



1. Presentation of enterprise

1.1 About enterprise

Smokio is a French company that develops and sells connected electronic cigarettes and innovative apps. It was founded in August 2013 in Paris, France.



Figure 1 - <http://www.smokio.com/>

The idea of our product came from our founder. Once as a heavy smoker, he knows well about smokers. He knows well the terrible harm that smoking brings to our health, and the spending of economy.

We want to help people to quit smoking, live a better and smarter life by bringing technology to everyday life.

1.2 Organisation

Steve Anavi: President

Alexandre: CEO

Benjamin Prieur: Lead Developer iOS

Ivan Fernando Soriano: Android Developer

Safia Boudia: Operations Manager

1.3 Product

Overall

We spent 6 months working on the hardware and software to create the first smart

and connected vaporizer.



Figure 2 - Overall: Smokio and Smokio App

Hardware

Unlike basic vaporizers, Smokio lets you monitor your real time intake by measuring number of puffs and nicotine levels. Smokio is a smart, but also beautiful vaporizer. The inside of Smokio, there is a circuit board designed to control the output power of vaporizer, collect user activities and communicate with our mobile application by using Bluetooth 4.0.

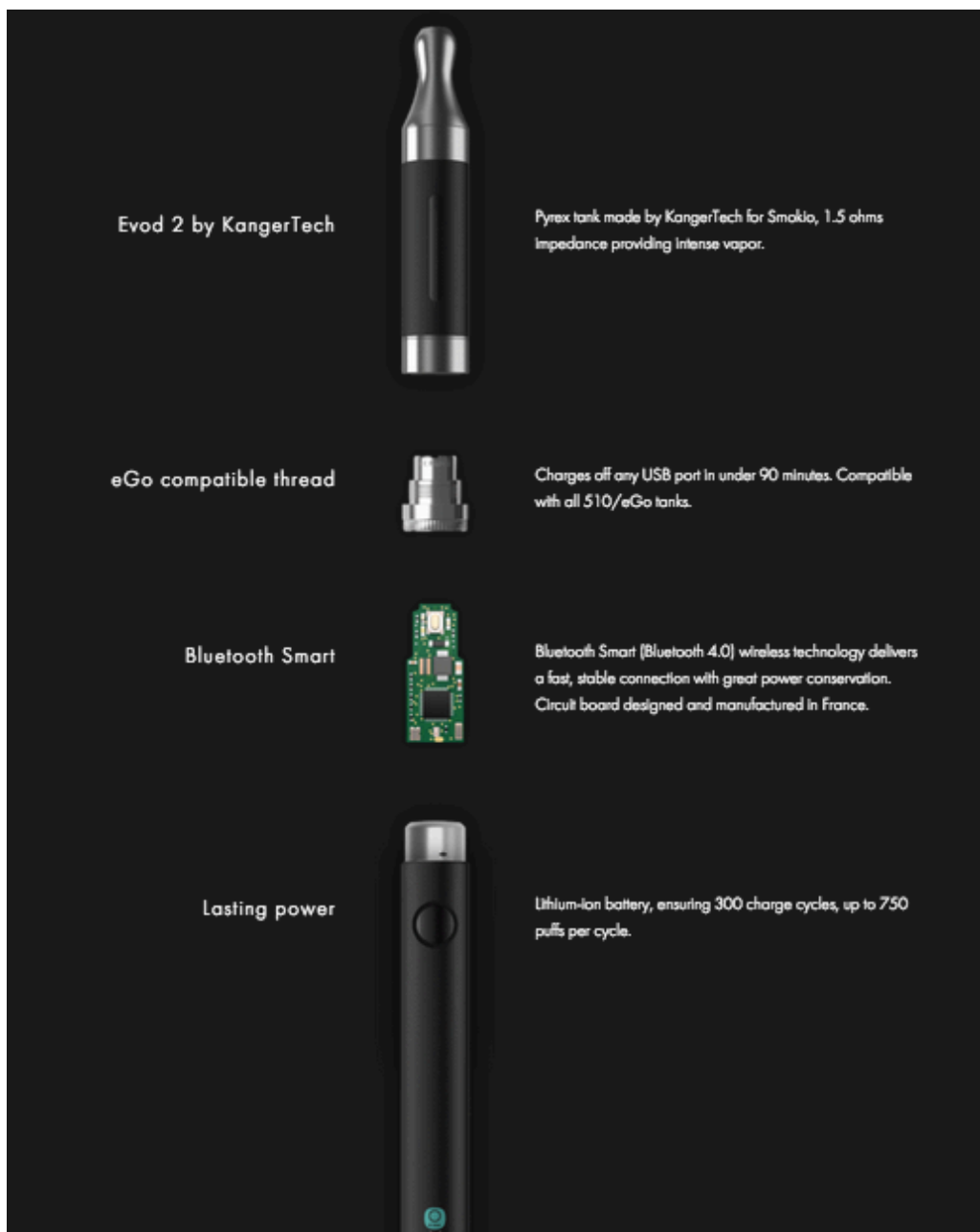


Figure 3 - Smokio: Smart vaporizer

Software

Mobile application

The Smokio app tracks your everyday vaping activity. It's full of quantified features to help you monitor your activity and improve your vaping experience.

- Track your usage

It shows how well you are doing thanks to insightful reports and customized notifications. You can then set up your milestones at your own pace.

- **Measure the benefits.**

To keep you motivated, Smokio keeps track of the financial savings you make every day! For the first time, you're able to directly reap benefits of your efforts.

- **Assemble your team.**

You can interact with your friends, compare stats and move together as a team. The Smokio app taps Facebook and Google+.

- **Remote control**

The Smokio app allows to control your Smokio device directly from your smartphone: check the level of battery, set puff limits or adjust the voltage for more/less intense puffs.

- **Connect Smokio with your activity trackers**

You can compare your Smokio stats with your activity such as the number of calories or the weight. The Smokio app allows you to connect external apps (Jawbone, Withings or Fitbit) to retrieve your data.

Observer application

Observer is a SaaS(Software as a service) application who permits our users to share their data with the professional researchers who might find relevant to access user's data. It is currently in beta version.

Observer has basic function of a SaaS application, such as Subscription, integration of the third party payment platform, access control, authentication, etc.

1.4 Belief

We believe in building quality experiences for our customers through elegance and simplicity while maintaining high level of security and privacy. These principles form the cornerstone behind our work, and reinforces our belief that that the best technology makes you smarter, puts you in control, and gives you access to the information you need.

2. Expected contributions

According objectives defined in my internship contract, my expected contributions will include two aspects of technical and cooperation. On the technical side, I need to use my expertise to the company's product development. In terms of cooperation, I need to integrate myself into our internal development team, as well as external development teams to communicate, and thus advance the development of our product.

2.1 Technique

Web services:

- With the new features to be added and improvements, on the server side, I need to design and implement some new APIs to collect data from multiple mobile platforms (currently iOS and Android), and in response to the request of Smokio App on the mobile terminals.
- In the back end, in case of need to use a third-party API, I need to integrate them into our services, such as Stripe, Mandrill, SendWithUs, etc.
- I need to develop our own Open API, share our data interface to third-party developers to enrich our platform product features.

Web application:

As we have concerned before, we are planning to create a SaaS web application, called Observer. For this web application, I need construct a high quality and well-designed web application who can be used to manage accounts, visual display of data users and interact with existing services.

Business intelligence

As the number of users increases, user data will become more and more, in order to fully exploit the value of these data. We need a platform to acquire and process historical data. By using data mining knowledge, find some meaningful patterns, and build some models.

- Establish and improve a platform which is in charge of integration and near real-time analysis of historical data stream.

- In order to provide the function of one-to-one coach, develop and integrate an intelligent model

System Management and Maintenance

- Deploy a new version of product to the server
- Fix system bugs
- Monitor and maintain the normal operation of the server

Test

- Before release of the new features, I need to participate in test.

2.2 Collaboration

Internal communication

Technique team

- Provide reliable web service for Smokio mobile app, to meet the development requirements of mobile application, in a efficient way to communicate with our team.
- If there is any disagreement, we consult together to determine a best solution.
- Regularly reporting progress, with the agile development team management.
- Regular working meetings to determine the development plan.

Custom support

When our customer support received technical feedback from users, forwarded it to the development team. I need without affecting the progress of the development, to assist other members of the development team, solve problems for users as soon as possible.

External communication

Third party plat-form

While using a third-party service platform, we would inevitably encounter some problems. May be some technology problems, it may be that we have not integrated

their services in the right way.

Therefore, we need contact their technical support contact in time, to figure out the source of the problem and provide solutions. That will reduce our losses to a minimum.

China

Somkio are produced and assembled in China, and our application becomes available to many countries and regions, recently entering into the Chinese market.

- I will be responsible for part of the Chinese translation of the document, as well as contact with Chinese suppliers and customers.
- Smokio APP will be launched in some principle Android App Stores in China.

3. Bibliographic study

- [1] Hartl, M. 2015. Ruby on Rails Tutorial: Learn Web Development with Rails (3rd ed.). Addison-Wesley Professional.
- [2] Flanagan, D. and Matsumoto, Y. 2008. The Ruby programming language. O'Reilly.
- [3] Stripe API Reference: 2015. <https://stripe.com/docs/api>. Accessed: 2015- 04- 25.
- [4] Stripe Payments in Rails, Part 1 - Stripe Checkout: 2015.
<http://www.gotealeaf.com/blog/stripe-checkout>. Accessed: 2015- 04- 25.
- [5] Stripe Payments in Rails, Part 1 - Stripe Checkout: 2015.
<http://www.gotealeaf.com/blog/stripe-checkout>. Accessed: 2015- 04- 25.
- [6] What Is Amazon EC2? - Amazon Elastic Compute Cloud: 2015.
<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>. Accessed: 2015- 04- 25.
- [7] Getting started · Bootstrap: 2015. <http://getbootstrap.com/getting-started/>.
Accessed: 2015- 05- 25.
- [8] S. Ruby, D. Thomas and D. Hansson, Agile web development with Rails 4, 4th ed. Pragmatic Bookshelf, 2013.

4. Technical solution

In this part, I will talk about our technical solution.

4.1 Hardware solution

4.1.1 Amazon web services:

About hardware environment, we use **Amazon Web Service** who offers a broad set of global compute, storage, database, analytics, application, and deployment services that help organizations move faster, lower IT costs, and scale applications. Our application servers and database servers are deployed in amazon web services.

Application Server

we are currently using an **EC2**(Elastic Compute Cloud) instance of AWS. Our application server is running on this instance.

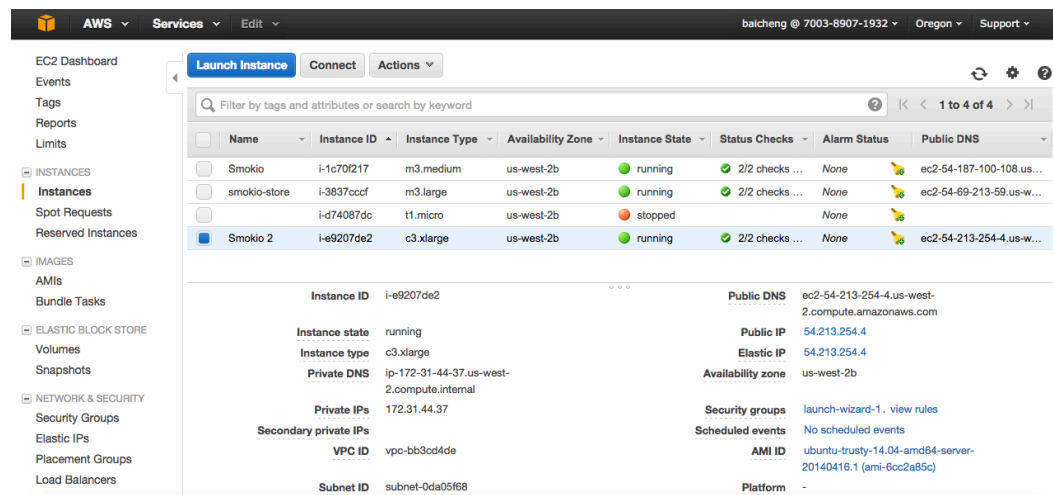


Figure 4 - EC2 Management Console

Type of instance

Considering our normal and potential requirement, we chosed an c3.xlarge instance as our principle server, for a beginning startup, I think this instance can satisfy our requirement.

Physical specifications:

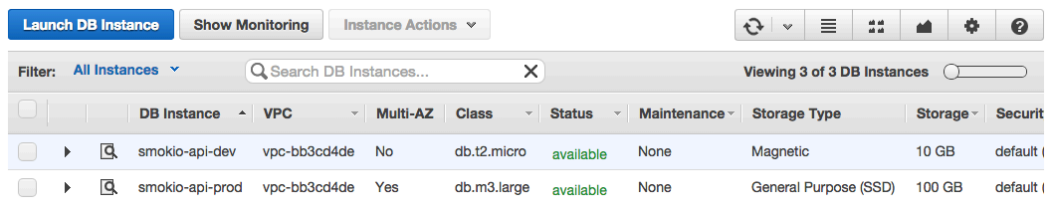
vCPU: 1 Memory(Gib): 7.5 SSD Storage(GB): 2 x 40

Software environment: Operating system: Linux(Ubuntu-trusty-14.04-amd64-server)
System Architecture: AMD x86-64

Database

Amazon offers the relational database service as well. Amazon **RDS** (Relational Database Service) makes it easy to set up, operate, and scale a MySQL, Oracle, SQL Server, or PostgreSQL database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks.

We have created two database instances, one of them for our developement and test environment, and the other one for production environment.



The screenshot shows the Amazon RDS Management Console interface. At the top, there are buttons for 'Launch DB Instance', 'Show Monitoring', and 'Instance Actions'. Below these is a search bar and a filter dropdown set to 'All Instances'. The main area displays a table of three database instances, with the first two visible:

	DB Instance	VPC	Multi-AZ	Class	Status	Maintenance	Storage Type	Storage	Security
<input type="checkbox"/>	smokio-api-dev	vpc-bb3cd4de	No	db.t2.micro	available	None	Magnetic	10 GB	default
<input type="checkbox"/>	smokio-api-prod	vpc-bb3cd4de	Yes	db.m3.large	available	None	General Purpose (SSD)	100 GB	default

Figure 5 - RDS Management Console

For these two RDS instances, we choosed **postgres** as our database engine. We can access our databases remotely, thanks to "**PG Commander**", a beautiful mac application who premits us to execute sql queries.

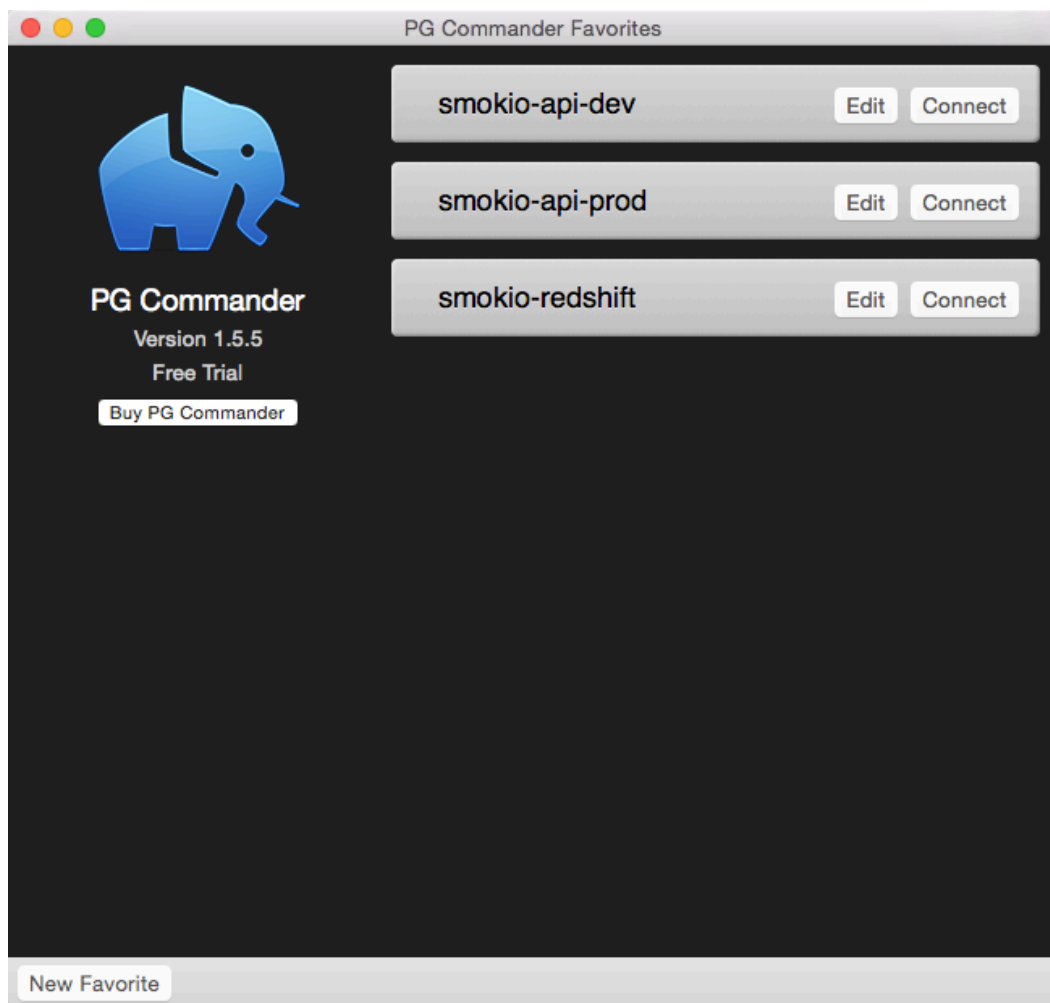


Figure 6 - PostgreSQL database GUI tool

Storage

In general, we need to store two types of files. Firstly, all firmware files, when user tries to initialize a new smokio, they need download these firmware files stored by us. Secondly, we have some files generated by our users to be stored in our server. **Amazon Simple Storage Service (S3)** is a service can be used to store and retrieve any amount of data.

All files in S3 are organized in the way of **buckets**. Generally, we use mostly two buckets "smokio-dev" and "smokio-prod" for the storages of files in two environment.

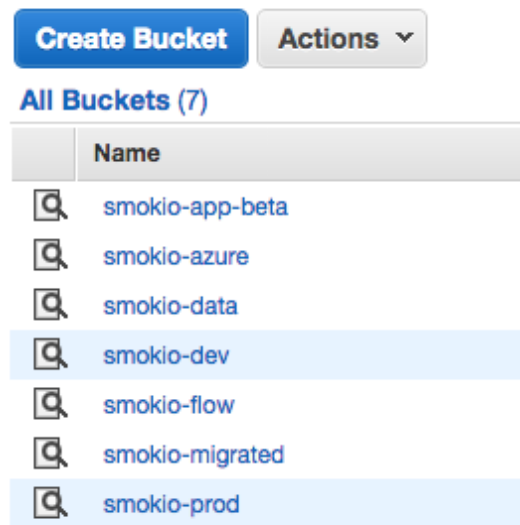


Figure 7 - S3 buckets-1

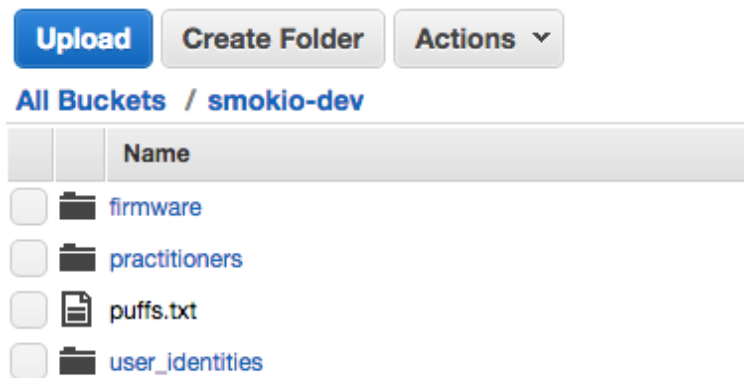


Figure 8 - S3 buckets-2

4.1.2 Smokio Device

Smokio is a microprocessor-controlled vaporizer. Its patented electronics captures all your vaping data such as puff duration and nicotine intakes, and sends it to your smartphone through Bluetooth Smart (or Bluetooth 4.0) without any additional efforts.

With microprocessor and bluetooth technique of Smokio, the acquisition of user usage data becomes possible.

4.1.3 Mobile Terminal

We develop the mobile applications for mobile terminals which are used by our users. These terminals are in charge of receiving smokio's usage data, synchronization with our backend system by using our **web services**(APIs), providing users an interface to track their uses, configure their smokio, communicate with friends, etc.

4.2 Software solution

We proposed a solution based on Ruby and "**Ruby On Rails**" framework. As we all know, Rails as an agile web development framework has been approved its value by a lot of startups, even some big companies. We can reduce the cost of development, and improve the efficiency. It's an extremely ideal technique for a startup like Smokio.



Figure 9 - Ruby On Rails logo

Without doubt, we choosed "Ruby On Rails" as our principle technique for comstructing web services and web application.

4.2.1 Web Service

Grape

[Grape](#) is a REST-like API micro-framework for Ruby. It's designed to run on Rack or complement existing web application frameworks such as Rails and Sinatra by providing a simple DSL to easily develop RESTful APIs.



Figure 10 - Grape: gem to build RESTful API

With the 3-layer model of Rails, we can design the common models who can be reused in both web services and web application. We can also use the

"**ActiveRecord**" module of Rails to encapsulate database query operations.

grape-swagger

To understand the reason why we use "[grape-swagger](#)" gem, I should explain two techniques firstly.

Swagger is a simple yet powerful representation of your RESTful API. It can offer an interface for our APIs which can help a consumer or other developers to understand and interact with the remote service with a minimal amount of implementation logic.

Swagger UI is a dependency-free collection of HTML, Javascript, and CSS assets that dynamically generate beautiful documentation and sandbox from a Swagger-compliant API.

The grape-swagger gem is able to provide an autogenerated documentation for your Grape API. The generated documentation is Swagger-compliant, meaning it can easily be discovered in Swagger UI.

So with all these tools, we can easily build our web services and provide an user-friendly interface to mobile developers in our team.

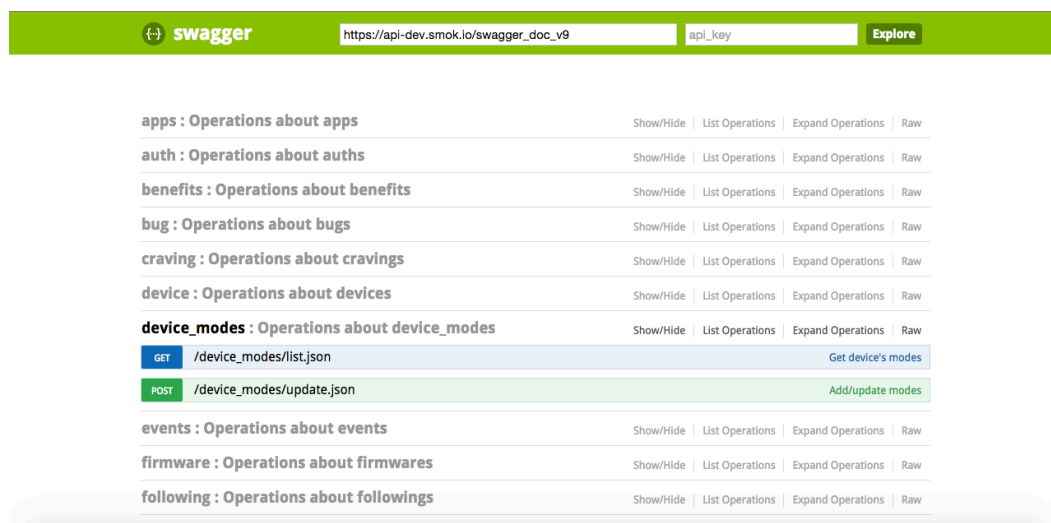


Figure 11 - Swagger: gem to generate API documentation

4.2.2 Observer application

Observer is a typical web application to be realised by "Ruby On Rails". Meanwhile, it's a typical SaaS platform as well.

For the beginning, I will use my knowledge of **Software Requirements Analysis** to

define our system specifications.

In the backend, we use **Ruby On Rails** to build our object model, business logic, REST-ful routes, html templates. We will follow Rails' **MVC pattern**. In model layer, we reuse the models with web services. In controller layer, we will implement all business logics we need, such like authentication, access control, subscription, third party payment, and so on.

In addition , we need to integrate third party services into our SaaS platform, such as **Stripe**(payment platform), **SendWithUs**(invoice generator). By using these existing services and framework, we will realize the fundamental functions of SaaS platform.

4.3 Technical architecture

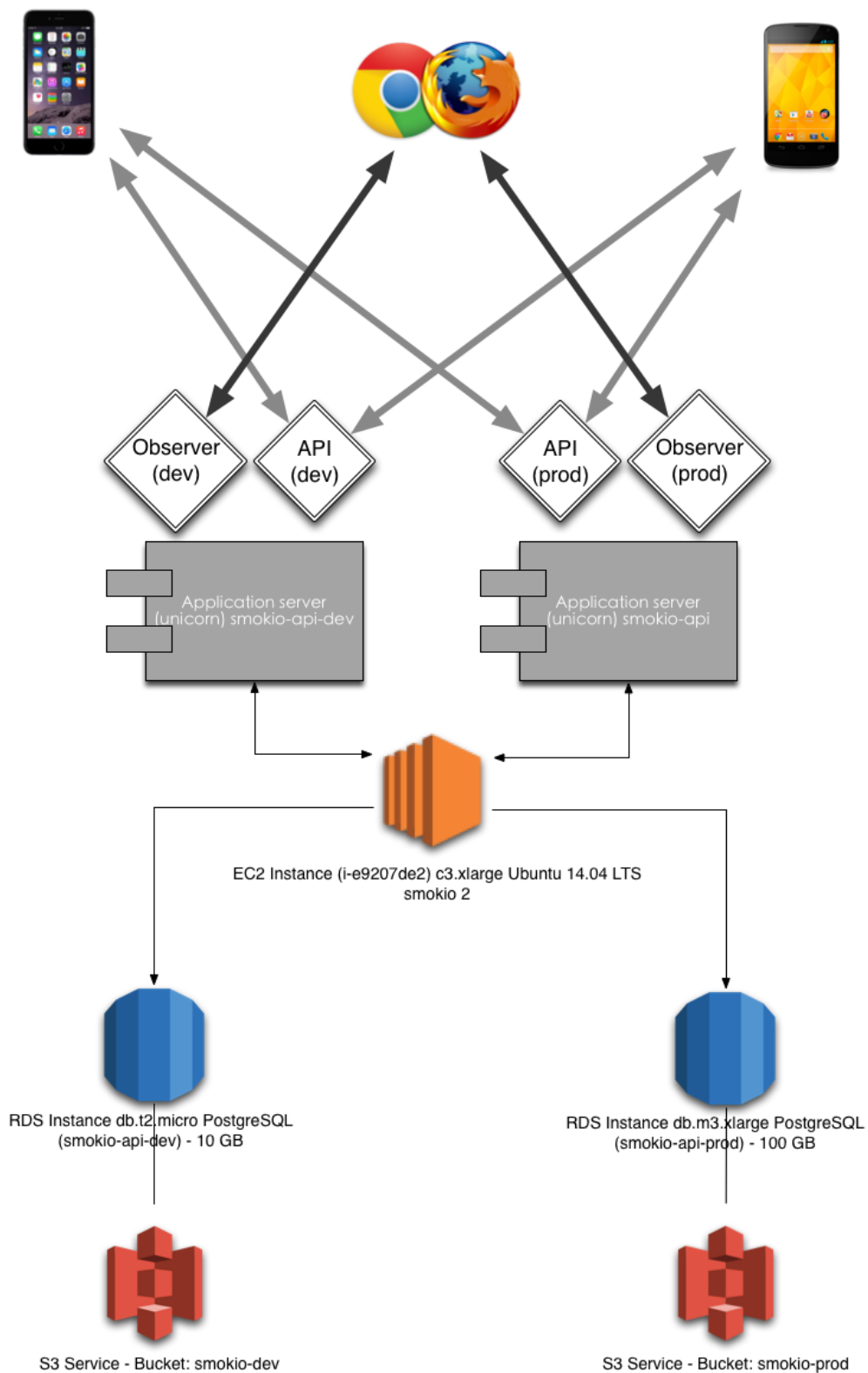


Figure 12 - Technical architecture

5 Specification

In this chapter, I will analyse the requirement specification of this project.

5.1 Presentation of projet

For the part of backend, to cope with the development of product and the introduction of new features, we need to develop new web services in the back-end to support the mobile development.

5.1.1 Context of projet

For this moment, my job involves primarily two parts. First of all, the developemnt of **backend** to support **Smokio app**. Secondly, construct an **observer** website to supply the **Saas service**.

a) Backend

We have already 6 version of **web services**. But as an ambitious and dynamic startup, we iterate frequently our product development. So, my principle task is to maintain and develop **web services**.

To develop more powerful **firmware**, in the current version of **firmware**, we have added some new commands to capture the temperature and power. Along with the developement of new **firmware**, **Smokio app** need do some change to make use of the new features of **firmware** and adapt to the change of **firmware**. All of these, we need backend supply **web services** that can response the request of **mobile application**.

To support the development of each version of **Smokio app**, I need manage different versions of **web service**, create a new version of web service and can't affect the other version.

If we want to add a new **feature**, for example, "management of different configuration mode of device". I will create a group of web services to implement this feature: `device_mode/add.json`, `device_mode/remove.json`, `device_mode/list.json`, `device_mode/update.json`, etc.

In addition, all version of **web services** share a same **data model** so that we have to consider the compatibility between different versions. Introduce the new change, without affecting the existing **web services** is the first principle.

b) Observer

As the number of users increases, we collect more and more user data. These data can bring **added value** by analyzing the user's usage data to study the working conditions of the electronic cigarette, and health effects of electronic cigarette to users. For the beginning, we just have developed the dashboard for the visualisation of user's usage data, and now We are trying to build our **Observer** platform in the form of SaaS, and to provide our data support to some of electronic cigarette research institutions.

For the access to our users' usage data, we offer the service to generate a csv file and download it. Taking into account the value of the data, observers need pay for accessing our service. To achieve this purpose, we need assure the security, reliability and availability of system. In the term of security, we should keep all sensitive information (card number, CVC code, etc) safe and invisible during the process of payment. We can't run a risk of leaking user confidential informations. So we need find a solution with a high security. In consideration of reliability, our payment process need treat every transaction correctly(amount, currency, etc) and have the ability of tracing every transaction and rolling back. For the availability of module, we should simplify the payment process and offer a friendly interface. We need integrate it into our existed observer system. So we want to keep the consistence in the dimension of design.

5.1.2 Objectif

a) Objectifs of Backend

Objectifs of backend are:

- Development continuous along with the evolution of **firmware** and **mobile application**
- Compatibility between different versions
- Maintenance of server and bug fixes

b) Observer

Except of the specific authorizations, the other payment process need to be reusable. Because a payment process is the most common module in the e-commerce application, there are already some mature platforms who offer their payment API to integrate in the third party application. We need create an independent payment module who is in charge of the payment process and the integration of the other plugins. It is responsible to invoke payment API to complete the payment, control the

authentication, respond to the webhook's event, generate and deliver the invoice to observer. We need abstract these components for reuse in other payment scenarios.

- Implementation of the basic **SaaS** functions (Subscription, Access Control, etc.)
- Integration of third party platform (Stripe, SendWithUs)
- Reimplement the registration process
- Management of subscription (upgrade, downgrade)
- Build a well-designed user interface by using **Bootstrap**

The goal of Observer is to construct an user-friendly and high-quality SaaS system and make it treat with payment correctly and safely.

5.2 Vocabulaire

In order to facilitate communication within the team and the preparation of documents, we have reached agreement with following defined terms:

- Backend:
- Observer:
- Web service:
- API:
- SaaS:
- Firmware:
- Smokio App:
- mobile application:
- Smokio:

5.3 Périmètre

Backend

For the moment, the users of web services are the **mobile developers** in our team, and the users of Smokio App are the **end-user** of Smokio who use Smokio to track their smoking.

As the only backend and full-stack developer in our team, I take charge of all the backend and website development.

So the constrain is to prevent other users from requesting our web services by authenticating their identities.

Observer

For conventional reason we name each of the user classes-actors with this format:

System Actors:

- **Observer:** The observer is the one who want to access our users' usage data for the purpose of research or commerce. They can join in our plans to get the authorization. The authorization contains the number of users that they can follow and the access to csv file.
- **User(Patient):** The user is the one who use our Smokio product and mobile app. These permit us to collect and analyse their usage data. Some of them want to stop smoking and need some professional guidance.
- **Administrator:** The administrator is our stuffs who are responsible to manage the pricing plan, monitor the payment events and supply the customer support.

The primary actor is the **observer** that connects to the site web and access the data. The term **user** is the data source, they produce the valuable data to observer.

5.4 Functional description

In this section, I will describe the functions of our system by using some examples.

For the part of Backend, all the functional description are listed in our [Web service documentation](#).

For the part of observer, by using the Flow chart, we can visually observe the workflow of the entire system.

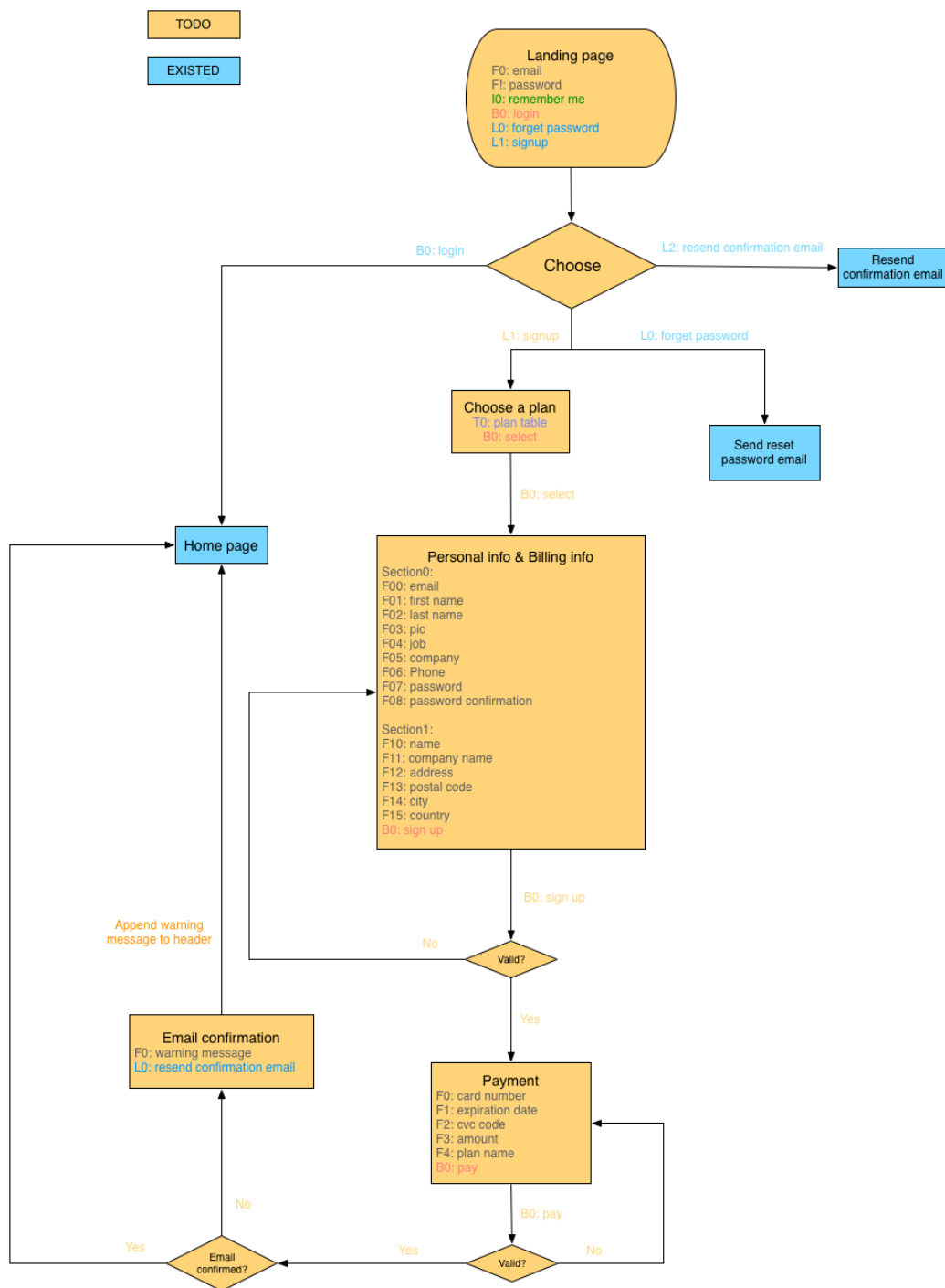


Figure 13 - Observer: registration process workflow

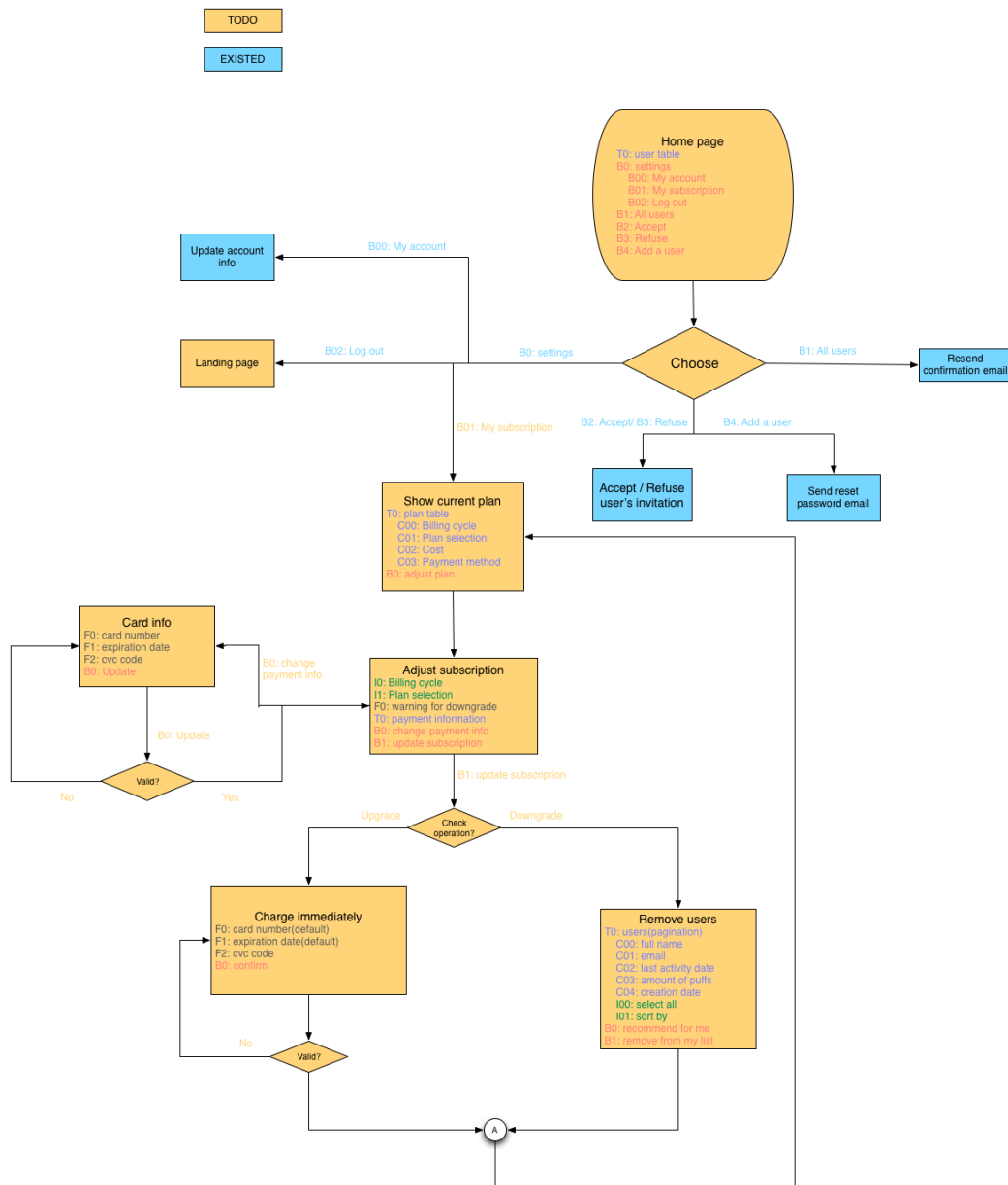


Figure 13 - Observer: subscription management process workflow

5.4.1 Plan Management::PM01

5.4.1.1 Introduction

We need an interface to manipulate the plans(create, parameter, delete). For the future, we could add the coupon or discount. A plan contains 4 principle parameters: price, frequency, csv authorization, user limit.

We can use the service of stripe for our plan management. It offers a nice interface to

manipulate the plan.

5.4.1.2 Inputs

The inputs are the content of plan. In the beginning, we provide 5 type of plan:

Name	Lite	Lite+	Standard	Plus	Pro
Price	0\$	20\$/month	50\$/month	399\$/month	1000\$/month
Frequence	Forever	monthly	monthly	monthly	monthly
User limit	1	10	50	100	300
CSV	No	No	No	Yes	Yes

5.4.1.3 Processing

Preconditions: Administrator has logged in with the stripe account.

1. Administrator open the stripe [dashboard](#)
2. Administrator click the menu "plan"
3. If we have already the plan, goto step 4, if administrator want to create a new plan, then goto step 6.
4. Administrator click on the plan name to show the detail of this plan

The screenshot shows a web application interface with a sidebar menu on the left and a main content area on the right. The sidebar menu includes sections for GENERAL (Dashboard, Customers, Recipients), TRANSACTIONS (Payments, Transfers, Balance), SUBSCRIPTIONS (Plans, Coupons), and REQUESTS (Events & Webhooks, Logs). The main content area is titled 'Lite+' with a price of '\$20.00/month'. It contains a 'Plan Details' section with fields for ID (lite+), Name (Lite+), Price (\$20.00 USD/month), and Trial period (No trial). Below this is a 'Subscribers' section showing 'No subscribers'. A 'Logs' section shows a log entry for '200 POST /v1/plans' at '2015/04/08 14:19:14'. An 'Events' section shows an event 'A new plan called Lite+ was created.' at '2015/04/08 14:19:14'. A red 'Delete Plan' button is located at the bottom right of the main content area.

5. Administrator click "Edit Details" to modify a plan, or click "Delete Plan" to delete a plan.
6. Administrator click "new"
7. Administrator fill the form of a plan

The screenshot shows a 'Create new plan' modal form overlaid on the application interface. The form has the following fields: ID (text input), Name (text input), Amount (text input with a currency symbol '€' and a value of '9.99'), Currency (dropdown menu showing 'EUR - Euro'), Interval (dropdown menu showing 'monthly'), Trial period days (text input), and Statement desc (text input). At the bottom of the modal are two buttons: 'Cancel' and 'Create plan'.

8. Administrator click "create plan"/"cancel" to confirm or cancel the operation

Postconditions: In dashboard the exist plan is disappeared or a new plan is appeared.

5.4.1.4 Outputs

A new created plan/ A modified plan

5.4.1.5 Error Handling

None, if the stripe server is down, we can't do anything.

5.4.2 Modification in the registration process::PM02

5.4.2.1 Introduction

During the process of registration, we need offer an interface to permit observer to select an plan, and complete the payment securely.

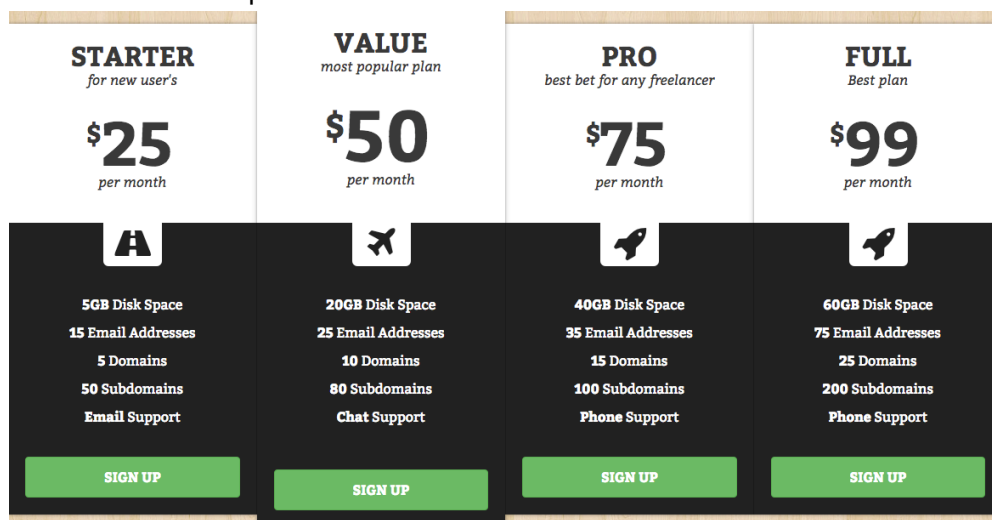
5.4.2.2 Inputs

Observer's selection and his credit card information.

5.4.2.3 Processing

Preconditions: Observer has filled his personal information.

1. Observer selects a plan.



2. A payment form is displayed.
3. Observer fills this form with his credit card.
4. Observer clicks "confirm".

5. Observer sees the payment information.
6. Observer clicks "Ok"
7. It will be redirected to the home page.

Postconditions: The home page is displayed.

5.4.2.4 Outputs

Our stripe account receives money, observer became a subscriber to a plan and has our authorization to access user data.

5.4.2.5 Error Handling

Subscription failed, please retry.

5.4.3 Subscription Management::PM03

5.4.3.1 Introduction

Observer should be able to manage their subscriptions, we need provide an interface to permit them to add more subscriptions, unsubscribe a plan.

Stripe provides the API to manage subscriptions, we need to integrate it.

5.4.3.2 Inputs

Observer's operations

5.4.3.3 Processing

Preconditions: Observer has logged in with his account.

1. Observer click "settings" in the top navigation bar.
2. Observer click the submenu "My subscriptions".
3. The page "My subscriptions" is displayed, there are some tables that contain detail information(detail about this plan, how many user limit left, can csv or not) for each current subscription.
4. Observer click the button "Unsubscribe" below a plan, goto step 5, if he want to subscribe more, goto step 9.
5. A pop up is displayed to ask the confirmation.
6. If observer confirms the unsubscription.
7. We stop the recurring his billing for next month.

8. He will no longer be able to access the service out of his subscription.
9. Observer click the button "Subscribe more"
10. Include PM01(Step 1-6).

Postconditions: Update observers' subscriptions and refresh the page "My subscriptions".

5.4.3.4 Outputs

Update observers' subscription relationships, and charge for the new subscription.

5.4.3.5 Error Handling

Subscription failed, please retry.

5.4.4 Authorization control

5.4.4.1 Introduction

After each payment of an observer, we should verify the detail of plan that he has rolled in, then authorize him the corresponding right to access the user data.

Likewise, when an observer unsubscribes a plan, we need to disable his access.

5.4.4.2 Inputs

Observer's successful payments or unsubscription operations.

5.4.4.3 Processing

Preconditions: None

1. Observer subscribes a new plan, goto step 2, if he unsubscribes a plan, goto step 3
2. Verify the payment, authorize the user limit and access to cvs.
3. Remove the authorization to this observer.

Postconditions: Update observers' authorizations

5.4.4.4 Outputs

Update observers' authorizations.

5.4.4.5 Error Handling

None.

5.4.5 Recurring the billing

5.4.4.1 Introduction

Every month, we need to charge for an observer's bill automatically. If an observer has stop his subscription, we will stop the charge for the corresponding plan too.

Stripe can recur the billing automatically, so we can use their service and integrate their API.

5.4.4.2 Inputs

Time chagement

5.4.4.3 Processing

Preconditions: Observer has subscribed a plan.

1. Observer's subscription is expired.
2. Verify this subscription isn't canceled.
3. Recur the billing for his plan.

Postconditions: None

5.4.4.4 Outputs

Our stripe account receive money for this subscription.

5.4.4.5 Error Handling

None, if the stripe server is down, we can't do anything.

5.4.6 Generation and delivery of invoices and receipts

5.4.4.1 Introduction

Once an observer is charged for a subscription successfully, we need generate and deliver his invoice and receipt automatically. We have some alternative solutions.

- Ruby has some gems for generating invoice:

- [payday](#)
- [Prawn](#)

- Some 3 party services:

- [sendWithUs](#)
- [invoiceninja](#) not free
- [invoiced](#)

Stripe is able to send the receipt for us, we could integrate it.

5.4.4.2 Inputs

Successful payment.

5.4.4.3 Processing

Preconditions: Observer is charged for his subscription.

1. We generate the invoice and receipt for observer subscription
2. We send the invoice and receipt to observer by email

Postconditions: None

5.4.4.4 Outputs

Invoices and receipts

5.4.4.5 Error Handling

Retry.

6 Planning

This chapter, I will introduce our development planning. We are a small and flexible team. Our schedule isn't always fix. We use the agile development methodology, like Scrum. We iterate very quickly.

6.1 Planned schedule

In general, according to the objective of my internship. I separate my job into 7 phases. As described in figure 12.

Phase 2 and 3 will go on parallelly, because during the development of a new version application, I will get familiar with our development process and methodology.

- Phase 1: Handover with former employee	0%		Start	Due	Assigned
Reading API development documentation	0%		Feb 23, 2015	Feb 27, 2015	Baicheng YU
Understanding the System Architecture	0%		Feb 23, 2015	Feb 25, 2015	Baicheng YU
Installation and configuration of development environment	0%		Feb 23, 2015	Feb 25, 2015	Baicheng YU
Learning the way of internal communication	0%		Feb 25, 2015	Feb 25, 2015	Baicheng YU
Learning configuration management	0%		Feb 25, 2015	Feb 26, 2015	Baicheng YU
* Task Milestone Group of Tasks					
- Phase 2: Learning Development Process	0%		Start	Due	Assigned
Learn to use Smokio	0%		Mar 2, 2015	Mar 3, 2015	Baicheng YU
Understanding every function of Smokio App	0%		Mar 2, 2015	Mar 6, 2015	Baicheng YU
Learn to use AWS	0%		Mar 4, 2015	Mar 10, 2015	Baicheng YU
Master database model	0%		Mar 11, 2015	Mar 20, 2015	Baicheng YU
Learn how to create a new web service	0%		Mar 11, 2015	Mar 25, 2015	Baicheng YU
Practice deploying code to the production environment	0%		Mar 26, 2015	Mar 27, 2015	Baicheng YU
* Task Milestone Group of Tasks					
- Phase 3: Development of new version App	0%		Start	Due	Assigned
- Phase 4: Development of Observer	0%		Start	Due	Assigned
- Phase 5: Integration of new firmware, as well as developing a new version of APP	0%		Start	Due	Assigned
- Phase 6: Open API	0%		Start	Due	Assigned
- Phase 7: Development of Intelligent Coach model	0%		Start	Due	Assigned

Figure 12 - Gantt Diagram: planned schedule

6.2 Actual schedule

phase 1: Handover with former employee (23/02/2015 - 28/02/2015)

- Reading API development documentation
- Understanding the System Architecture
- Installation and configuration of development environment
- Learning the way of internal communication
- Learning configuration management

phase 2: Learning Development Process (02/03/2015 - 27/03/2015)

- Learn to use Smokio
- Understanding every function of Smokio App
- Learn to use AWS
- Master database model
- Learn how to create a new web service
- Practice deploying code to the production environment

phase 3: Development of new version App (05/03/2015 - 01/04/2015)

- Development of v8 version API
 - support of deep-linking
 - send push notification to mobile
 - notification for the recovery progress of health
 - deployment v8 into the production environment
- Fix bugs
- Observer: improvements of existing features
 - flash messages in the registration
 - Email notifications of approval)

Phase 4: Development of Observer(02/04/2015 - 20/05/2015)

- Development of SaaS fonctions
 - Requirement analysis of SaaS
 - Leans to use Stripe
 - Stripe integration
 - Implementations of SaaS
- In-app purchase
 - Create a new ec2 instance
 - Installation and Configuration of development environment
 - Implement, deploy an e-commerce website(Spree)
 - Multi-currency support
 - Account-recurring support

Phase 5: Integration of new firmware, as well as developing a new version of APP
(21/05/2015 - 12/06/2015)

- Development of v9 version API

- add mode management for device
- add temperature, power attributes for device
- Improvement of Observer
 - Referring Zendesk, improve the subscription process
 - Improved UI, and add dynamic Ajax effects

Phase 6: Open API (15/06/2015 - 03/07/2015)

Phase 7: Development of Intelligent Coach model (03/07/2015 - 21/08/2015)

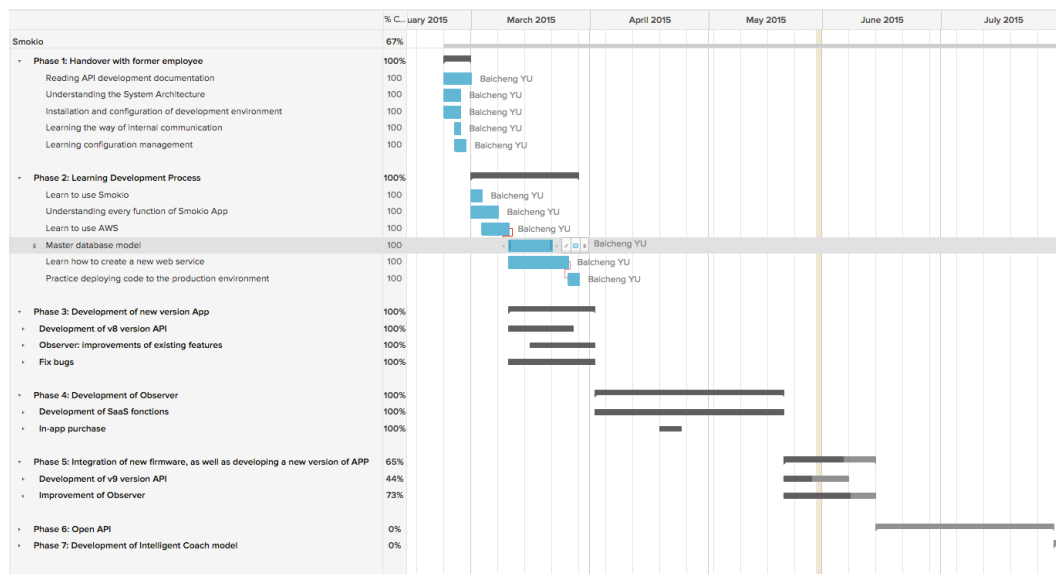


Figure 13 - Gantt Diagram: actual schedule

7 System Design

7.1 Backend

7.1.1 System architecture

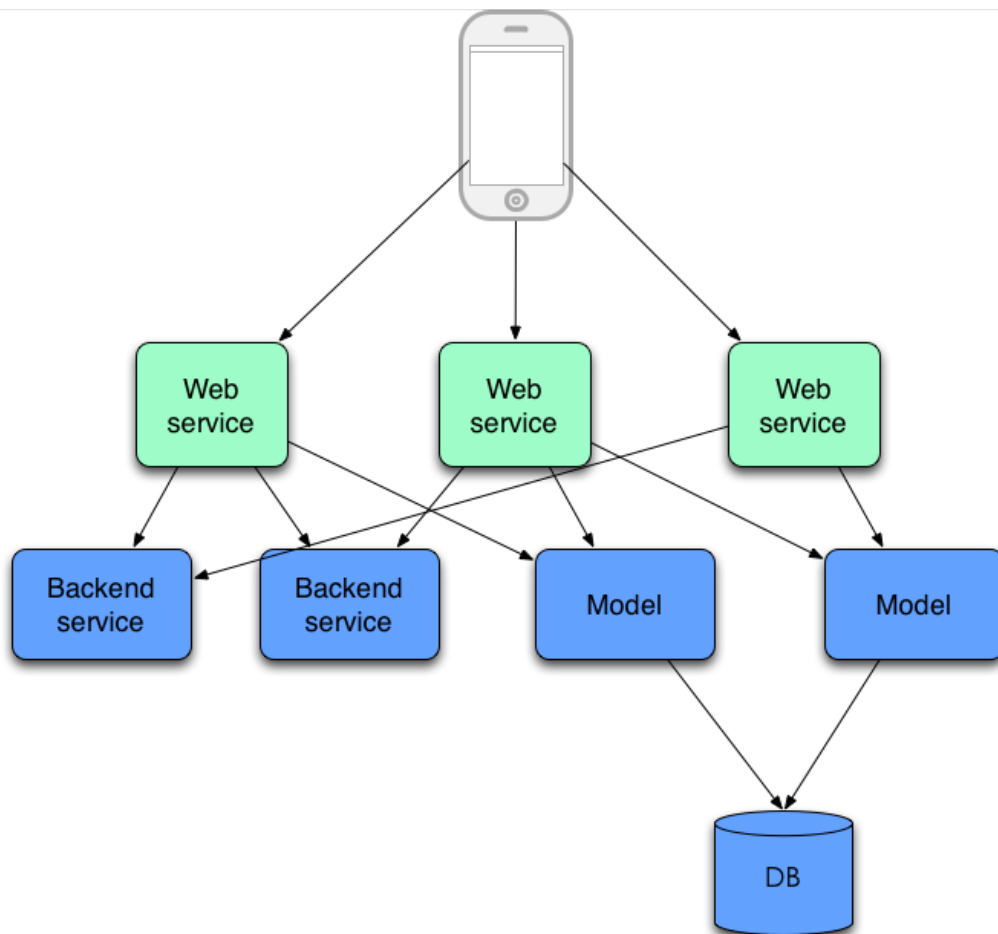


Figure 12 - Backend architecture

7.1.2 Database model

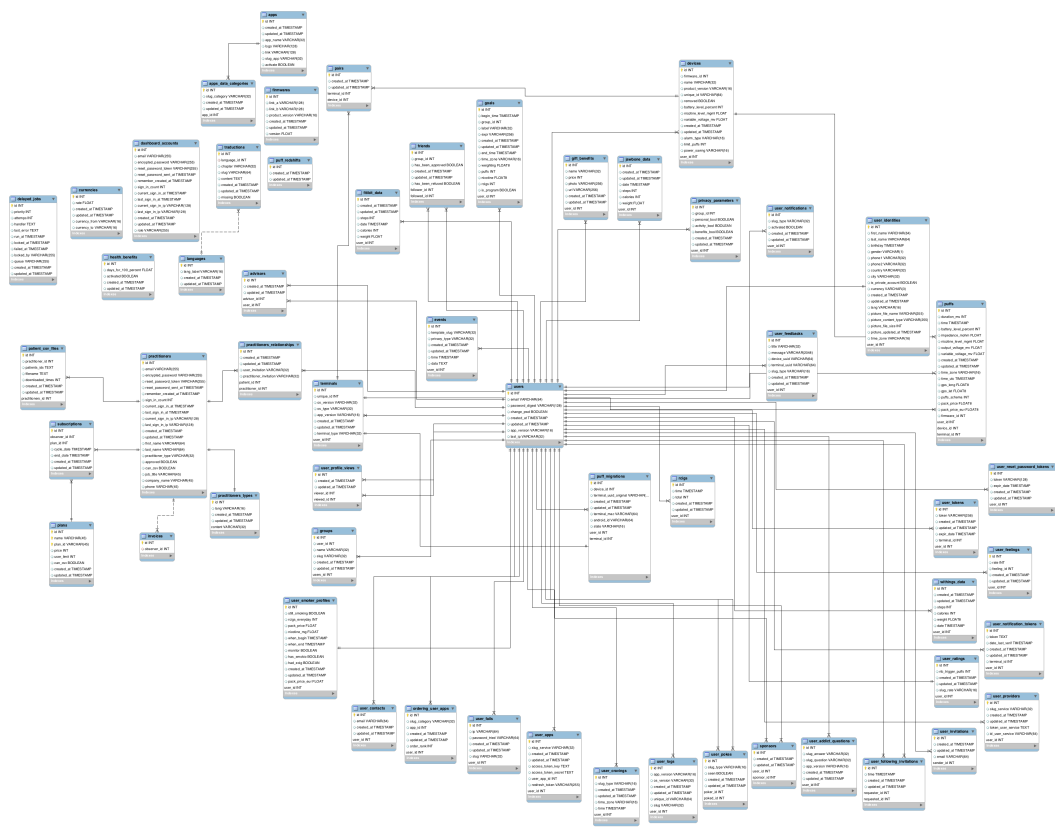


Figure 12 - Backend-Database model

7.2 Observer

7.2.1 System architecture

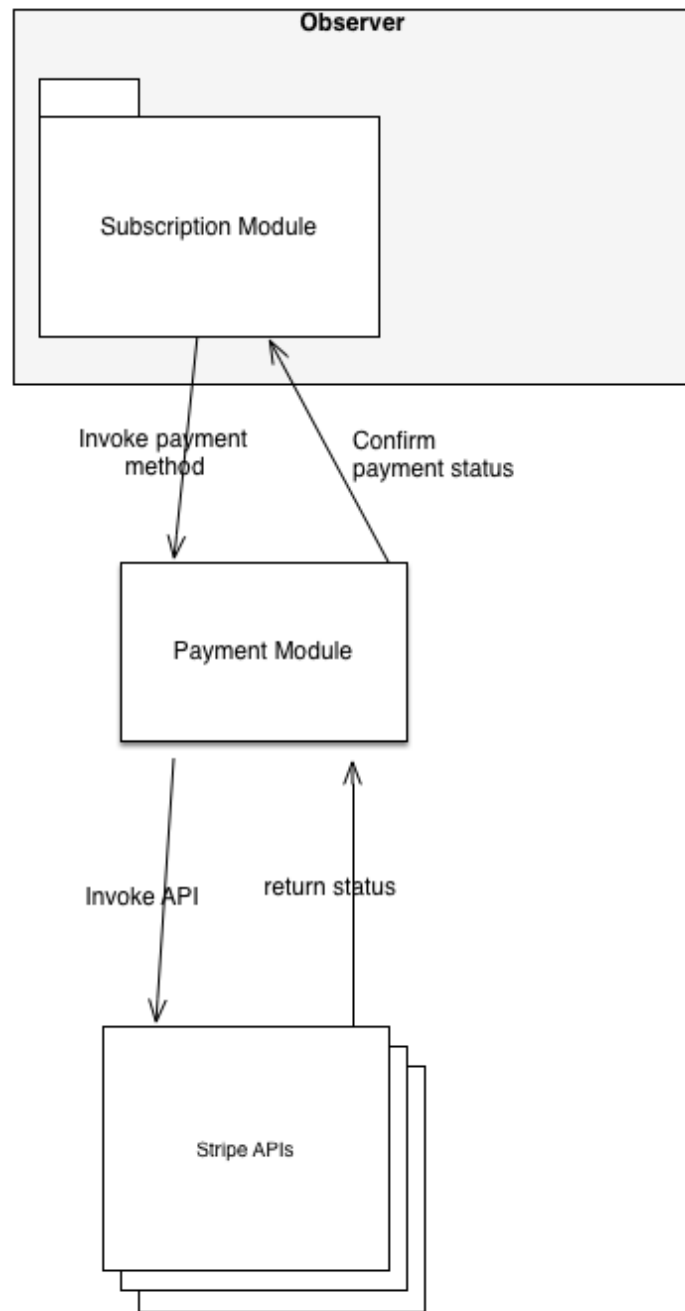


Figure 12 - Observer architecture

7.2.2 Database model

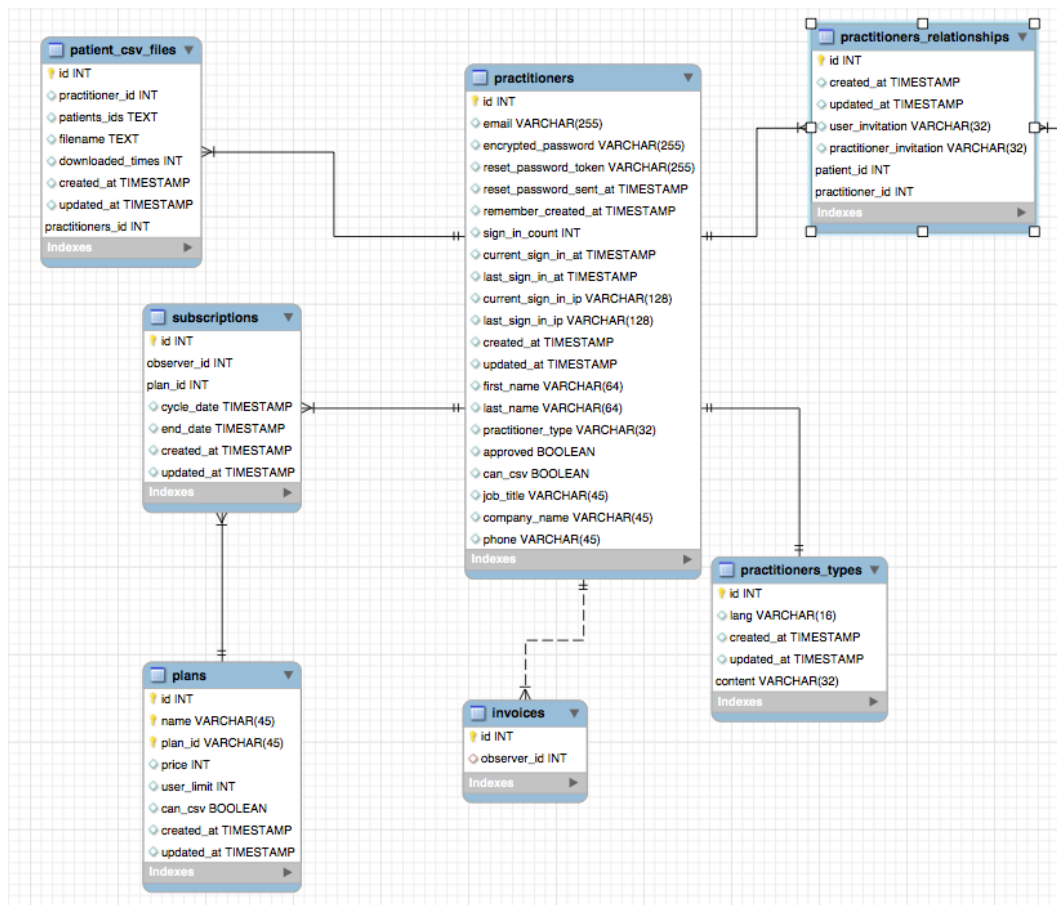


Figure 12 - Observer-Database model