REST API Auto Generation: A Model-Based Approach

Conference Paper · September 2020

CITATIONS
0

READS
888

3 authors:

Salah Hussein
Berzeit University
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Samer Zein
Birzeit University
27 PUBLICATIONS **226** CITATIONS
SEE PROFILE

Norsaremah Salleh
International Islamic University Malaysia
74 PUBLICATIONS **1,786** CITATIONS
SEE PROFILE

March 2020

REST API Auto Generation: A Model-Based Approach

Salah HUSSEIN salah.hussein@gmail.com [a,1], Samer ZEIN szain@birzeit.edu [a]
Norsaremah SALLEH norsaremah@iium.edu.my [b]
[a] *Master Software Engineering, Birzeit University, Ramallah, Palestine*
[b] *Department of Computer Science, International Islamic University, Kuala Lumpur, Malaysia*

Abstract. Most of software products, especially mobile applications (apps) rely on a back-end web services to communicate with a shared data repository. Statistics have demonstrated exponential demand on web services, mainly REST, due to the continuous adoption of IoT (Internet of Things) and Cloud Computing. However, the development of back-end REST web services is not a trivial task, and can be intimidating even for seasoned developers. Despite the fact that there are several studies that focus on automatic generation of REST APIs, we argue that those approaches violate the rules of code flexibility and are not appropriate for novice developers. In this study, we present an approach and a framework, named RAAG (REST Api Auto-Generation), that aims to improve productivity by simplifying the development of REST web services. Our RAAG framework abstracts layers, where code generation has been avoided due its limitations. A preliminary evaluation shows that RAAG can significantly improves development productivity and is easy to operate even by novice developers.

Keywords. code generation, model-based, software framework, SOA, web services, and REST APIs.

1. Introduction

Software applications that rely on a back-end web services are increasingly popular. Software back-end development is a challenging area, where it needs significant efforts to properly handle thousands of simultaneous actions. For instance, data availability, integrity, confidentiality, and privacy must be handled precisely and on time. Accordingly, wide knowledge with strong competences in back-end services development is an essential qualifications to proceed in this area. Furthermore, large percent of developers, especially in the field of mobile apps are known to be novice, and many of them are coming from non-computing background [1], so they tends to use auxiliary tools and frameworks.

Recently, significant attention has been paid to software development techniques, to adapt and to handle the huge demand on software development, its requirements, and to deliver feasible business applications and solutions. REST (Representational State Trans-

[1]Corresponding Author: Salah Hussein, Birzeit University, PO Box 14, Birzeit, Ramallah, West Bank, Palestine; E-mail: salah.hussein@gmail.com.

fer) web services is the most common style used for applications architecture [2]. For instance, thousands of developed mobile apps available at online stores, require a back-end repository to store their shared data. Moreover, rapid development environment, short-time-to-market, and tough competition make it more challenging to find more productive and flexible techniques to develop such back-end services.

In recent years, cloud architecture has seen rapid adoption in most enterprises, as systems infrastructure, which has been a leading cause to reallocate backbone components and services for thousands of systems and applications. Backbone services of these systems have been migrated out of on-premise to remote hosts facilities, such as cloud services, where cloud architecture relies on SOA (Service Oriented Architecture) web architectural style. Further, the use of Internet of Things (IoT) applications increased the number of connections, where various aspects of life and several types of devices have been engaged with the Internet. REST style is the new common architectural style for web services, which made a tremendous change in the world of web APIs. Accordingly, most of the time-critical and safety domains in the world have been re-developed again according to the REST architecture, such as government, finance, payment systems, health, and military domains [3].

Creating REST APIs is a tedious, error-prone, and time-consuming task [4], where most frameworks assume familiarity of developers with the employed modeling technologies [5]. The study by [6] conducted a survey of errors produced by consuming web services indicated significant problems (60,000 API error responses per day). Another issue is that developers have to represent data models in object format, transform requests to SQL statements and vice versa to implement the business logic, and they have to de/serialize exchanged messages [7]. Further, the similarity of CRUD operations makes the development process tedious and error prone [8, 9]. Hence, there is a lot of repetitive coding due uniform interfaces [3], which increases gap between design and implementation [10].

According to prior work, several studies focused on rapid REST APIs development, and introduced their approaches, techniques, tools, and frameworks. For instance, code generation was the most significant approach in this field of research [8, 3, 12, 10, 5]. However, we argue that these approaches are not productive and are not well suited to be operated by novice developers [3]. In this research, we present a model-based approach and a frameworks named RAAG (REST Api Auto-Generation) that can automatically build REST APIs rapidly. We conducted a preliminary evaluation through an experiment, and the results show that our framework can significantly increase work productivity while introducing a more easy-to-use approach.

Our framework has the following contributions:

- Reduce development time for backbone services.
- Reduce time to learn or to understand the built software.
- Enable customization without restrictions.
- Avoid limitations of auto code generators.
- Utilize particular frameworks in a loosely coupled manner.

The next section introduces related work and addresses knowledge gap. Section 3 presents the proposed approach and provides an overview about research implementations. In section 4, represents evaluation through experiment and survey. Results are discussed and analyzed in section 5. Finally, the conclusion (section 6) summarizes the findings and recommends future work.

2. Related Work

2.1. Guidance Approaches

Existing studies in this area [13, 14, 15, 16, 4, 17, 18] proposed a guidance approach, based on model-driven engineering and advise a specific architecture over a set of utilities within a unified framework to orchestrate REST APIs. The approach proposed by Tavares and Vale [15] provides more abstraction via meta-model language for model transformation. Costal et al. [16] study goes further by semi-automating the APIs generation process. Used methods tends to guide for the most efficient and suitable technologies with certain sequence and context, over data mapping, business logic, and web-based APIs layers.

In the first layer, Jeon and Chung recommend the use of common frameworks, such as Grails ORM [17], while Costal et al. [16] proposed data models derived by resource entities and relations [16]. The second layer guides for middleware language, such as Groovy. A model-to-model transformation based on Eclipse plugins, and use data injectors to engage reusable libraries, which also guides to produce a single global ontology [16]. In third layer, REST APIs use JAX-RS (Java Architecture for XML to build RESTful Web Services) Grails with Spring security [17]. Jeon and Chung [17] introduced a process that requires manual edit for Groovy model code, JavaScript, server pages, and corresponds entity properties. Ed-douibi, et al. [4] used API composer that compose REST APIs as OpenAPI definitions based on EMF (Eclipse Model Framework) and on OData. Composer includes two components, an importer and resolver, but this approach is suitable for data retrieval only.

2.2. EMF Based Approaches

Approaches proposed by [7, 19] rely on EMF plugins to generate REST-based web APIs, and depend on common and standards libraries. All required artifacts is derived from UML, which semi-automate OData services out of relational database [7]. EMF-REST approach [19] generates APIs out of Ecore models by JET (Java Emitter Templates for Code Generation) and EGL (Epsilon Generation Language) templates for model-to-text transformations, (i) start in Maven project, (ii) extend EMF to cover JAXB (Java Architecture for XML Binding) with validation, (iii) use JET to get corresponding code for annotation of JAXB with OCL validation methods, (vi) use model-to-text transformations EGL that produce JAX-RS, CDI (Contexts and Dependency Injection), and EJB code. Ed-Douibi, et al. [7] used OData models over EMF to define EDM (Entity Data Model), SQL transformation, sterilizer, and formatter, a proofed his approach by building Eclipse plugin.

2.3. Model-Driven Code Generation

Studies by [8, 3, 12, 10, 5] apply a model-based approach and automatically generate code for common operations of database access, and then wrap it into RESTFul APIs, which reduces efforts and improves flexibility and reusability, such as model-to-model transformation in [3]. In general, designed approaches spread into three stages, (i) generate PSM (Platform Specific Model) with CURD operations, (ii) transform model-to-model/text/code, and (iii) generate reference API. Hibernate and Lucene ORM frame-

works were used [5]. In second stage, a model-to-model and model-to-text transformation were used. Fischer [3] used ETL (Epsilon Transformation Language) for model-to-model to get PIM (Platform Independent Model) and PSM. Zolotas, et al. [5] designed (Computationally Independent Model) CIM-to-PIM, PIM-to-PSM, and PSM to source code transformations. In the last stage, Zolotas, et al. [5] compressed Java, Hibernate/Lucene ORMs, JAXB (XML-Java), and JAX-RS over Jersey, all these are compressed in PSM models, but in most cases it needs manual coding. Terzić, et al. [12], da Cruz Gonçalves and Azevedo [8] used DSL (Domain Specific Language) over OpenAPI to provide architecture specification of REST micro-service.

Most of the proposed approaches introduced guidance only [13, 14, 15, 16, 4, 17, 18], and several ones used code generation [8, 3, 12, 10, 5], which negatively impacts development process with huge initial efforts, possibility of losing code, code rigidity, technical complexity, and restrictions for senior developers. Also most of them didn't empirically verify the effectiveness of the approaches except for few ones [8, 3, 10], where the evaluation of productivity and easy-to-learn are essential for PoC (Proof of Concept). Some studies only supports relational DB and other only for data retrieval [10, 4]. In general, related studies proposed solutions that are based on a very complex platforms and tools that require experience and longer time for training [20, 21, 22]. This study aims to simplify the automatic creation for REST APIs while avoiding code generation.

3. The Solution Approach

Our solution approach is based on integrated framework, which combines the most needed components and aspects into one place as an infrastructure platform. On the other hand, the framework architecture is designed to be loosely coupled with certain technologies, as will be explained, to achieve and realize below features and advantages:

- Compliance with Parallel Agile (PA) approach for parallel development among teams [23], which facilitates allocation of teamwork tasks into sprints.
- Compatibility with new trends of technologies and service platforms, such as Microservices architecture, Cloud services, NoSQL and Big Data platforms, where REST became the key interface that provides an infrastructure for IoT world.
- Centralism in this framework enables easy control to manage logging, security, and permissions.
- Realized unification enables architects to build and integrate unit test and also to automate testing on back-end level.
- Reusability for others' work in easy way, where the framework already integrated the most common and useful utilities and frameworks.
- Unify code style that can be implemented according to different developers' levels, which simplifies code for reading and understanding, since the applied abstraction technique coped complexity [24], whereas automation is the most defective method to boost productivity with quality.
- Minimize efforts to learn and simplify required experience and qualification to use complex frameworks, where novice developers can utilize these platforms indirectly over this framework.

- Introduce enough margin for senior developers to innovate their techniques and tools in parallel with utilizing RAAG framework without any conflicts or restrictions.

3.1. Technologies and Platforms

RAAG framework reused the following platforms and technologies in a unified and decoupled way. The framework integrated these packages as one infrastructure with plugins inside STS Eclipse IDE. Where Spring Framework provides a platform to develop Java applications, Hibernate ORM to map Java object with relational data entity, Jackson Object Mapper to serialize/de-serialize JSON to Java, Springsource Tool Suite (STS) is the preinstalled plugins on Eclipse IDE to provide supplementary features for Spring developers, Eclipse DTP (Data Tools Platform) to interact with any SQL database over Eclipse IDE, and Apache Maven as Eclipse plugin that manage different utilized tools and frameworks inside projects. JBoss Hibernate Tools technologies is a group of plugins inside Eclipse, such as Hibernate, JPA, Apache Camel, Maven, (X)HTML, CDI, JSF, JBoss AS, OpenShift, Docker, Red Hat JBoss Fuse.

3.2. Integration with Eclipse IDE

In this solution, Spring framework features are integrated into Eclipse IDE through STS, hence developer can simply use Spring feature and focus on application logic. DTP is also plugged into Eclipse IDE to integrate SQL database, and then it facilitates data management from the environment side. Moreover, DTP offers a database connection for JPA tool to fetch schema meta-data for the intended entities. Where JPA tool is a JBoss Hibernate Tool that used to generate data model from database entities, then model can be used from both sides, Hibernate ORM and Spring business logic. On the other hand, Jackson object mapper used over REST APIs to marshal and de-marshal data Model into JSON syntax object.

3.3. Architecture and Design

One of the most important phases in software development life cycle is the design, it impacts other phases, and it influences support alongside with maintainability, quality, and performance. Design quality depends on developer's experience and expertise only. However, RAAG framework applies principles, standards, and patterns, which can improve the re-usability, maintainability, and scalability of software applications.

Framework architecture consists of three main aspects: (i) data source, (ii) core framework, and (iii) data services with their model. In general, these three aspects are based on Spring Framework version 5 as a platform that abstracts all infrastructure implementations, as shown in figure 1.

Firstly, the data source aspect is responsible for handling data from its marts, this part is based on Object relational Mapping (ORM) framework to parse and marshal data entities into Java objects, where Hibernate framework is used to manage these roles. ORM functionality and database engine can be replaced with any equivalent tools, or it may be developed from scratch if needed. Therefore, data source is loosely coupled with core aspect, which realize an extreme flexibility, where dependency injection (DI) of Spring is applied as aspect oriented architecture.

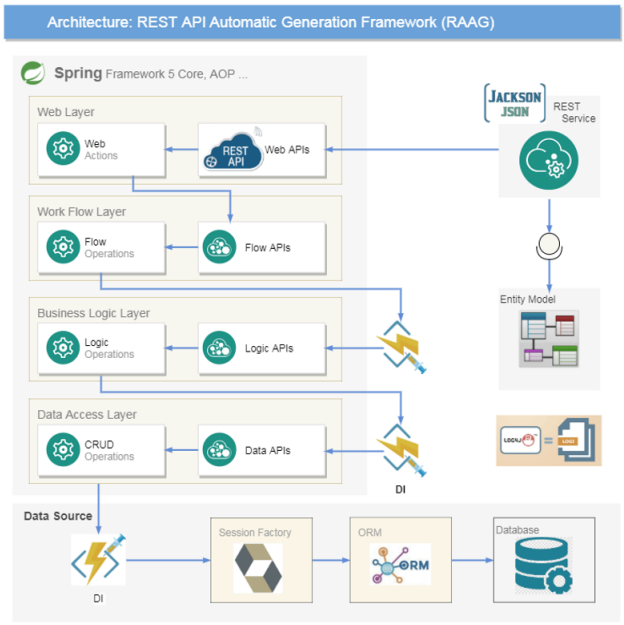


Figure 1. Framework Architecture (RAAG).

Secondly, the core aspect is based on four layers architecture: layers realize separation of concerns into data access, business logic, work flow, and web APIs. Each layer services the above one, and it is separated with Spring dependency injection DI to enable customization and replacement with different layer or layers. Finally, on top of web APIs tier, module of "data services and model" is designed to handle common web APIs over each entity model, then it can expose common APIs as-is, or override, customize, or expand them based on system and software requirements. This architecture enables to abstract database CRUD operations at least (Create, Retrieve, Update, and Delete).

Accordingly, this architecture can speed REST APIs generation and reduce their development time and required efforts. Where the most common operations (e.g. CRUD) were encapsulated inside the designed layers, and also any more ones can be embedded on app or framework level, which hides code complexity and avoids repetitive code. Moreover, it achieves many constraints and non-functional requirements, such as flexibility, agility, simplicity, reusability, maintainability, and other more values that mentioned in section **??**. Efficiency of this methodology and its impact on the process of software life-cycle have been investigated and discussed, as shown in section 4.3 and section 5.2.

It is also important to mention that the introduced framework is aligned with SOLID principles [25, 26] that lead to high quality of software according to the empirical validation of Singh and Hassan [27].

3.4. Solution Procedure

The main objective of our solution is to enable developers to build and maintain REST web services in a simple way without impacting the code quality. Using our framework, the following steps summarize the procedure to build REST APIs in a relatively short time:

1. Setup the environment as mentioned in section 3.1 over STS Eclipse IDE while using Maven project.
2. Integrate and configure the auxiliary tools and plugins to support the prepared IDE, which required to be done for one time only, as discussed in the integration section 3.2, where ORM Hibernate support DB interactions and data tool with JBoss Hibernate tools help to generate data model.
3. Generate model source code by using the integrated tools in IDE, where it can be used to connect with database and then generate model classes for the selected tables.
4. Implement abstract methods in data model to be compatible and plug-able into *RAAG* framework, these methods are empty or very simple in default.
5. Extend Web APIs layer, which discussed in section 3.3, where SpringWebAPI can be extended to create the required service based on certain model object/s, default empty service class inherits all the common predefined APIs, such as CRUD and search APIs.
6. Finally, implementer can customize or maintain the created service by adding to the existing functionality or overriding the predefined APIs, which also can be expanded inside framework to include an other common functions.

RAAG framework design implicitly fulfills the following objectives:

- *Avoid restrictions for senior developers:* based on this architecture, seniors can expand the already predefined REST APIs (CRUD), and meanwhile they can utilize *RAAG* capabilities, also can replace complete layers by Aspect Oriented Programming.
- *Avoid downsides of auto code generators: RAAG* applies polymorphism without any code generation (model can be generated, it contains values without logic).
- *Enable loosely couple for other frameworks:* a complete aspects can be replaced without any impact on the built APIs, such as data source, or layers.

4. Evaluation

We have conducted a preliminary developer evaluation of RAAG for REST APIs development. Our primary aims in this user evaluation is to evaluate developer acceptance and examine how using RAAG can speed up REST APIs development for novice and experienced software developers.

4.1. Experiment Setup and Tasks

Experiment setup can be described in the below steps:

1. Tasks were carefully designed to be adequate and short as much as possible.
2. Prepared a training material as short movie as well as brief document that explains experiment procedure.
3. Setup lab instruments with virtual machine (VM ware) to encapsulate environment (OS, DB, and IDEs) to keep similarity.
4. Prepared a short survey to measure participants satisfaction and expectation.

5. Recruited participants carefully, and then scheduled sessions for two groups, Group 1 and Group 2 (G1 & G2 hereinafter), G1 used proposed approach and G2 used common used approaches.
6. Experiment Sessions started by showing training movie.
7. G1 participants were asked to apply RAAG framework solution procedure, as discussed in section 3.4.
8. Progress and time was monitored using stopwatch instrument.
9. Data was collected and analyzed using MS-Excel.

Training material was prepared to create awareness about the solution technique. Thereby, a short 15 minutes movie was uploaded on YouTube [2] to offer a complete, fast, parallel, similar, and sustainable material for each participant. Moreover, Java source code and Java API documentations were packaged and provided for participants. Environment setup was encapsulated in VM ware instance to keep similarity and portability for whole platforms with setting, such as OS, DB, and IDE with plugins.

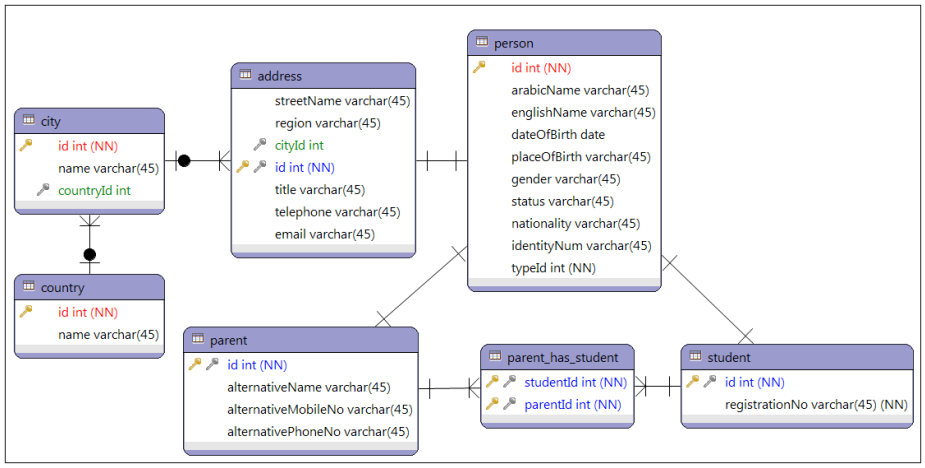


Figure 2. ER diagram for experiment exercise.

According to experiment design principals, specific exercise should be designed as a comparable task between the two groups of participants, the prerequisite knowledge should already be transferred to them through the prepared material. The experiment assignment was a Student model that consists of a Person, Student, Parent, Address, Country, and City entities. Model exercise was structured to cover all possible relations between entities, as shown in Figure 2, building a comprehensive exercise is a very important task to measure valuable indicators between groups. For instance, Student is a detailed data entity, which joint with Person entity as master composition in a one-to-one relation, also Student refers to Parent through many-to-many breakdown entity. From UML class diagram perspective, Person is a super class for Student child, which contains list of Parent objects and vice versa.

[2]Training movie uploaded here on YouTube

4.2. Participants

A specific set of tasks was assigned to all participants in each group, as detailed out in section 4.1. The collected data was recorded into the excel sheets. Participants were allocated into two groups, each group includes diverse individuals in terms of years of work experience.

The first group (G1) involved seven participants who were assigned to use RAAG framework to develop the required Student model and REST web service according section 4.1. Three participants of this group are fresh graduate engineers (novice), they worked as trainees for few weeks in local company. The other three engineers have an experience of more than five years, the last participant has one year of experience.

The second group (G2) involved four experience participants. This group applied different frameworks, where each participant used his preferred framework according to his experience. A database dump was provided for each participant. The database dump precisely leads to structure the intended model for Student entity and the REST web service on top of that model, just like the exercise of a first group. Hence similar test procedure can be used to avoid threats of validity. Two developers of this group are experts in their followed approaches, one of them used Spring Boot framework and the other used CherryPy with Alchemy ORM in local company. The other two engineers have mid experience, the first has two years and the last participant has one year experience.

4.3. Survey

A survey was presented for the participants in first group, those who applied our RAAG framework and took training material about building REST services. Questions of survey were designed to explore opinions around three main topics; sufficiency of learning material, reliability of experiment, and RAAG framework quality. Around the first topic, opinions of the involved participants were very useful to indicate achievement range for the *"Reduce time to learn and to understand"* objective. These opinions used to demonstrate correspondence between participants' expectations and design assumptions, which assumes objectives realization when applying the proposed architecture.

Point of scale	1	2	3	4	5
Agreement	Strongly disagree	disagree	undecided	agree	Strongly agree

Figure 3. Likert scale options.

Table 1 illustrates the statistical information of the introduced survey, which have been analyzed based on Likert rating scale, where Likert rating scale applied five options, which included a neutral option [28, 29], as shown in figure 3. Data was gathered from seven participants after finishing their experiment over the proposed approach.

5. Results and Discussion

5.1. Survey Results

As can be seen in table 1, there is a high values of responds from the learning material, which indicates a good understanding for proposed approach (see Q1 & Q2 1). Results

Table 1. Survey Statistics.

Category	Question	Mean	SD
Sufficiency of Training Material	The training session (such as demonstrative video and docs) was clear.	4.14	0.69
	The information in the training video was helpful to complete the tasks and experiment.	4.57	0.53
	I needed more help to complete the tasks during the experiment.	2.14	.069
Average		**3.62**	**0.64**
Experiment Reliability	The required exercise included real-world scenarios.	**4.43**	**0.53**
Framework Performance and Quality	I am satisfied with how it was easy to use this framework to build REST API.	4.86	0.37
	This framework minimized usual efforts to build REST APIs.	4.71	0.48
	This framework reduced the time needed to create REST APIs.	4.57	0.89
	Making changes to the generated REST APIs required little time.	4	1
	I could fix problems easily and quickly in this framework.	4.14	0.69
	I believe that proposed framework can improve productivity.	4.7	0.48
	This framework has all the functions and capabilities that I expect to build REST APIs.	4	0.58
Average		**4.43**	**0.63**

confirm the understanding of learning material (see Q3 1), it shows the intermediate percentage of seeking help from the participants who performed the experiment, this degradation mostly refers to the high complexity of experiment model exercise, since model was designed to include all types of relations to simulate real-word scenarios. For reliability of experiment, the comprehensiveness of the proposed solution can illustrated after implementing the experiment that simulates real world scenarios (see Q4 1). And for framework quality, results demonstrate excellent expectations around efficiency, quality, and easy-to-use over proposed framework. (see table 1)

5.2. Experiment Results

Data charts demonstrates a significant variance on the required time between the two groups. Even though participants of the second group had high prior experience, the mean value of their spent time is more than twice of the first group. Furthermore, data of the first group shows easiness of learn, where a little impact on the required time versus many years of experience, which seems as neutral factor in the first group. For REST development aspect, charts demonstrate zero help requests with zero bugs, except one brief help for two of the fresh engineers, as shown in figure 4.

5.2.1. Analysis

Results analysis will focus on the impact of our approach on development time, analysis confirms the objectives (Reduce development time & Reduce time to learn and to understand), which leads to formulate below two hypotheses:

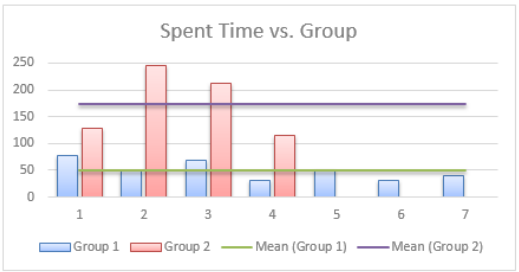
H1 *There is no significant difference in the development time for REST services when using traditional proposed RAAG framework, or other approaches.*

H2 *there is a significant difference between development time (over the proposed approach) in terms of experience.*

A B

Figure 4. (A) Experiment results for G1 and (B) Experiment results for G2.

Descriptive statistics was used to visualize collected data, and then analyze results, as shown in figure 6, and figure 7.



A

Spent Time	*Group 1*	*Group 2*
Mean	49.2857	174.75
Variance	318.5714	4066.25
Observations	7	4
Hypothesized Mean Difference	0	
df	3	
t Stat	-3.8498	
P(T<=t) one-tail	0.0155	
t Critical one-tail	2.3534	
P(T<=t) two-tail	0.0309	
t Critical two-tail	3.1824	

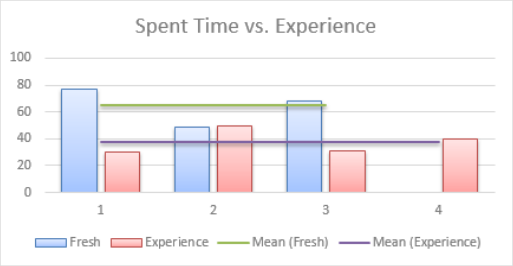
B

Figure 5. (A) Spent time in G1 vs. G2 and (B) Time difference calculations.

Figure 5 shows the spent development time for the two study groups, as per each participant. From figure 5, it is obvious that developers from first group (G1) seem to spend a shorter period. Moreover, it is noticeable that the distribution variation seems to be larger among the developers from the second group (G2). In total, there are seven G1 participants and four in G2. Where the mean time value for G1 developers is 49.3 minutes with a standard deviation of 17.9, and for the G2 mean time is 174.8 minutes with a standard deviation of 63.8. Huge mean's variance between G1 and G2 inspires that there is more than two times improvement on the productivity of G1. Thus, it is possible to investigate the difference statistically in the first hypothesis test.

On the contrary of time results, years of prior experience is not a prerequisite in first group, where mean years of experience in G1 is 3.9 Years, and 5.5 Years for second group. Results means that first group superior with less time and less prior experience too.

Further, the first group (G1) was divided into two sub-groups, G1.1 and G1.2. The sub-group G1.1 has the 4 experts participants, while G1.2 has 3 remaining novice participants. It can be seen from Figure 4 that sub-groups G1.1 and G1.2 have high difference in average years of experience. From table 2, and although of huge difference between means of experience (0 against to 6.75 Years), it can be seen that both sub-groups almost took same time for development during the session. figure 6 and figure 7 demonstrate the low difference in spent time between expert and novice participants. When looking into



A

Spent Time	*Fresh*	*Experience*
Mean	64.6667	37.7500
Variance	204.3333	86.9167
Observations	3	4
Hypothesized Mean Difference	0	
df	3	
t Stat	2.8398	
P(T<=t) one-tail	0.0328	
t Critical one-tail	2.3534	
P(T<=t) two-tail	0.0657	
t Critical two-tail	3.1824	

B

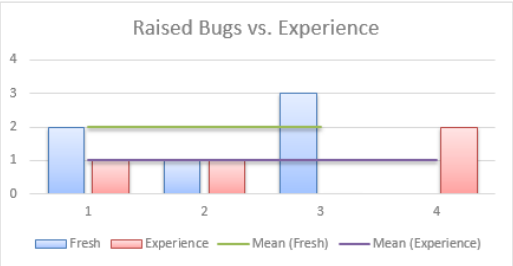
Figure 6. (A) Spent time in G1.1 vs. G2.1 and (B) Time difference calculations.

Table 2. Experiment descriptive statistics.

Desc	**Participants**		**Spent Time**		**Model Bug**		**REST Bug**	
Variable	Count	Exp	Mean	SD	Mean	SD	Mean	SD
Fresh	3	0	64.67	14.29	2.00	1.00	0.00	0.00
Experience	4	6.75	37.75	9.32	1.00	0.82	0.00	0.00

mean values of spent time and model bugs, both variables seem to be a little tendency according to heavy development duties in software, which indicates that no significant difference in effort to learn and understand in terms of experience. There is no extreme large deviations, and somehow both are closed together. Actually, data confirms that no REST bugs found and no request for help to implement REST APIs either from experts or fresh engineers, absolutely.

While the presented descriptive statistics have introduced an excellent insight into data, both variables tends toward what can be expected from the hypothesis testing, as the following.



A

Raised Bugs (REST Bugs = 0)	*Fresh*	*Experience*
Mean	2	1
Variance	1	0.6667
Observations	3	4
Hypothesized Mean Difference	0	
df	4	
t Stat	1.4142	
P(T<=t) one-tail	0.1151	
t Critical one-tail	2.1318	
P(T<=t) two-tail	0.2302	
t Critical two-tail	2.7764	

B

Figure 7. (A) Raised model bugs in G1.1 vs. G2.1 and (B) Bugs difference calculations.

Both hypotheses were evaluated by using a t-test, MS Excel was used for the statistical computing [30]. For used approach vs. productivity, results of two-tailed for unpaired t-test show (p-value = 0.0309 $<$ 0.05), as shown in Figure 5 (B). Therefore, we can conclude that the no significant difference hypothesis (H_0) is rejected, which means that there is a significant difference in productivity (with less spent time) for developers

Table 3. Frameworks comparison.

Framework Feature	RAAG	Spring	Apache Camel	CherryPy
Repetitive coding	Implicitly encapsulates repetitive code for all common operations, such as CRUD, logging, testing, permissions, ... etc.	Should be done explicitly in a repetitive way.	Should be done explicitly in a repetitive way.	Should be done explicitly in a repetitive way.
Parallel development	Different pieces of code can be plugged into RAAG to run as one application without more coding to be combined.	Requires explicit coding.	Requires explicit coding.	Requires explicit coding.
Code unification	Follows a simple unified style of coding.	Depends on developer.	Depends on developer.	Depends on developer.

of first group who used the proposed approach. Since p-value is less than 5% (0.05), results of spent time are highly significant.

For spent time vs. experience, results of two-tailed for unpaired t-test confirm that (p-value = $0.0657 > 0.05$), as shown in Figure 6 (B). And also for bugs vs. experience, t-test results illustrate that (p-value = $0.2302 > 0.05$), as shown in Figure 7 (B). Based on both results, we can conclude that the significant difference hypothesis (H_0) is rejected, which means that there is no significant difference in the spent time and raised bugs in terms of experience in first group. Hence, new approach is easy to use either from experts or fresh graduate. Since p-value is more than 5% (0.05), results of spent time and also raised bugs are not significantly different.

Actually, this framework was proposed as a complementary for the common used frameworks, and it was not introduced as a replacement for other frameworks. So, it aims to integrate several frameworks to facilitate development over them in an easy and unified approach without complex or repetitive coding. However, table 3 compares frameworks usage with and without RAAG.

5.3. Threats to validity

In general, our tool has two limitaions: first, the tool covers only the back-end layers and does not provide code genration for front-end layers. Secondly, the tool does not provide solution for unit tests auto-generation. On the other hand, the experiment has two threats that belong to internal validity, and those are developers' selection and used instrumentation. Developers were carefully involved to be familiar in database and web services development, and in the same time to be diverse in their experience. To avoid fatigue, each participant was only allocated to a single task in a single treatment, and most of them were strongly motivated through their employers to be serious enough. While in the second group, competition spirit was the common attitude to confirm superiority of their own approaches.

6. Conclusions

We proposed an approach and a framework that abstract layers for REST web APIs, business logic, data access, and model operations. Results from our preliminary evaluation shows a significant improvement on the productivity and easy-to-use. Meanwhile, survey

results confirmed RAAG design goals through an excellent indication around utilizing a useful frameworks in a loosely coupled manner without restrictions on customization and also about avoiding limitations of code generators. In the future work, since framework enables development by several teams in a different places as sprints, we intend to study RAAG compliance with Parallel Agile PA approach, and to study its impacts on the productivity. Furthermore, we intend to study the impact of framework centralism on building test cases, test-automation, security, and permissions plugins.

7. Acknowledgment

We would like to thank all participants that took part in our experimentation.

References

[1] Chenkai Guo, Jing Xu, Hongji Yang, Ying Zeng, and Shuang Xing. An automated testing approach for inter-application security in android. In *Proceedings of the 9th international workshop on automation of software test*, pages 8–14. ACM, 2014.
[2] ProgrammableWeb.com. Apis show faster growth rate in 2019 than previous years, 2019. https://www.programmableweb.com/news/research-shows-interest-providing-apis-still-high/research/2018/02/23, Last accessed on June 2019.
[3] Markus Fischer. *Model-driven code generation for REST APIs*. Thesis, 2015.
[4] Hamza Ed-douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Apicomposer: Data-driven composition of rest apis. In *European Conference on Service-Oriented and Cloud Computing*, pages 161–169. Springer, 2018.
[5] Christoforos Zolotas, Themistoklis Diamantopoulos, Kyriakos C Chatzidimitriou, and Andreas L Symeonidis. From requirements to source code: a model-driven engineering approach for restful web services. *Automated Software Engineering*, 24(4):791–838, 2017.
[6] Joop Aué, Maurício Aniche, Maikel Lobbezoo, and Arie van Deursen. An exploratory study on faults in web api integration in a large-scale payment company. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 13–22. IEEE, 2018.
[7] Hamza Ed-Douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Model-driven development of odata services: An application to relational databases. In *2018 12th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE.
[8] Rafael Corveira da Cruz Gonçalves and Isabel Azevedo. *RESTful Web Services Development With a Model-Driven Engineering Approach*, pages 191–228. IGI Global, 2019.
[9] Diego Serrano, Eleni Stroulia, Diana Lau, and Tinny Ng. Linked rest apis: a middleware for semantic rest api integration. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 138–145. IEEE.
[10] Bo Wang, Doug Rosenberg, and Barry W Boehm. Rapid realization of executable domain models via automatic code generation. In *2017 IEEE 28th Annual Software Technology Conference (STC)*, pages 1–6. IEEE.
[11] Kalvin Eng, Diego Serrano, Eleni Stroulia, and Jacob Jaremko. (semi) automatic construction of access-controlled web data services. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, pages 72–80. IBM Corp., 2018.
[12] Branko Terzić, Vladimir Dimitrieski, Slavica Kordić, Gordana Milosavljević, and Ivan Luković. Development and evaluation of microbuilder: a model-driven tool for the specification of rest microservice software architectures. *Enterprise Information Systems*, 12(8-9):1034–1057, 2018.
[13] Ángel Mora Segura, Jesús Sánchez Cuadrado, and Juan de Lara. Odaas: Towards the model-driven engineering of open data applications as data services. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, pages 335–339. IEEE.
[14] Florian Haupt, Dimka Karastoyanova, Frank Leymann, and Benjamin Schroth. A model-driven approach for rest compliant services. In *2014 IEEE International Conference on Web Services*, pages 129–136. IEEE.
[15] Nírondes AC Tavares and Samyr Vale. A model driven approach for the development of semantic restful web services. In *Proceedings of International Conference on Information Integration and Web-based Applications and Services*, pages 290–299. ACM, 2013.
[16] Dolors Costal, Carles Farré, Cristina Gómez, Petar Jovanovic, Oscar Romero, and Jovan Varga. Semi-automatic generation of data-intensive apis, 2017.
[17] Jeong-cheol Jeon and Jaehwa Chung. Developing a prototype of rest-based database application for shipbuilding industry: A case study. In *2017 International Conference on Platform Technology and Service (PlatCon)*, pages 1–6. IEEE.
[18] Digvijaysinh M Rathod, Satyen M Parikh, and BV Buddhadev. Structural and behavioral modeling of restful web service interface using uml. In *2013 International conference on intelligent systems and signal processing (ISSP)*, pages 28–33. IEEE.
[19] Hamza Ed-Douibi, Javier Luis Cánovas Izquierdo, Abel Gómez, Massimo Tisi, and Jordi Cabot. Emf-rest: generation of restful apis from models. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1446–1453. ACM, 2016.
[20] Hamza Ed-Douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Example-driven web api specification discovery. In *European Conference on Modelling Foundations and Applications*, pages 267–284. Springer, 2017.
[21] David Sferruzza, Jérôme Rocheteau, Christian Attiogbé, and Arnaud Lanoix. Extending openapi 3.0 to build web services from their specification. In *International Conference on Web Information Systems and Technologies*, 2018.
[22] David Sferruzza, Jérôme Rocheteau, Christian Attiogbé, and Arnaud Lanoix. A model-driven method for fast building consistent web services from openapi-compatible models. In *International Conference on Model-Driven Engineering and Software Development*, pages 9–33. Springer, 2018.
[23] Ali Asfour, Samer Zain, Norsaremah Salleh, and John Grundy. Exploring agile mobile app development in industrial contexts: A qualitative study. *International Journal of Technology in Education and Science*, 3(1):29–46, 2019.
[24] Diaeddin Rimawi and Samer Zein. A model based approach for android design patterns detection. In *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 1–10. IEEE, 2019.
[25] Robert C Martin. The principles of ood. url: http://butunclebob. com/articles. unclebob. *PrinciplesOfOod (visited on 01/09/2017)*, 2003.
[26] Bernd Bruegge and Allen H Dutoit. Object–oriented software engineering. using uml, patterns, and java. *Learning*, 5(6):7, 2009.
[27] Harmeet Singh and Syed Imtiyaz Hassan. Effect of solid design principles on quality of software: An empirical assessment. *International Journal of Scientific and Engineering Research*, 6(4), 2015.
[28] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.
[29] Likert scale questions: Definitions, examples + how to use them — typeform. https://www.typeform.com/surveys/likert-scale-questionnaires/. (Accessed on 01/04/2020).
[30] How to calculate p value in excel (step-by-step tutorial). https://spreadsheeto.com/p-value-excel/. (Accessed on 01/10/2020).