shinde and Y. Sun, "Template-based code generation framework for data-driven software development," in *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science Engineering (ACIT-CSII-BCD)*, Dec 2016, pp. 55–60.

[10] H. Puripunpinyo and M. H. Samadzadeh, "A framework for building type-safe configurations for jvm using code generation techniques," in *2017 Systems and Information Engineering Design Symposium (SIEDS)*, April 2017, pp. 50–55.

[11] S. Chen, L. Fan, T. Su, L. Ma, Y. Liu, and L. Xu, "Automated cross-platform gui code generation for mobile apps," in *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*, Feb 2019, pp. 13–16.

[12] R. M. Gomes and M. Baunach, "Code generation from formal models for automatic rtos portability," in *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Feb 2019, pp. 271–272.

[13] OpenAPI-Initiative, "The openapi specification." [Online]. Available: https://www.openapis.org/specification/repo

[14] I. Koren and R. Klamma, "The exploitation of openapi documentation for the generation of web frontends," in *Companion Proceedings of the The Web Conference 2018*, ser. WWW '18. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2018, pp. 781–787.