FEDERAL UNIVERSITY OF FRONTEIRA SUL
CAMPUS OF CHAPECÓ
COURSE OF COMPUTER SCIENCE

DOUGLAS ROSA

A PROTOTYPE WEB PLATFORM TO CONFIGURE APIS FOR MOBILE APPLICATIONS WITH GENERIC DATA STRUCTURES IN DATABASES

CHAPECÓ
2022

DOUGLAS ROSA

A PROTOTYPE WEB PLATFORM TO CONFIGURE APIS FOR MOBILE APPLICATIONS WITH GENERIC DATA STRUCTURES IN DATABASES

Final undergraduate work submitted as requirement to obtain a Bachelor's degree in Computer Science from the Federal University of Fronteira Sul.
Advisor:: Prof. Dr. Samuel da Silva Feitosa

CHAPECÓ
2022

Rosa, Douglas

A PROTOTYPE WEB PLATFORM TO CONFIGURE APIS FOR MOBILE APPLICATIONS WITH GENERIC DATA STRUCTURES IN DATABASES /Rosa, Douglas. – 2022.
47 pp.: il.

Advisor:: Prof. Dr. Samuel da Silva Feitosa.
Final undergraduate work – Federal University of Fronteira Sul, course of Computer Science, Chapecó, SC, 2022.

1. Configurable APIs. 2. Generic data structures. 3. Mobile applications. 4. Model-Driven Engineering. 5. Generation of RESTful APIs. I. Prof. Dr. Samuel da Silva Feitosa, advisor. II. Federal University of Fronteira Sul.

DOUGLAS ROSA

A PROTOTYPE WEB PLATFORM TO CONFIGURE APIS FOR MOBILE APPLICATIONS WITH GENERIC DATA STRUCTURES IN DATABASES

Final undergraduate work submitted as requirement to obtain a Bachelor's degree in Computer Science from the Federal University of Fronteira Sul.

Advisor:: Prof. Dr. Samuel da Silva Feitosa

This final undergraduate work was defended and approved by the examination committee on:

19 Aug 2022.

ACKNOWLEDGEMENT

First and foremost, I give thanks to **God** for all his protection, kindness and care with this objective reached and so important to my professional career giving me strength in discouraging times at university.

I thank for having my wife **Graziele Garbin Moreira da Silva Rosa** and my little boy **Augusto Moreira Rosa** during all this time, making my days lighter and happier, hugely missing them in the travels and stays in the city of the university campus.

Thanks to my parents, **Lovani Maria Halmenschlager** and **José Nerino Rosa**, for their love and support throughout my life and in particular my mother for her dream of having her children graduated.

Not forgetting my first and continuous reference in software development area, my brother **Fernando Rosa**, for transmitting enthusiasm in all the conversations that involve web development.

I am profoundly grateful to my final undergraduate work advisor, **Prof. Dr. Samuel da Silva Feitosa**, an amazing and passionate teacher about everything he does, for their guidance throughout this study and for their confidence in me.

I also would like to acknowledge **Prof. Dr. Denio Duarte** and **Prof. Dra. Raquel Aparecida Pegoraro** as the board examiners of this defense. I am gratefully indebted to them for their very valuable comments on this work.

ABSTRACT

In a professional software development context, it is highly noted the importance of professionals who have enough expertise to deliver a complete product (a.k.a. full-stacks) in companies with reduced technical staff, or in the early stages of startups. However, it is evident the lack of mastery of the technologies to develop APIs by these professionals, not only regarding the implementation of routes, but also the hosting project in the cloud. This problem, if not taken care of, end up resulting in low performance and security problems. Not least, this ecosystem deconstructs expert profiles in mobile careers. In this context, this project proposes the development of a web platform for configuring new APIs to be consumed by these applications, allowing developers/companies to model their systems without technical skills in the back-end and infrastructure. The proposed API configurations are analogous to schema files, very common in relational databases, but supporting real-time changes without requiring the unavailability of services.

Keywords: Configurable APIs; Generic data structures; Mobile applications; Model-Driven Engineering; Generation of RESTful APIs;

FIGURES

TABLES

SUMMARY

1 INTRODUCTION

Careers in mobile applications usually require greater technical mastery of languages, architectures, and features that many times assume a lower level of abstraction needing native implementations. Integrated Development Environments (IDEs) such as XCode, Android Studio, and others have numerous helping tools, like the emulation of specific architectures of hardware, and operating systems, which are most of the time physically unavailable.

In contrast, full-stack developers have extensive knowledge of all aspects of a given technology stack. They can quickly turn a concept into a working solution. Their big picture and visibility allow them to anticipate issues in advance and guide projects around them. Many organizations actively recruit these developers. Full-stack development is an upward trending concept in software development. Developers are expected to design, build and implement end-to-end technology solutions that meet business requirements (24).

There are also cons to be considered when choosing a specialist career instead of a full-stack profile, since it brings high responsibilities, sometimes outside of the studied and experienced area (43). Among the factors that influence the decision, there are market offers and the professional's economic situation. It is important to mention there is a growing market in the mobile development area.

We can witness professional specialization at business level, where several companies outsource software products and services. As a result, we can note internal competitiveness in the country, a decisive factor in guaranteeing the quality of products and services. By outsourcing part of their activities, companies make their processes more efficient and competitive, adding competence and technical quality to their production stages (11).

In this context, this practical project initiative proposes the development of a WEB platform, in which these mobile application developers can refrain from this stage of development that they have no experience or interest in, allowing the construction of models, their relations, and also the business rules, consuming the API accordingly to the documentation that will be provided, with no need for technical skills about back-end and infrastructure. In addition, the results of this project intend to improve the security and performance, the introduction of best practices in the development of APIs, as well as in structuring and relating tables and in configuring servers.

1.1 RESEARCH PROBLEM

Is it possible to provide for mobile application developers a web platform for building configurable APIs to consume data in the cloud avoiding the need to perform tasks related to the back-end and its infrastructure?

1.2 RESEARCH HYPOTHESIS

A web platform for building configurable APIs can be useful for mobile application developers aiding the consumption of data in the cloud without the need to perform back-end and infrastructure tasks.

1.3 OBJECTIVES

1.3.1 General objectives

Develop a web platform for building configurable APIs to aid mobile application developers to consume data on the cloud without the need to perform back-end and infrastructure tasks.

1.3.2 Specific objectives

- Review the literature related to mobile development and database concepts;
- Propose a generic model structure for databases without the generation of specific tables;
- Develop a prototype implementation for a web platform to model APIs hosted in the cloud;
- Document all implemented data structures in database and application levels;

1.4 JUSTIFICATION

Developers of mobile applications need to pay attention to hybrid and/or native languages, tools with numerous resources and updates of Android and iOS operating systems for mobile architectures, simulation and debugging in specific physical environments and situations, optimization of hardware and storage resource data synchronized in the cloud (20).

It becomes impossible for the mobile application developer to temporarily absent himself from this context for the development of endpoints to be consumed in mobile applications and to enter a completely different one: the development of the back-end with maintenance in databases and eventual maintenance in dedicated servers (5).

Many professionals seek to be specialists in what they do, allied to corporate objectives, which aim to guarantee quality in the services and products that they provide. This is a reality in many corporations, which generally are in a stage with well-segmented sectors internally. However, on the other side, in corporations that are growing up or that are in initial stage, like start-ups, there is a hard culture of full-stack developers (27).

The objective of this practical work is exactly to meet this need to assign functions to full-stack mobile application developers by offering a tool that can focus their efforts in the context of mobile development and exempt themselves from the back-end development and deployment environments.

1.5 STRUCTURE

The remainder of this text is organized as follows: Chapter 2 summarizes the background concepts, including mobile and back-end development, and existing tools related to this proposal. Chapter 3 presents the steps which should be followed during the development of this project. Chapter 4 presents the implementation characteristics of the prototype, and finally, Chapter 5 brings some final considerations about the presented text.

2 BACKGROUND READING

2.1 FULL-STACK

Full-stack development is a methodology that addresses all layers of the application. In practice, the professionals who fit into this category are developers who have enough skills and experience to deliver a minimum viable product in a certain defined stack but not necessarily mastery in all layers (24). The Figure 1 shows an example of some well known technologies, but not limited to them, in full-stack cultures.

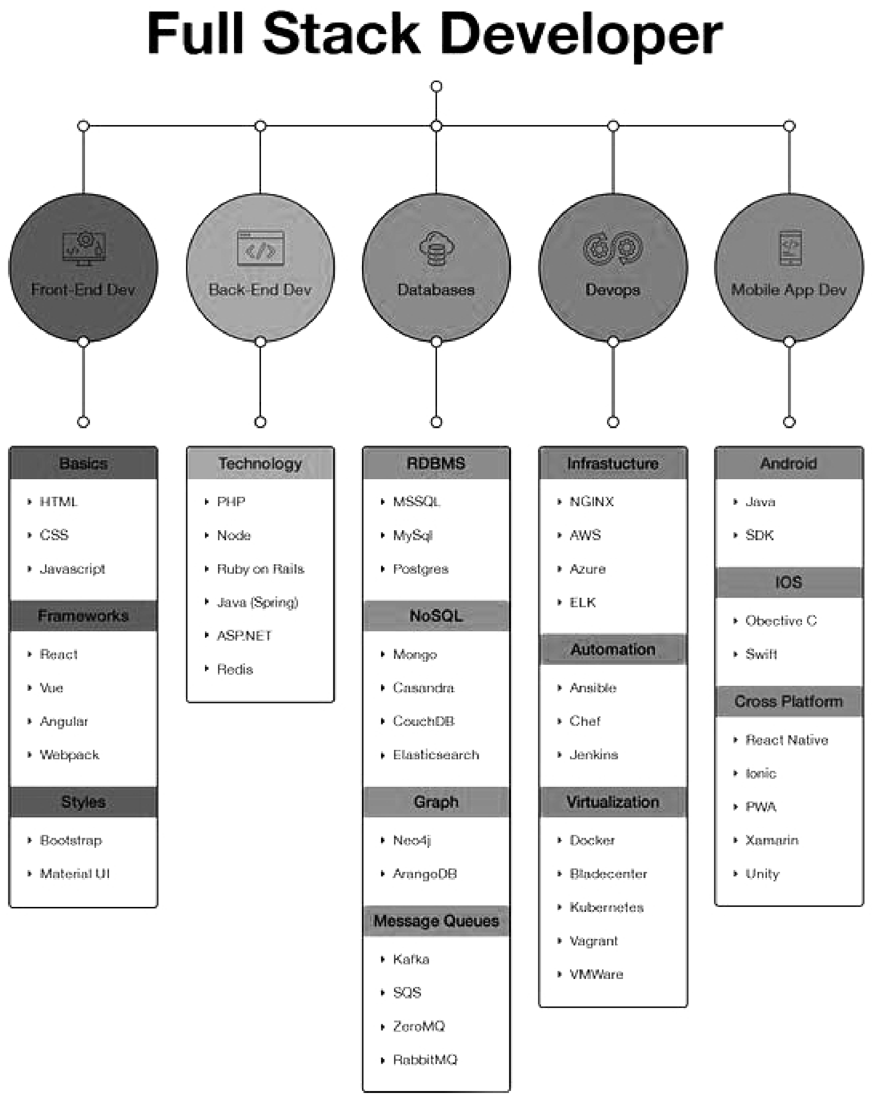


Figure 1 – Full-Stack in start-ups.

2.2 START-UPS

Startups are young companies founded to develop a unique product or service, bring it to market and make it irresistible and irreplaceable for customers. Rooted in innovation, a startup aims to remedy deficiencies of existing products or create entirely new categories of goods and services, disrupting entrenched ways of thinking and doing business for entire industries (17).

Startups apply principles learned from manufacturing and supply chain management to bring together the principles of customer development and agile methodologies seeking to reduce waste by mitigating risk through short and frequent actions, tests and fixes (10).

2.3 MOBILE DEVELOPMENT

When a developer says they are a mobile app programmer, most of the time they are referring to that front-end part (5).

Depending on the size of the team producing the app, there can be many different people involved in the front-end mobile app design and development (41). The size of the team can range from a single developer who does everything associated with building the app, to dozens, hundreds, and more people with specialized skills (5). The Figure 2 shows the mobile developer workflow commonly seen in corporations.

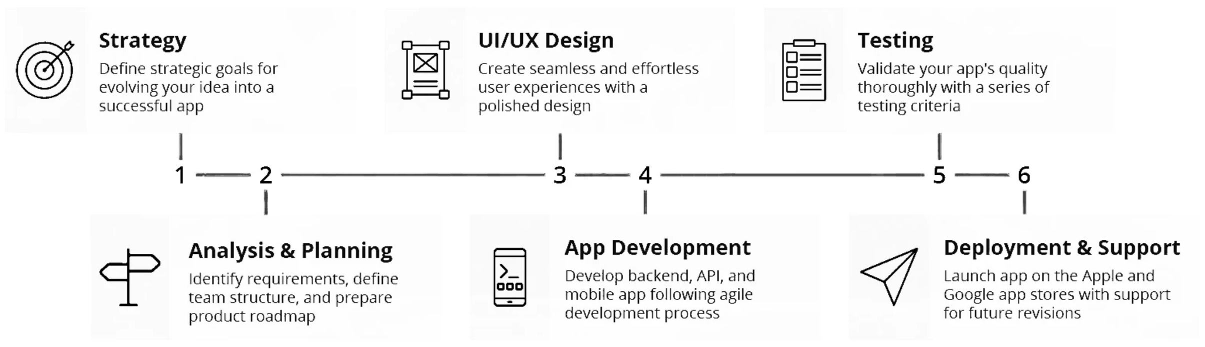

Strategy
Define strategic goals for evolving your idea into a successful app
UI/UX Design
Create seamless and effortless user experiences with a polished design
Testing
Validate your app's quality thoroughly with a series of testing criteria
1
2
3
4
5
6
Analysis & Planning
Identify requirements, define team structure, and prepare product roadmap
App Development
Develop backend, API, and mobile app following agile development process
Deployment & Support
Launch app on the Apple and Google app stores with support for future revisions

Figure 2 – Mobile app development process.

2.3.1 Dominant market

There are two dominant platforms in the modern smartphone market. One is Apple iOS platform. The iOS platform is the operating system that powers Apple's popular iPhone line of smartphones. The second is Google's Android. The Android OS is used not only by Google devices, but also by many other to build their own smartphones and other smart devices (5).

While there are some similarities between these two platforms when building apps, iOS development versus Android development involves using different software development kits (SDKs) and a different development tool chain (20).

2.4 BACK-END DEVELOPMENT

For most applications, service developers are responsible for creating and managing back-ends for their applications. The mobile developer may not be an expert or even particularly skilled at building and running a back-end infrastructure (5).

Developers can take advantage of a cloud service provider – a back-end as a service (BaaS) provider – that handles all the heavy lifting and back-end management grunt work, so developers can focus only on features and functionality that are built into their applications, without worrying about scalability, security and reliability (5).

2.4.1 APIs

APIs (Application Programming Interfaces) offer a simple way for connecting to, integrating with, and extending a software system, in this case, mobile applications. More precisely, APIs are used for building distributed software systems, whose components are loosely coupled. The APIs studied in this project are web-APIs, which deliver data resources via a web technology stack. Typical applications using APIs are mobile apps, cloud apps, web applications, or smart devices (9).

The advantages of using APIs are that they are simple, clean, and approachable. They provide a reusable interface that different applications can connect to easily. However, APIs do not offer a user interface, they are usually not visible on the surface, and typically, no end-user will directly interact with them. Instead, APIs operate under the hood and are only directly called by other applications. APIs are used for machine-to-machine communication and for the integration of two or more software systems (9). The Figure 3 shows a sample of how many kinds of clients an API can serve to respect the particularities of each one.

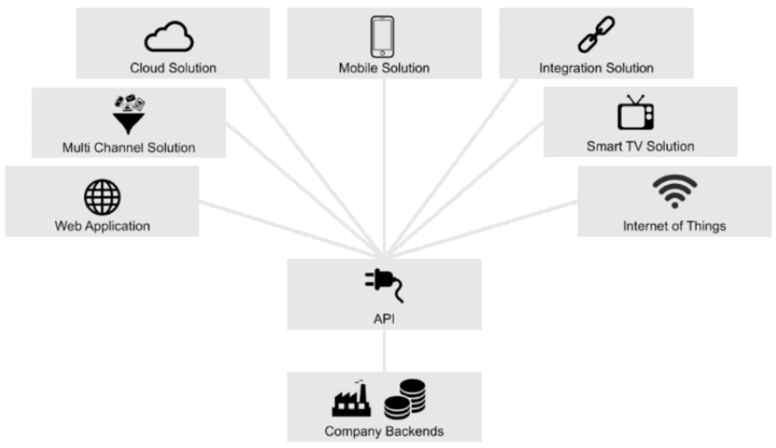


Figure 3 – API client connections.

2.4.1.1 REST

REST (REpresentational State Transfer) is an architectural style that defines the set of rules to be used for creating web services. Web services that follow the REST architectural style are known as RESTful web services (19).

> In 2000, Roy Fielding, one of the key contributors to HTTP and URI, codified the architecture of the Web in his doctoral thesis titled "Architectural Styles and the Design of Network-Based Software Architectures." In this thesis, he introduced an architecture style known as Representational State Transfer (REST). This style, in abstract terms, describes the foundation of the World Wide Web. The technologies that make up this foundation include the Hypertext Transfer Protocol (HTTP), Uniform Resource Identifier (URI), markup languages such as HTML and XML, and web-friendly formats such as JSON (2).

REST is designed to make optimal use of an HTTP-based infrastructure and the HTTP protocol (9). There are six architectural constraints that define the REST style, as follows:

Uniform Interface. The interface must uniquely identify each resource involved in the interaction between the client and the server (36).

Stateless. The server never stores any application state. In a stateless application, the server considers each client request in isolation and in terms of the current resource state. If the client wants any application state to be taken into consideration, the client must submit it as part of the request. This includes things like authentication credentials, which are submitted with every request (38).

Cacheable. The cacheable constraint requires that a response should implicitly or explicitly label itself as cacheable or non-cacheable. If the response is cacheable, the client application gets the right to reuse the response data later for equivalent requests and a specified period (36).

Client-Server. The client-server design pattern enforces the separation of concerns, which helps the client and the server components evolve independently. By separating the user interface concerns (client) from the data storage concerns (server), we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components (36).

Layered System. The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior. For example, in a layered system, each component cannot see beyond the immediate layer they are interacting with (36). The Figure 4 shows the workflow between clients that request some resource to the server using the HTTP format, which validates the requested format, processes, and responds to the resources required in one of the defined formats.

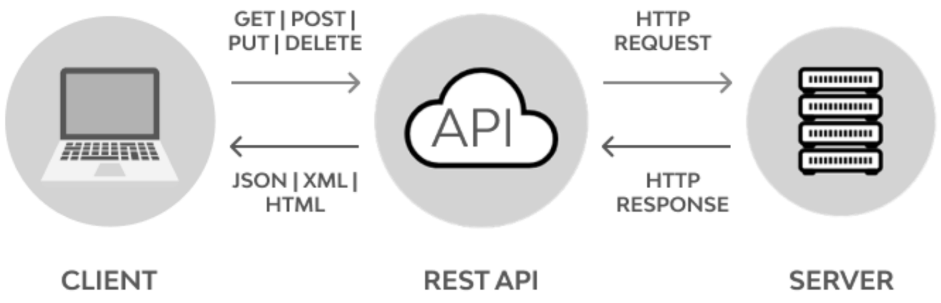


Figure 4 – Rest APIs workflow.

2.4.1.2 GraphQL

A GraphQL API is also another option for developers as it makes it easier to work with back-end data in a mobile app. GraphQL supports queries through a single API endpoint and a data schema that can be used to easily create and extend data models used in the application (5).

GraphQL is a data fetching language that allows clients to declaratively describe their data requirements in a JSON-like format. In this respect, it is comparable to SQL. Furthermore, GraphQL is database agnostic. Its design allows one to deal with any kind of database (9).

GraphQL could be implemented once with its high flexibility and dynamic nature, allowing the clients to pick and choose the data without having to have to worry if an endpoint exists or not for the specific query (25).

2.4.2 Business rules

Business rules, sometimes referred to as business logic or application logic, are what govern how an app should perform and what validations to apply.

Client-side validations are useful to provide immediate feedback to the user, which makes for a better user experience. For instance, when writing a tweet, it is possible to instantly see if it went over the 140-character limit or not. However, client-side validations are not enough on their own since they are easily bypassed. Browsers (and mobile apps for that matter) can be hacked to bypass certain restrictions put in by the application's developers. Browsers allow one to inspect and modify the HTML and JavaScript quite easily, so it is not a good practice to rely only on front-end validations (15).

2.5 MARKET ANALYSIS

2.5.1 Configure.IT

Configure.it is a fully web platform for the creation of APIs and the generation of front-end screens in native code for both platform in Java and Swift languages. The corporation started 10 years ago, with its software contemplating several features, which seems to be very complete and robust (13). The figure 5 shows the initial creation of a new screen which can be later exported to native android/iOS native language and the figure 6 shows the interface for modeling new tables, treated as models, similar to the proposal of this work (12).

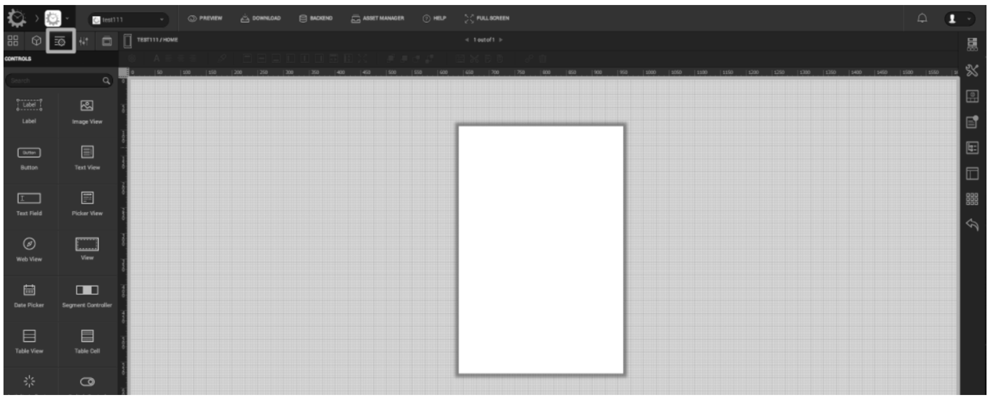
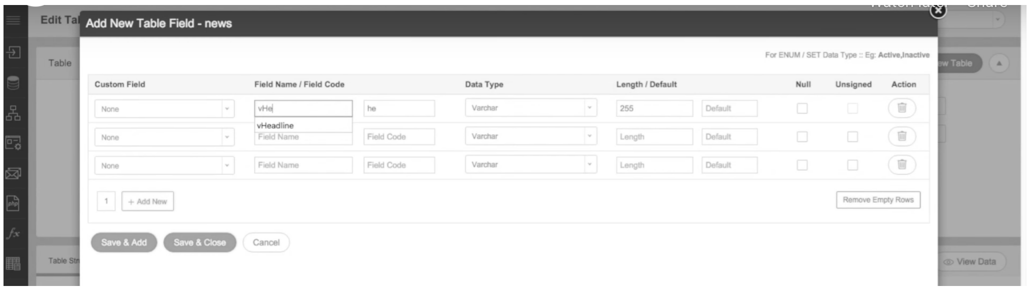


Figure 5 – Designing a new screen in Configure.IT.


Add New Table Field - news

For ENUM / SET Data Type :: Eg: Active,Inactive

Custom Field	Field Name / Field Code		Data Type	Length / Default		Null	Unsigned	Action
None	vHe	he	Varchar	255	Default	☐	☐	🗑
None	Field Name	Field Code	Varchar	Length	Default	☐	☐	🗑
None	Field Name	Field Code	Varchar	Length	Default	☐	☐	🗑

vHeadline

1 + Add New

Remove Empty Rows

Save & Add Save & Close Cancel

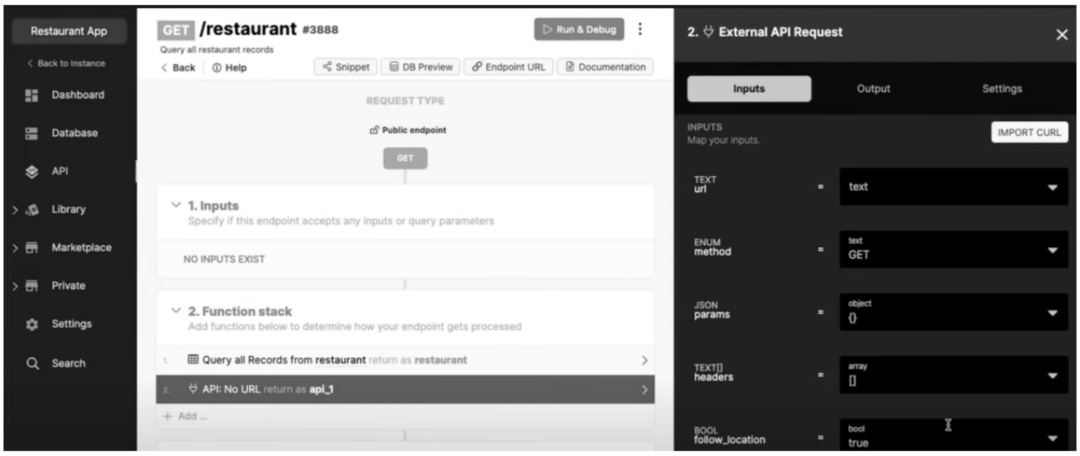
Figure 6 – Modeling a new table in Configure.IT.

In general terms, the platform seems to be quite complete in terms of what it proposes to do, not only configuring the back-end to be consumed, but also generating code for native web and mobile applications, which is an aspect that differs from the proposal of this work, as defined in the methodology section.

2.5.2 Xano

Xano is another fully web platform as the Configure.IT, but without generation of native code. The platform worry mainly about the back-end functionalities. It was originally created to help to accelerate the speed of back-end development while keeping the team small (13). The

Figure 7 presents the interface for the configuration of endpoints created on the platform (44) and the figure 8 shows the interface for listing tables created into the platform (44).



GET /restaurant #3888
Query all restaurant records

REQUEST TYPE
Public endpoint
GET

1. Inputs
Specify if this endpoint accepts any inputs or query parameters
NO INPUTS EXIST

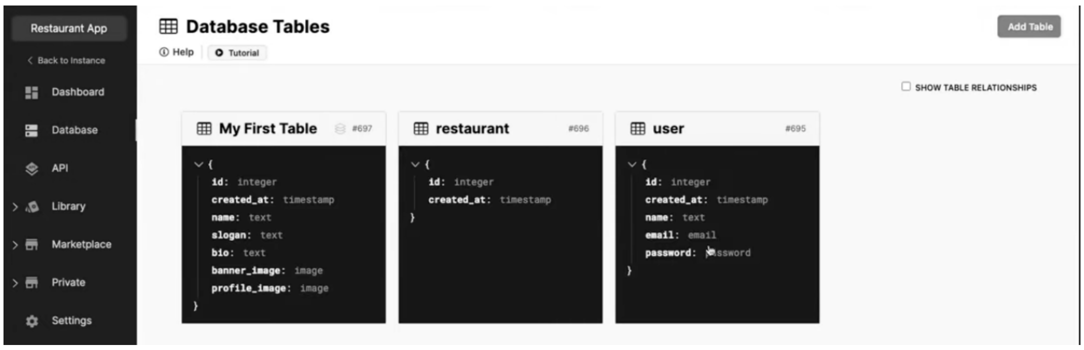
2. Function stack
Add functions below to determine how your endpoint gets processed
1. Query all Records from restaurant return as restaurant
2. API: No URL return as api_1

2. External API Request

INPUTS
Map your inputs.

TEXT url	=	text
ENUM method	=	text GET
JSON params	=	object {}
TEXT[] headers	=	array []
BOOL follow_location	=	bool true

Figure 7 – Showing a endpoint created in Xano.



My First Table #697
```
{
  id: integer
  created_at: timestamp
  name: text
  slogan: text
  bio: text
  banner_image: image
  profile_image: image
}
```

restaurant #696
```
{
  id: integer
  created_at: timestamp
}
```

user #695
```
{
  id: integer
  created_at: timestamp
  name: text
  email: email
  password: password
}
```

Figure 8 – Showing the table characteristics in Xano.

2.5.3 Backendless

Backendless is a no-code application development platform designed to streamline and accelerate the mobile application development process. The platform consists of a no-code UI builder, mobile back-end-as-a-service (mBaaS) product, API management solution and a hosting environment (7). The Figure 9 shows the interface for listing tables created into the platform (8) and the Figure 10 shows the interface for listing data generated through the data models into the platform (8).

2.5.4 Parse.com

Parse.com is the most used open-source framework to develop application back-ends. It helps developers to accelerate app development and reduces the total amount of effort required to build an app. A large community of engaged developers supports the platform which has been evolving since 2016 (33).

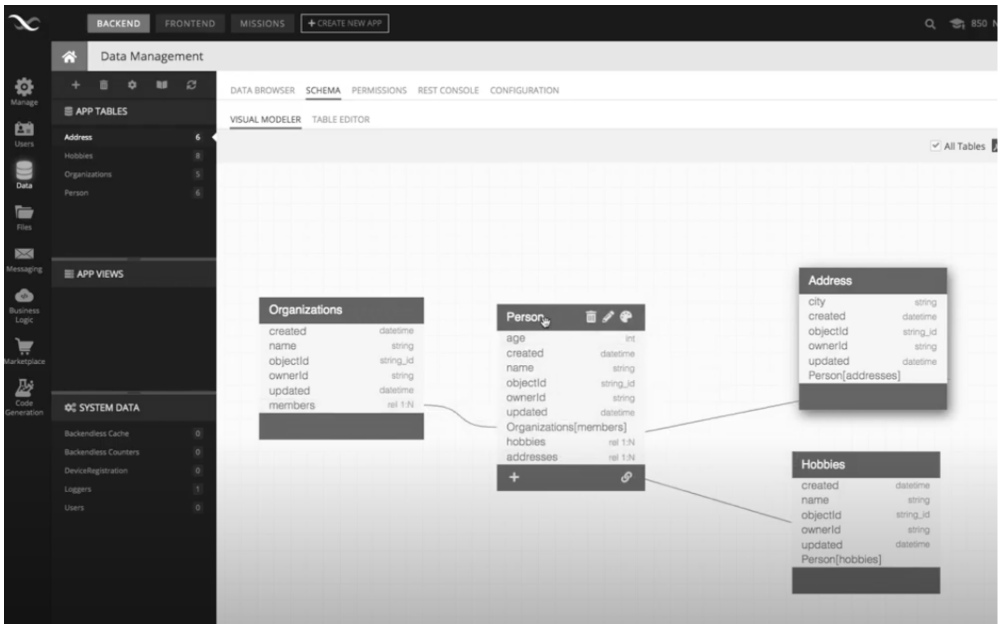
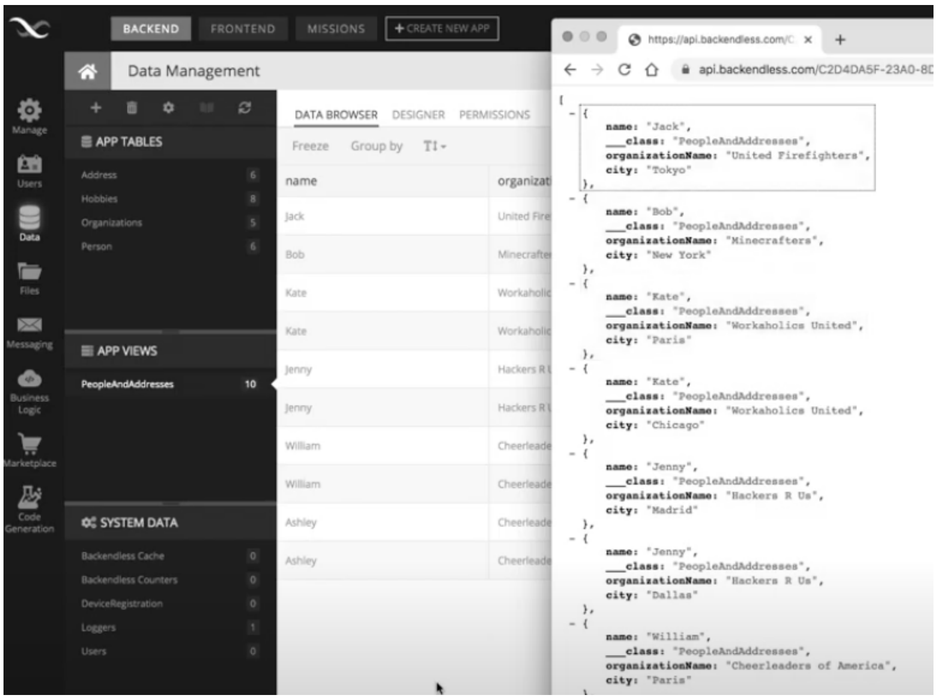


Figure 9 – Relation between models in Backendless.

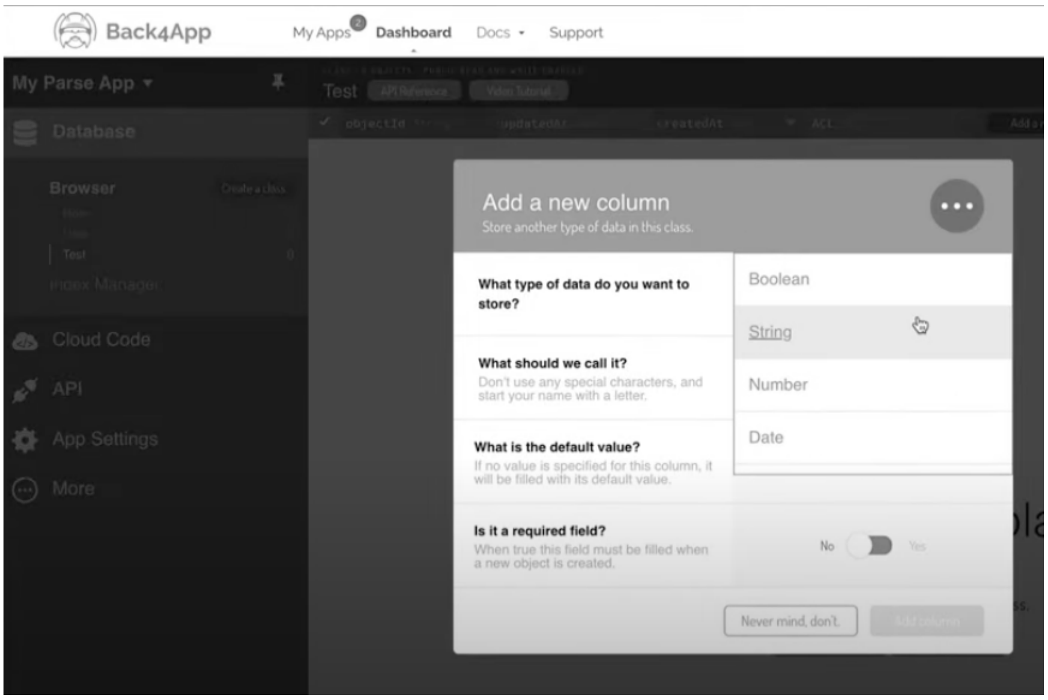


```
[
 - {
      name: "Jack",
      ___class: "PeopleAndAddresses",
      organizationName: "United Firefighters",
      city: "Tokyo"
   },
 - {
      name: "Bob",
      ___class: "PeopleAndAddresses",
      organizationName: "Minecrafters",
      city: "New York"
   },
 - {
      name: "Kate",
      ___class: "PeopleAndAddresses",
      organizationName: "Workaholics United",
      city: "Paris"
   },
 - {
      name: "Kate",
      ___class: "PeopleAndAddresses",
      organizationName: "Workaholics United",
      city: "Chicago"
   },
 - {
      name: "Jenny",
      ___class: "PeopleAndAddresses",
      organizationName: "Hackers R Us",
      city: "Madrid"
   },
 - {
      name: "Jenny",
      ___class: "PeopleAndAddresses",
      organizationName: "Hackers R Us",
      city: "Dallas"
   },
 - {
      name: "William",
      ___class: "PeopleAndAddresses",
      organizationName: "Cheerleaders of America",
      city: "Paris"
```

Figure 10 – Listing data of modeled tables in Backendless.

2.5.5 Back4app

Back4app is based on Parse.com with many improvements in functionalities as well as developer experience in it. The platform offers features with low-code like in Push Notifications and many kinds of integration like GraphQL, REST APIs, and Cross-Platform SDKs including libraries for native apps. The Figure 11 shows the modeling a new class attribute into the platform (6).



Add a new column
Store another type of data in this class.

What type of data do you want to store?

What should we call it?
Don't use any special characters, and start your name with a letter.

What is the default value?
If no value is specified for this column, it will be filled with its default value.

Is it a required field?
When true this field must be filled when a new object is created.

Figure 11 – Modeling a class attribute on Back4app

2.5.6 Appmachine.com

Appmachine.com is a platform very similar to the Configure.It about the functionalities, with methods to create the applications inside it and the final result generating native code for mobile applications. It uses the concept of LEGO assembled with ready-to-use functional blocks. The Figure 12 shows the creation of a screen layout into the platform (3).

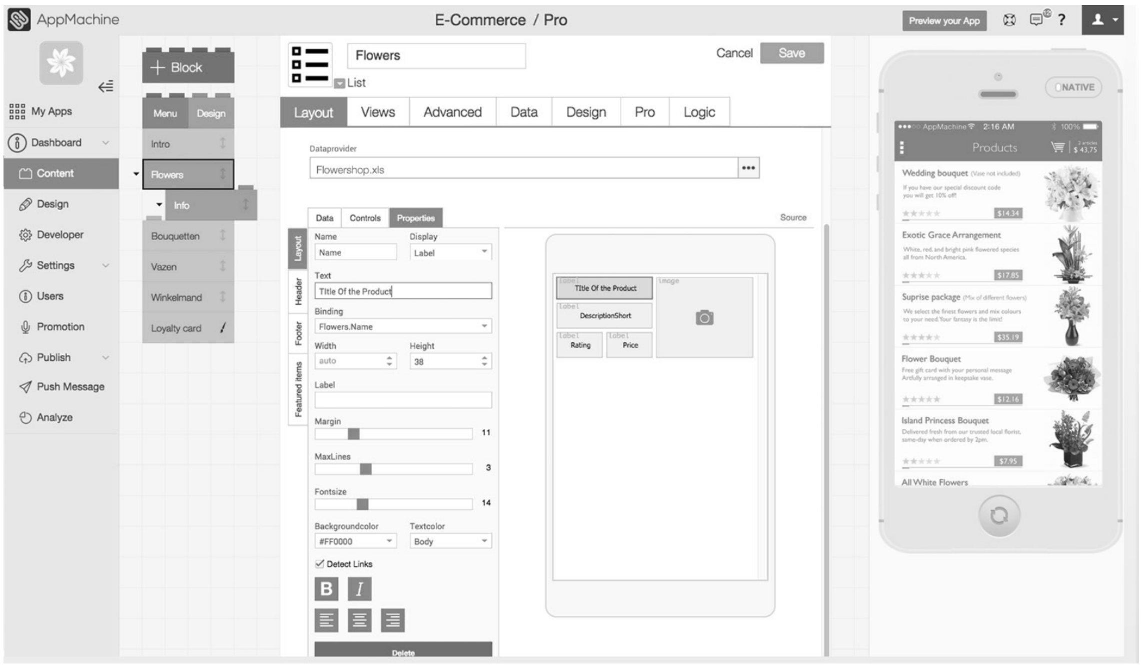


Figure 12 – Designing layout for a screen

2.6 CROSS MARKET FUNCTIONALITIES

The Table 1 presents the crossing of market features with the proposed project considering only core functionalities.

Platform features	**PROTOTYPE**	Configure.It	Xano	Backendless	Parse	Back4app	AppMachine
WEB Platform to manage products	x	x	x	x	x	x	x
Configure APIs	x	x	x	x	x	x	x
Integration for each product	x	x	x	x	x	x	x
Configure models and attributes	x	x	x	x	x	x	x
Relationship between models	x	x	x	x	x	x	x
Definition of business rules	x	x	x	x	x	x	x
Manipulate data records	x	x	x	x	x	x	x
Integration by JSON	x	x	x	x	x	x	x
SDK for integration in JS	x					x	
Sample in React-Native	x						
Generation of frontend		x					x
Generation of backend	x		x		x		x

Table 1 – Crossing concurrent features with our purpose

2.7 RELATED WORK

Several efforts have been made to bring together MDE and Web Engineering. This field is usually referred to as Model Driven Web Engineering (MDWE) and proposes the use of models and model transformations for the specification and semiautomatic generation of Web applications (16).

Some of these works provide support for the generation of Web services as well, but support for generation of RESTful APIs is very limited (37). Moreover, these approaches require the designer to specifically model the API itself using some kind of tool-specific DSL from which then the API is (partially) generated. Instead, our approach is able to generate a complete RESTful API implementation from a plain data model (16).

2.8 CHAPTER'S FINAL REMARKS

This chapter presented some important concepts which were useful to proceed with the development of this project. In addition, some related tools were presented, which were used to help in the understanding of the objectives of this work and in studying different features which could be implemented in the proposed prototype.

3 METHODOLOGY

The Minimum Viable Product (MVP) proposed consists of a web platform that make it possible to configure APIs to be integrated into mobile applications with limited features. Some of them can be changed or tweaked during the development process, however, the main concepts should be preserved, i.e., the platform should not generate specific tables as in traditional back-ends development, and it is out of the scope of this project the generation of front-end code.

The features and requirements for the first presentable MVP have enough functionalities to represent what this work proposes. The main features expected for the practical project are listed below:

- Data models and their attributes
- Integration token for each API
- CRUD of registers belonging to models

To be able to produce the proposed project, there are several steps to be followed, as described next:

Review database concepts. Consists of analyzing conceptual documents to further explore unexplored features of relational and non-relational database architectures.

List MVP functional requirements. Consists of producing a list of requirements for the project, identifying essential functional requirements for the delivery of the minimum viable product for what the platform proposes.

List and test candidate technologies. Study different technologies to choose the appropriate ones for this project, as well as evaluate the required and available resources, experimenting each of them.

Define the road map. Consist of defining the script board of stories that have high priority according to the functional requirements. In this step, inter-dependencies of stories should be identified, allowing changes to the design of the proposed MVP.

Organization of tasks. This step consists of splitting tasks into small parts as possible applying the Agile concept.

Implement the MVP. The implementation of the proposed MVP according to the functional requirements, and available resources, considering deadlines for the release.

Write the monography. Complement the text of this monography explaining the complete steps performed to achieve the results of this project, as well the conclusions about the validation of the MVP.

4 PROTOTYPE

This chapter describes the implementation of the prototype WEB platform, as well as the mobile client sample, presenting the particularities of each them.

4.1 PROJECT

Was adopted the Agile methodology (1) to organize sprints with deadlines of an average of 15 days. There were exceptions and unachieved goals in some of them. Frequent brainstorms were carried out to restructure and adapt the initial idea to provide the minimum viable product as soon as a short development period was noticed, which in our case was less than 2 months.

We used Google Docs (22) to organize the development sprints, containing basic deadlines and checklists of tasks to be completed by each date, with remote meetings held on Google Meet (23). We used Diagrams.net (26) to create the entity-relationship diagram, which was implemented later on this project.

4.2 PLATFORM DEVELOPMENT

This section refers to the administration interface for the APIs that was modeled, as well as the backend development that supports and responds to all requests made by this platform, as well as the client mobile applications that consumes the routes for manipulating model records.

4.2.1 Backend Development

We are referring to the database modeling activities, the technology selection for implementation, the specifics of their configuration in a local development environment, and the project's organizational standards.

4.2.1.1 Technologies

Among the main technologies used in the backend project (but not limited to) are:

NodeJS
Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project (32).

Typescript
TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale (31).

Sequelize

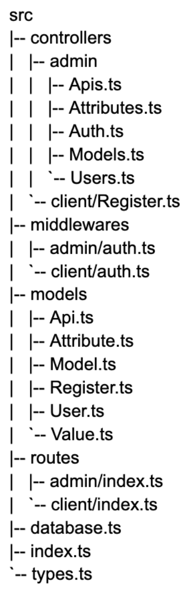
Sequelize is a modern TypeScript and Node.js ORM for Postgres, MySQL, MariaDB, SQLite and SQL Server, and more. Featuring solid transaction support, relations, eager and lazy loading, read replication and more. (39) (40).

PostgreSQL

PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development that has earned a strong reputation for reliability, feature robustness, and performance (42)

4.2.1.2 Patterns

Figure 13 represents the organization of the files on the backend project. The following organization (design pattern) was chosen because it is widely used and accepted in the backend community, as well as because it perfectly attend the project's didactic requirements.



```
src
|-- controllers
|   |-- admin
|   |   |-- Apis.ts
|   |   |-- Attributes.ts
|   |   |-- Auth.ts
|   |   |-- Models.ts
|   |   `-- Users.ts
|   `-- client/Register.ts
|-- middlewares
|   |-- admin/auth.ts
|   `-- client/auth.ts
|-- models
|   |-- Api.ts
|   |-- Attribute.ts
|   |-- Model.ts
|   |-- Register.ts
|   |-- User.ts
|   `-- Value.ts
|-- routes
|   |-- admin/index.ts
|   `-- client/index.ts
|-- database.ts
|-- index.ts
`-- types.ts
```

Figure 13 – Files tree of the Backend.

4.2.1.3 Middlewares, Routes and Controllers

As can be seen, routes, controllers, and middlewares were separated by consumer application, with one structure handling requests from the WEB platform (called Admin) and another handling requests from mobile applications (called Client).

4.2.1.3.1 Middleware

Functions that have access to the request object, the response object, and the next middleware function in the application's request-response cycle. Middleware can perform tasks such as make changes to the request and the response objects, end the request-response cycle and call the next middleware in the stack (28).

Each type of application (admin and clients) has its own authentication middleware and totally different integration rules which will be seen in the security section.

4.2.1.3.2 Admin Routes

The following routes have been implemented to be consumed by the Admin WEB platform:

Unauthenticated
Unauthenticated routes, for example, do not need to send any access token information in the request header.

POST **/admin/login** - For authentication
body: { login: string; password: string; }

POST **/admin/users** - To create users
body: { login: string; password: string; }

Authenticated
Authenticated routes, in the other hand, need to send the access token information in the request header. The same will be used to verify not only whether the user has an active login session, but also whether the user who made the request has access to what he expects to receive from the request.

For manage APIs:

- GET **/admin/apis** - get all APIs
- GET **/admin/apis/:uuid** - get one API
- POST **/admin/apis** - create one API
- PUT **/admin/apis/:uuid** - update one API
- DELETE **/admin/apis/:uuid** - delete one API

For manage Models:

- GET **/admin/models** - get all Models
- GET **/admin/models/:uuid** - get one Model

- POST **/admin/models** - create one Model
- PUT **/admin/models/:uuid** - update one Model
- DELETE **/admin/models/:uuid** - delete one Model

For manage Attributes:

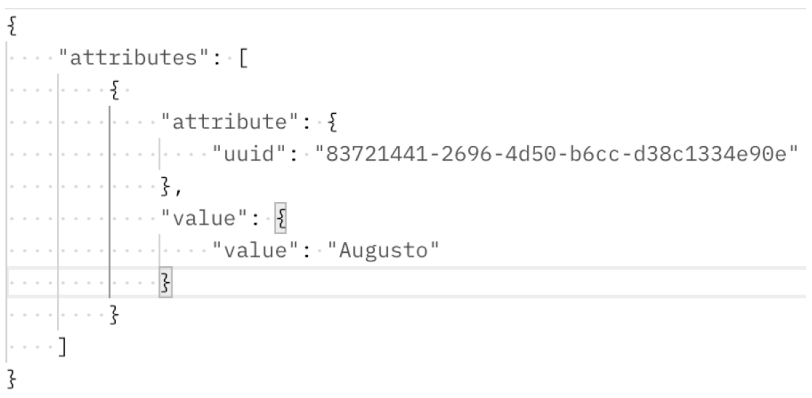
- GET **/admin/attributes** - get all Attributes
- GET **/admin/attributes/:uuid** - get one Attribute
- POST **/admin/attributes** - create one Attribute
- PUT **/admin/attributes/:uuid** - update one Attribute
- DELETE **/admin/attributes/:uuid** - delete one Attribute

4.2.1.3.3 Client Routes

Authenticated
For Registers:

- GET **/client/apis/:uuidApi/models/:uuidModel/registers** - get all registers of an API and specific Model
- POST **/client/apis/:uuidApi/models/:uuidModel/registers** - create one register of an API and specific Model
- GET **/client/apis/:uuidApi/models/:uuidModel/registers/:uuidRegister** - get one register of an API and specific Model
- PUT **/client/apis/:uuidApi/models/:uuidModel/registers/:uuidRegister** - update one register of an API and specific Model
- DELETE **/client/apis/:uuidApi/models/:uuidModel/registers/:uuidRegister** - delete one register of an API and specific Model

The Figure 14 captures a snapshot from the Postman tool (34), which was used to test the responses of routes created. In that case, the route body sent is of the POST route above sending the value ’Augusto’ as value for the attribute ’Name’ for Sellers model.



```json
{
    "attributes": [
        {
            "attribute": {
                "uuid": "83721441-2696-4d50-b6cc-d38c1334e90e"
            },
            "value": {
                "value": "Augusto"
            }
        }
    ]
}
```

Figure 14 – Request body of the creation register route.

4.2.1.4 Models and Hooks

4.2.1.4.1 Models

The following models were created to support the prototype:

User: Used to log in and link the modeling performed on the WEB platform
API: Used to be integrated through the proposed integration token.
Model: Used to group records and represent an entity.
Attribute: Used to represent attributes of entities.
Value: Used to store values belonging to a record

4.2.1.4.2 Hooks

Some hooks are executed, the most important of which are the generation of uuids before to the creation of each record of the models and the generation of the APIs integration token.

4.2.1.5 Database Diagram

The Figure 15 captures a snapshot from the DBeaver tool (14), which was used to manage and analyze the structures and data created to support the prototype.

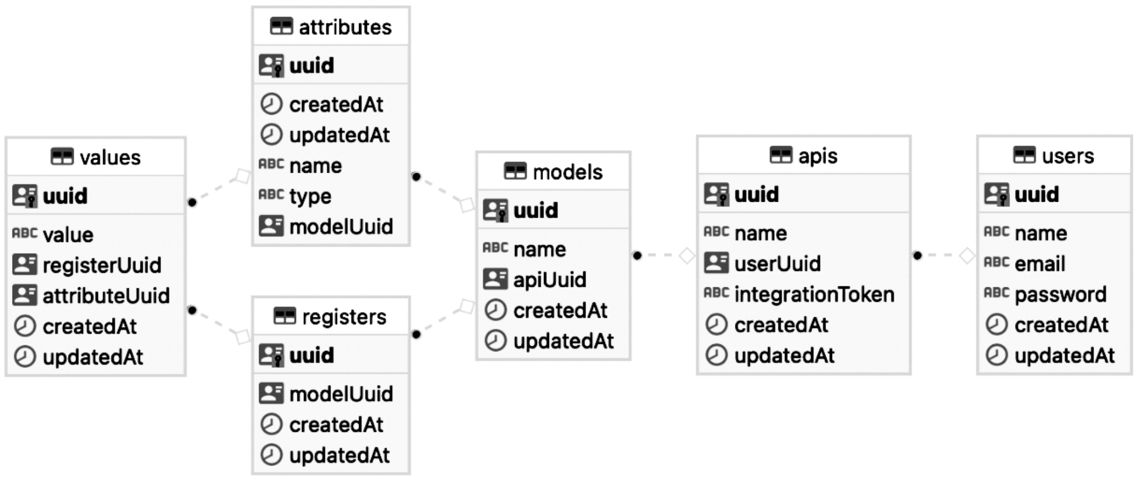


Figure 15 – Snapshot of database created.

4.2.1.6 Authentication

Admin We use JWT generated in the login route to send login and password credentials after signup in the Admin authentication middleware. After receiving it (the JWT), we include it in all requests as part of the Authorization key header.

JSON Web Token (JWT) is an open standard that defines a compact way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret or a public/private key pair (4).

Client We double-check the request in the Client authentication middleware. If the identifier (uuid) of the API in the URL exists, we check whether the integration token belongs to the requested and found API.

4.2.1.7 Data security assurance

We made some arrangements to ensure the isolation and security of data belonging to a single API, since we share the same schema as well as the same tables in our initial implementation. For example, we avoid problems during the integration with an API from a mobile client application and possible attacks to get records belonging to other models already created in the database.

In addition to other verifications already mentioned, after we ensure that the token belongs to the API requested (as mentioned in the section of Authentication), we check whether the model coming as a parameter specified in the URL belongs to the requested API, and in the case of record manipulation, we search if the record belongs to the requested model (that also comes in the URL).

4.2.2 Frontend Development

The frontend project is a supplement to the prototype, and the main idea is to avoid modeling the APIs through requests, as in Postman (34).

We can easily manage all of the modeling of APIs, Models, Attributes, and Relationships using the WEB interface, in addition to consulting the integration documentation available in the domain itself (as mentioned in the Documentation section).

4.2.2.1 Technologies

Among the main technologies used in the frontend project (but not limited to) are:

ReactJS React is a JavaScript library for building user interfaces, building encapsulated components that manage their own state and then compose them to make complex UIs (30).

Formik Formik is the world's most popular open source form library for React and React Native. Its top goals are getting values in and out of form state, validation and error messages and handling form submission (18).

Redux React Redux is the official React UI bindings layer for Redux. It lets React components read data from a Redux store, and dispatch actions to the store to update state (35).

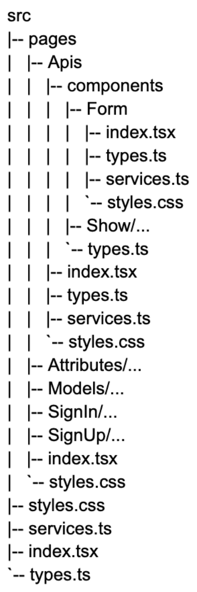
4.2.2.2 Patterns

The Figure 16 represents the organization of the files for the frontend project. A custom design pattern was adopted, which I've been using recently in my personal projects, separating styles from rendering and this one from the logical part isolated in 'services'.

4.2.2.3 Features

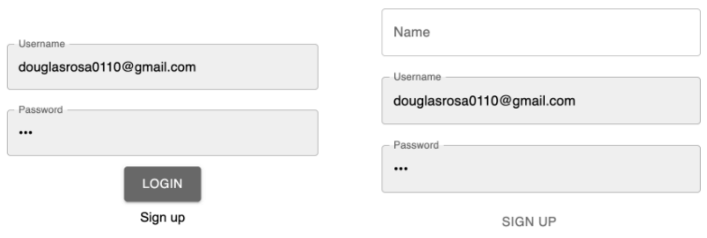
First and foremost, the platform's registration and access functionalities for new users were implemented. Following that, the functionality for creating, reading, editing, and deleting tables that reference the APIs, Models, and Attributes was implemented. We chose some screenshots to demonstrate and give an idea of how the top features were visually implemented.

Figure 17 shows the screenshot picked from the Login page. Both pages appear if the user is not authenticated and are used to register new users and for authentication.



```
src
|-- pages
|   |-- Apis
|   |   |-- components
|   |   |   |-- Form
|   |   |   |   |-- index.tsx
|   |   |   |   |-- types.ts
|   |   |   |   |-- services.ts
|   |   |   |   `-- styles.css
|   |   |   |-- Show/...
|   |   |   `-- types.ts
|   |   |-- index.tsx
|   |   |-- types.ts
|   |   |-- services.ts
|   |   `-- styles.css
|   |-- Attributes/...
|   |-- Models/...
|   |-- SignIn/...
|   |-- SignUp/...
|   |-- index.tsx
|   `-- styles.css
|-- styles.css
|-- services.ts
|-- index.tsx
`-- types.ts
```

Figure 16 – Files tree of the Frontend.



Username: douglasrosa0110@gmail.com
Password: •••

Name
Username: douglasrosa0110@gmail.com
Password: •••

Figure 17 – Screenshot of the Signin and Signup pages.

The Figure 18 shows the sidebar picked from the home page, after signed in. It comes with APIs, Models, Attributes items to be clicked. All clicks result the listing of all data created by the current user for that item.

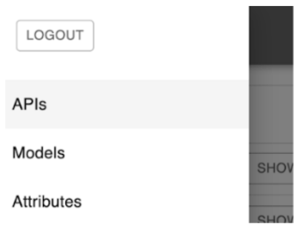
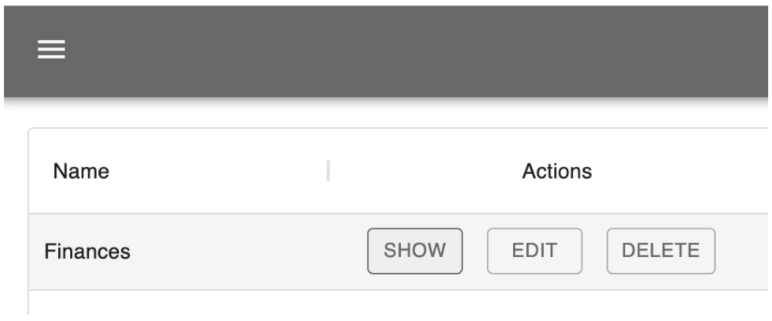


Figure 18 – Screenshot of the Sidebar.

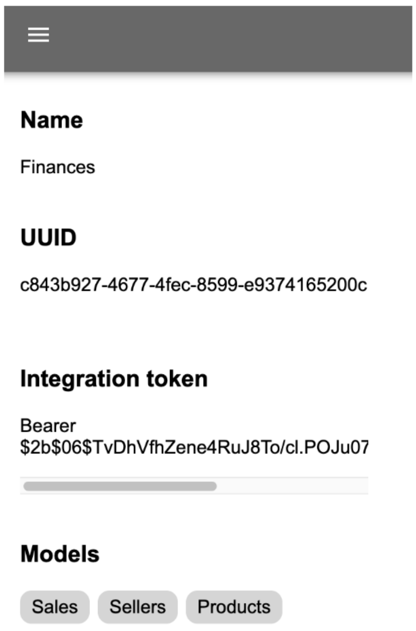
The Figure 19 shows the listing of APIs created by the current user. All features like APIs, Models, and Attributes listings essentially follow the same layout, with a single difference in the information presented. All of them allow you to view, edit, and delete individual records.



Name	Actions
Finances	SHOW EDIT DELETE

Figure 19 – Screenshot of the APIs listing.

The Figure 20 shows a detailed API with relevant information about its integration. These information - UUID and the Integration token - are used by client mobile app in the request or in the SDK library offered.



Name

Finances

UUID

c843b927-4677-4fec-8599-e9374165200c

Integration token

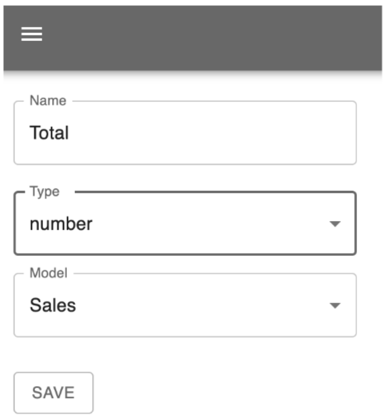
Bearer
$2b$06$TvDhVfhZene4RuJ8To/cl.POJu07

Models

Sales Sellers Products

Figure 20 – Screenshot of an API detailed.

The Figure 21 shows an attribute being edited. We provide the fields **Name** to better identify the field (not just by uuid), **Type** to refer to the type of the field, which at the time of writing this document had the options of ’number’, ’boolean’ and ’text’, and finally **Model** to indicate that this attribute belongs to specific model only within those created by the current user.


Name: Total

Type: number

Model: Sales

Figure 21 – Screenshot of Attribute form.

4.2.2.4 Documentation

We provide a public documentation page linked to the WEB platform where it is possible to consult how to make requests, as well as specifications of response protocols and possible status returned, allowing direct access by API version by changing the value of the version parameter assigned in the browser's URL, such as /doc?version=1.

4.3 CASE STUDY: MOBILE CLIENT DEVELOPMENT

The mobile application validates the prototype's applicability because it is primarily intended to serve this market niche in order to facilitate the backend development process, particularly for those professionals who spend the majority of their time or have their primary focus on the development of mobile applications.

4.3.1 App Development

This mobile application aims to build a system that can carry out sales that are linked to sellers and products. We did this by implementing the following models' creation, editing, and deletion screens in the WEB platform:

- Sales containing attributes such as Total, Seller (by relationship belongsTo), and Products (by relationship hasMany);
- Seller attributes such as Name;
- Product attributes such as Description, Price, and Stock;

4.3.1.1 Technologies

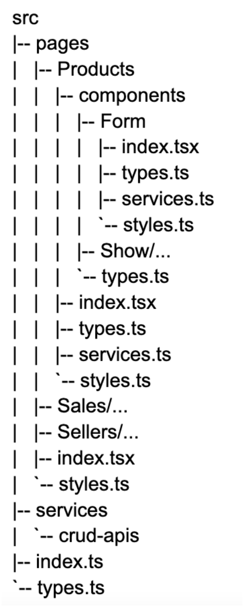
Among the main technologies used in the mobile project (but not limited to) are:

React Native
React Native is an open-source UI software framework created by Meta Platforms, Inc. It is used to develop applications for Android, Android TV, iOS, macOS, tvOS, Web, Windows and UWP by enabling developers to use the React framework along with native platform capabilities (29).

Formik
As already explained in this section 4.2.2.1

4.3.1.2 Patterns

The Figure 22 represents the organization of the files for the app client project. The same pattern is used in the Admin project because they are both frontend projects that use similar frameworks. The SDK used to make easy the integration is located in the project and is being categorized as a 'service' (crud-apis) that will be later locally imported rather than an external importation, such as an NPM package (21).



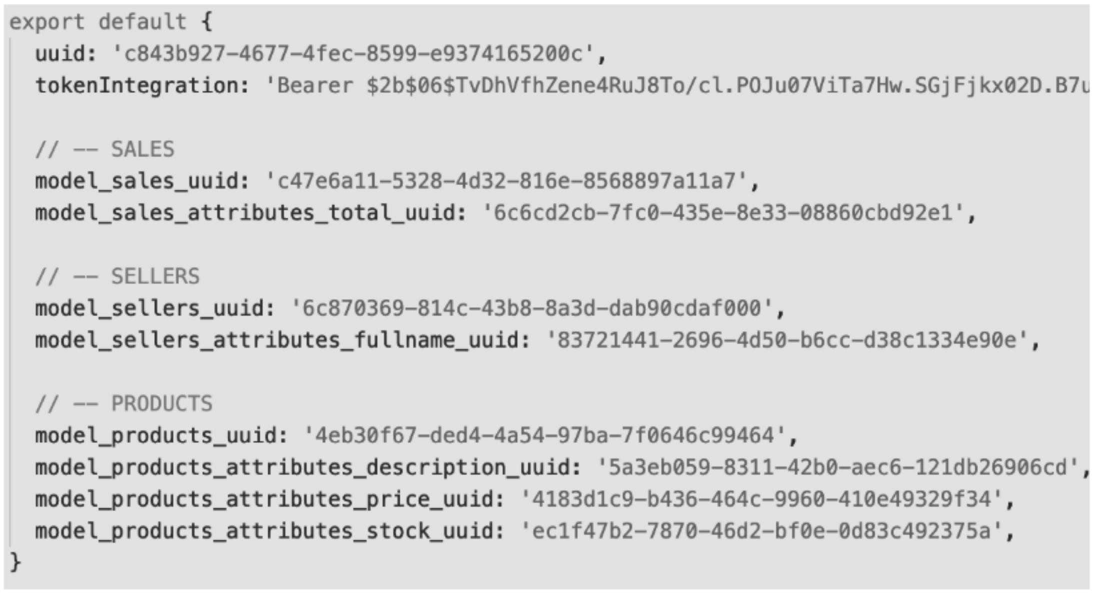
```
src
|-- pages
|  |-- Products
|  |  |-- components
|  |  |  |-- Form
|  |  |  |  |-- index.tsx
|  |  |  |  |-- types.ts
|  |  |  |  |-- services.ts
|  |  |  |  `-- styles.ts
|  |  |  |-- Show/...
|  |  |  `-- types.ts
|  |  |-- index.tsx
|  |  |-- types.ts
|  |  |-- services.ts
|  |  `-- styles.ts
|  |-- Sales/...
|  |-- Sellers/...
|  |-- index.tsx
|  `-- styles.ts
|-- services
|  `-- crud-apis
|-- index.ts
`-- types.ts
```

Figure 22 – Files tree of the App.

4.3.1.3 SDK Implementation

The developed SDK consists of methods exported for use in JavaScript projects. Among the functionalities are methods for getting APIs, Models, and Records and was built with object orientation. This SDK also includes interfaces for use in Typescript projects. The Figure 23 shows a possible way suggested to map identifiers got from the WEB platform to use in the app client integration.

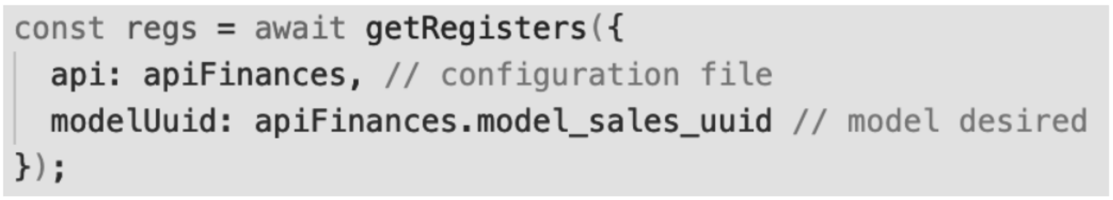
This approach is very useful to make easier the manipulation of data, because the platform only provides to us the UUIDs as keys to manipulate everything we have created on the WEB platform.



```
export default {
  uuid: 'c843b927-4677-4fec-8599-e9374165200c',
  tokenIntegration: 'Bearer $2b$06$TvDhVfhZene4RuJ8To/cl.POJu07ViTa7Hw.SGjFjkx02D.B7u

  // -- SALES
  model_sales_uuid: 'c47e6a11-5328-4d32-816e-8568897a11a7',
  model_sales_attributes_total_uuid: '6c6cd2cb-7fc0-435e-8e33-08860cbd92e1',

  // -- SELLERS
  model_sellers_uuid: '6c870369-814c-43b8-8a3d-dab90cdaf000',
  model_sellers_attributes_fullname_uuid: '83721441-2696-4d50-b6cc-d38c1334e90e',

  // -- PRODUCTS
  model_products_uuid: '4eb30f67-ded4-4a54-97ba-7f0646c99464',
  model_products_attributes_description_uuid: '5a3eb059-8311-42b0-aec6-121db26906cd',
  model_products_attributes_price_uuid: '4183d1c9-b436-464c-9960-410e49329f34',
  model_products_attributes_stock_uuid: 'ec1f47b2-7870-46d2-bf0e-0d83c492375a',
}
```

Figure 23 – Suggestion of configuration of uuids.

The Figure 24 shows the integration with the Finances API to get the registers of the Sales model configured already using the previously prepared UUIDs mapping.

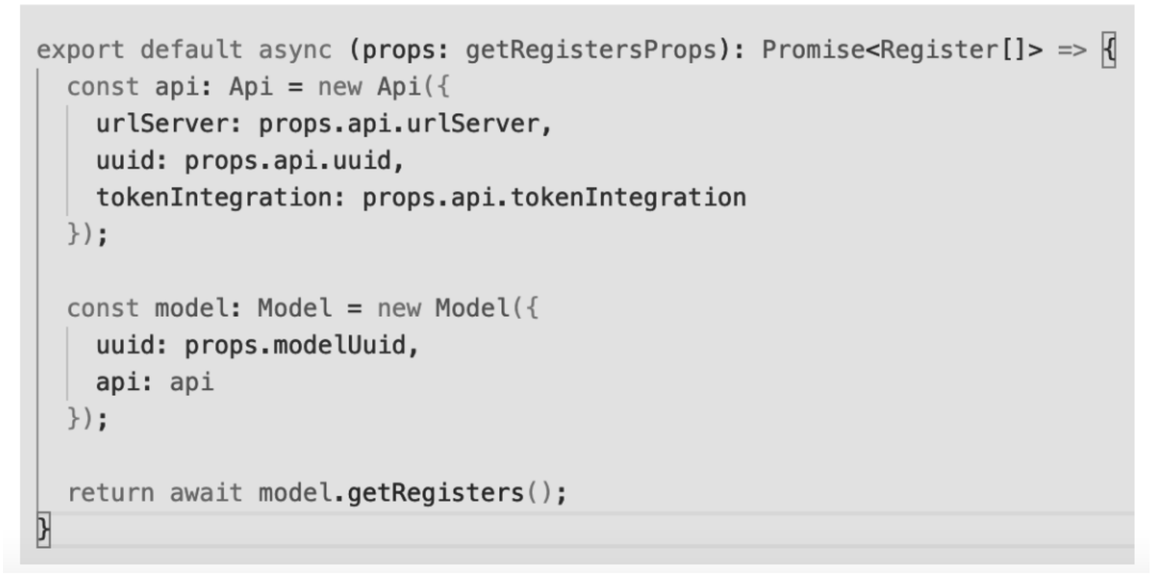


```
const regs = await getRegisters({
  api: apiFinances, // configuration file
  modelUuid: apiFinances.model_sales_uuid // model desired
});
```

Figure 24 – Getting registers of the Sales model from Finances API.

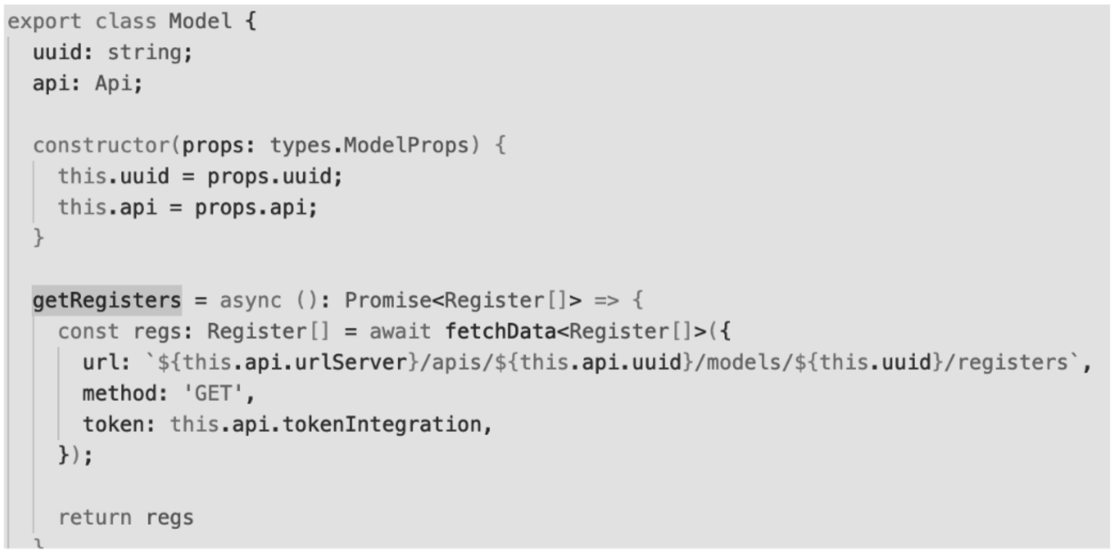
The Figure 25 shows the implementation of the method exported that get all registers from the model passed as a parameter. It is important to note that the API is first instantiated and then used as a parameter for handling and listing records related to the desired model that belongs to the instantiated API.

The Figure 26 shows the implementation of the Model class to be instantiated passing the UUID identifier and the API that it belongs as well the public method to get registers of this model using the fetchData method that is basically the customization of the fetch method of the Javascript.



```
export default async (props: getRegistersProps): Promise<Register[]> => {
  const api: Api = new Api({
    urlServer: props.api.urlServer,
    uuid: props.api.uuid,
    tokenIntegration: props.api.tokenIntegration
  });

  const model: Model = new Model({
    uuid: props.modelUuid,
    api: api
  });

  return await model.getRegisters();
}
```

Figure 25 – Getting registers implementation method.



```
export class Model {
  uuid: string;
  api: Api;

  constructor(props: types.ModelProps) {
    this.uuid = props.uuid;
    this.api = props.api;
  }

  getRegisters = async (): Promise<Register[]> => {
    const regs: Register[] = await fetchData<Register[]>({
      url: `${this.api.urlServer}/apis/${this.api.uuid}/models/${this.uuid}/registers`,
      method: 'GET',
      token: this.api.tokenIntegration,
    });

    return regs
```

Figure 26 – Model class and the get registers public method.

4.3.1.4 Features

We were able to create, edit, and delete all models using the developed SDK in the client App.

The Figure 27 shows the screen that list all sales created from the mobile app. Basically is the data got from the exported SDK method getRegisters() passing the API of Finances configured and the model of sales and in the Figure 28 shows sidebar to navigate between screens of Sales, Products and Sellers as described in 4.3.1

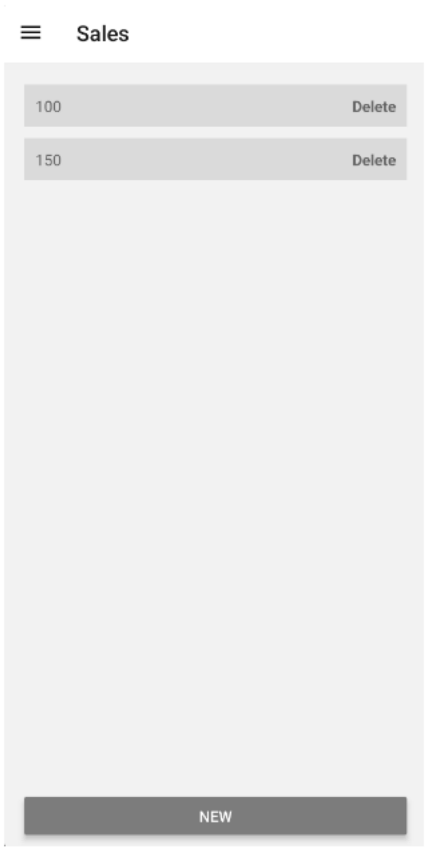


Figure 27 – Listing Sales registers.

The Figure 29 shows the creation of a new register of a Product model in the mobile client app through the form interface offering 3 fields for this model: a Description, a Price and the quantity acquired for the stock called Stock.

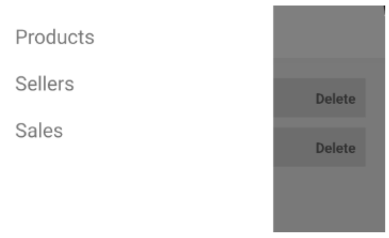


Figure 28 – Sidebar of the app.

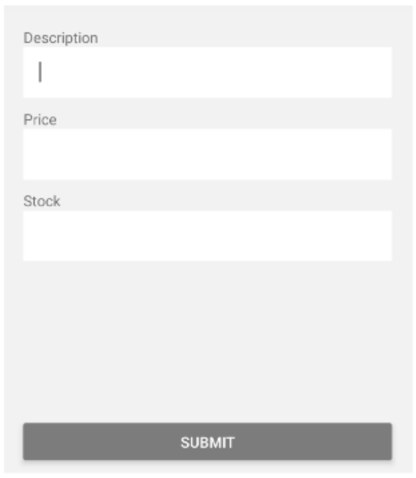


Figure 29 – Form for new product.

4.4 CHAPTER’S FINAL REMARKS

This chapter presented the implementation characteristics of the Backend project (a), where we have the core features for the proposed prototype, as well as the Frontend project (b), where we have an API configuration interface for mobile developers, and finally the implementation aspects of a hybrid language in a mobile project (c) using the integration SDK, which is also implemented in this project. As realized and mentioned, we chose an idea of product in the finances area for basically make sales, where we have the best scenario to demonstrate the prototype’s functionalities for the purpose of prototype validation.

5 CONCLUSIONS

This project started with the study of the literature considering different tools used to design APIs and through projects with possibilities of real-time data consumption in mobile applications. The literature and the concepts were useful to reach the results expected in the implementation.

The complexity of simplifying the structuring as much as possible so that we do not attach responsibilities that could be delegated to the frontend is one of the main challenges faced in developing the prototype.

We believe that this project has its space in the market and it is better suited for small businesses, such as startups, at least for this prototype, because the scalability of the data generated on the platform necessitates some treatment to avoid the problem of big data and performance in the searches possible innumerable relationships as the project grows.

We were unable to implement and document certain features until the end of this document, and these features are currently under development. Among them are the characteristics of model relationships and the specific business rules for each attribute within a model.

We are aware that the state of the art of our prototype does not limit us to stopping here and it is only a part of what we intend to do, but it can now become a marketable product continuing with a pre-incubation process in the university program and/or even as an open source project.

REFERENCES

1 AGILITY. **What is Agile**. [S.l.: s.n.]. Available from: <https://agility.im/frequent-agile-question/what-is-agile/>.

2 ALLAMARAJU, Subbu. **Restful web services cookbook: solutions for improving scalability and simplicity**. [S.l.]: " O'Reilly Media, Inc.", 2010.

3 APPMACHINE. **Tour on AppMachine**. [S.l.: s.n.]. Available from: <https://www.appmachine.com/tour/>.

4 AUTH0. **JWT**. [S.l.: s.n.]. Available from: <https://jwt.io/introduction>.

5 AWS. **What is Mobile Application Development?** [S.l.: s.n.]. Available from: <https://aws.amazon.com/pt/mobile/mobile-application-development/>.

6 BACK4APP. **What Is Parse and Back4App**. [S.l.: s.n.]. Available from: <https://www.youtube.com/watch?v=vLfIDscFgQI&list=PL_lJrbgUtzded_bF8KVl_puWZ-zDCLw7R&t=4s>.

7 BACKENDLESS. **Backendless Corp**. [S.l.: s.n.]. Available from: <https://www.linkedin.com/company/backendless-corp/about/>.

8 ________. **Creating Database Views**. [S.l.: s.n.]. Available from: <https://youtu.be/tsj45-uVdkQ?list=PLWRqDbbT5r9DXiks8mZmp_84W6cRymoIc>.

9 BIEHL, Matthias. **RESTful Api Design**. [S.l.]: API-University Press, 2016. v. 3.

10 BUFFARDI, Kevin; ROBB, Colleen; RAHN, David. Tech startups: realistic software engineering projects with interdisciplinary collaboration. **Journal of Computing Sciences in Colleges**, Consortium for Computing Sciences in Colleges, v. 32, n. 4, p. 93–98, 2017.

11 CNI. **Terceirização de serviços e atividades é estratégica para a indústria no Brasil**. [S.l.: s.n.]. Available from: <https://www.portaldaindustria.com.br/industria-de-a-z/terceirizacao/#e-boa-para-o-brasil>.

12 CONFIGURE.IT. **App Development, Re-invented with Configure.IT**. [S.l.: s.n.]. Available from: <https://www.youtube.com/watch?v=78elHbbKC-8&t=91s>.

13 CONFIGURE.IT. **Configure.It**. [S.l.: s.n.]. Available from: <https://www.configure.it/>.

14 DBEAVER. **DBeaver**. [S.l.: s.n.]. Available from: <https://dbeaver.io/>.

15 DESIGNING FOR SCALE. **Business Rules Must Be Enforced by the API**. [S.l.: s.n.]. Available from: <https://designingforscale.com/business-rules-must-be-enforced-by-the-api/>.

16 ED-DOUIBI, Hamza et al. EMF-REST: generation of RESTful APIs from models. In: PROCEEDINGS of the 31st Annual ACM Symposium on Applied Computing. [S.l.: s.n.], 2016. p. 1446–1453.

17 FORBES. **What Is A Startup?** [S.l.: s.n.]. Available from: <https://www.forbes.com/advisor/investing/what-is-a-startup/>.

18 FORMIUM. **Formik**. [S.l.: s.n.]. Available from: <https://formik.org/>.

19 GEEKSFORGEEKS. **REST API Architectural Constraints**. [S.l.: s.n.]. Available from: <https://www.geeksforgeeks.org/rest-api-architectural-constraints/>.

20 GIGS. **GIGS: A Day in the Life of: A MOBILE APP DEVELOPER**. [S.l.: s.n.]. Available from: <https://www.youtube.com/watch?v=ulSxrbaj5rs>.

21 GITHUB, INC. **NPM packages**. [S.l.: s.n.]. Available from: <https://www.npmjs.com/>.

22 GOOGLE. **Google Docs**. [S.l.: s.n.]. Available from: <https://workspace.google.com/products/docs/>.

23 ________. **Google Meet**. [S.l.: s.n.]. Available from: <https://workspace.google.com/products/meet/>.

24 HEIKKINEN, Eetu. Full Stack Development, 2022.

25 HELGASON, Arnar Freyr. **Performance analysis of Web Services : Comparison between RESTful &amp; GraphQL web services**. [S.l.: s.n.], 2017. p. 66.

26 JGRAPH. **Diagrams.net**. [S.l.: s.n.]. Available from: <https://www.diagrams.net/>.

27 LINKEDIN. **Reasons to hire a Fullstack Developer in a Startup Environment.** [S.l.: s.n.]. Available from: <https://www.linkedin.com/pulse/reasons-hire-fullstack-developer-startup-environment-shane-sale/>.

28 MEDIUM. **How Node JS Middleware works?** [S.l.: s.n.]. Available from: <https://selvaganesh93.medium.com/how-node-js-middleware-works-d8e02a936113>.

29 META PLATFORMS, INC. **React Native**. [S.l.: s.n.]. Available from: <https://reactnative.dev/>.

30 ________. **ReactJS**. [S.l.: s.n.]. Available from: <https://reactjs.org/>.

31 MICROSOFT. **Typescript**. [S.l.: s.n.]. Available from: <https://www.typescriptlang.org/>.

32 OPENJS FOUNDATION. **NodeJS**. [S.l.: s.n.]. Available from: <https://nodejs.dev/learn>.

33 PARSE.COM. **Parse.com**. [S.l.: s.n.]. Available from: <http://parseplatform.org/>.

34 POSTMAN, INC. **Postman**. [S.l.: s.n.]. Available from: <https://www.postman.com/>.

35 REDUX. **Redux**. [S.l.: s.n.]. Available from: <https://react-redux.js.org>.

36 RESTFULAPI. **What is REST**. [S.l.: s.n.]. Available from: <https://restfulapi.net/>.

37 RIVERO, José Matıas et al. MockAPI: an agile approach supporting API-first web application development. In: SPRINGER. INTERNATIONAL Conference on Web Engineering. [S.l.: s.n.], 2013. p. 7–21.

38 RODRIGUEZ, Alex. Restful web services: The basics. **IBM developerWorks**, v. 33, p. 18, 2008.

39 SEQUELIZE. **sequelize**. [S.l.: s.n.]. Available from: <https://sequelize.org>.

40 ________. **sequelize-ts**. [S.l.: s.n.]. Available from: <https://www.npmjs.com/package/sequelize-typescript>.

41 SIXT CARRIERE. **A day as an iOS Developer at SIXT**. [S.l.: s.n.]. Available from: <https://www.youtube.com/watch?v=aLccZMTLsmQ>.

42 THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **Postgres**. [S.l.: s.n.]. Available from: <https://www.postgresql.org/>.

43 WEINER, Andrew; REMER, Rory; REMER, Pam. Career Plateauing: Implications for Career Development Specialists. **Journal of Career Development**, v. 19, n. 1, p. 37–48, 1992. Available from: <https://doi.org/10.1177/089484539201900104>.

44 XANO. **What is Xano?** [S.l.: s.n.]. Available from: <https://www.youtube.com/watch?v=ng0GiVYOFnc>.