Loyalty Program For Urban Bicycle Promotion

António Maria de Sousa Nunes Marques Pinto
antonio.m.pinto@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2017

Abstract

Cycling as a means of locomotion alternative to the car is starting to become appealing in urban environments. The environmental and health advantages are a growing focus in society and the economical advantages, both locally and globally, while less obvious are perfect to promote bicycle use. Taking this into account a system of challenges and rewards, where shop owners can register, created around the use of bicycle as a means of transport, would fulfill the objective of encourage the use of bicycle and increase local economy. The focus of this thesis is the system that allows the shop owner to easily implement customized challenges and rewards for their stores and to allow cyclists to visualize, participate and complete challenges.

Keywords: Cycling, Loyalty Programs, User-generated content, Reward Schemes, Usability

1. Introduction

In recent years, bicycle use as a means of transport has become more popular, with around 200 million bicycles in the EU alone in 2000. While in some countries the use of bicycles as a main mean of transport is already happening, like Denmark and Netherlands, in others, steps are starting to being taken into account. Cities are trying to provide better cycling infrastructures; other steps being put into action are bicycle-centric tourism, e-bikes, and bike sharing.

There are two main motivations for this project.

First is the environmental and social aspect, by promoting people to use a bicycle as an alternate means of transportation it will reduce the dependency of fossil fuels and emission of greenhouse gasses like CO_2 and increase people's day to day fitness activities[1, 2]. Even when including the CO_2 emissions of food required by the increased cyclist dietary intake, the bicycle is still the lowest emitter of greenhouse gasses per passenger kilometer traveled, around ten times lower than a car and 5 times lower than a bus. The increase of fitness activities on the day to day life also increases happiness and reduces the probability of developing depression, this combined with the decrease of traffic and costs increases the productivity of people.

The second motivation is economical since unlike what most people think using the bicycle as a means of transportation actually has sizable economic benefits both in large scale and in small local economies. In large scale, using bicycles increases jobs, presently there are more than 650 000 jobs in the EU related to cycling, by doubling bicycle usage we could add more than 400 000 extra jobs.

More importantly for this project specifically are the local economy benefits, since this is where the investment is done and the promotion is made. Retailers do not have a notion and often underestimate a number of clients that go shopping by bicycle and overestimate a number of clients that use a car. We can see an example of this in Bristol, England, where a survey [3] showed that retailers not only underestimated a number of customers that travel by bicycle to their shops and the distance they would travel but also overestimated by around 100% the share of customers that drive to their stores. At the moment, customers going shopping by bike accounts for 111 billion euros in EU [3]; if this number doubles it would generate an increase of more than 27 billion euros in local retailers alone.

Currently, these systems are all focused in very specific situations and communities. For example, Ride2School[4] is designed specifically to be used in schools since the teacher manages the counting of a few students. This method would not be easily applied to other communities with more users. So there is a gap to be filled by an application that is flexible enough that can have its reward schemes be tailor made to the audience while still being simple to use. This thesis is one of a three part system. Cycle-to-shop tries exactly to encourage people to bike while being relevant in any community. The three parts are, tracking the biker, storing the tracking data, and the one this thesis approaches, the definition of reward schemes by the user.

2. Background

To create an easy-to-use system that allows non-programmers to create content we must delve into end-user programming [5]. From end-user programming, we can take away the difficulties the shop owners will face when creating their own reward schemes. The following sections further explore the current solutions in easy-to-use content creation.

2.1. End-user Programming

End-user programming is a field that allows end-users to program an application. In this case, we want to allow shop owners to create their own reward scheme logic. So, to be able to design a system that allows shop owners to create their own reward schemes, we must understand what are the end-users main difficulties and how to lessen them. What distinguishes a professional user from an end-user is mostly knowledge but also motivation, background, and knowledge. This means that end-users can be professionals in a technical field but not have in-depth programming knowledge, a researcher in an unrelated field or just a normal user. So end-user programming ends up being a practice where users usually have less technical knowledge [6].

Surprisingly, even with less technical knowledge, end-users have the same difficulties as professional users, which are designing, testing and debugging their programs. These are especially important since there have been stories of severe consequences due to end-user errors. A famous example was a Texas oil firm lost millions of dollars due to an error in a spreadsheet formula [5].

There are currently four approaches to making a flexible easy to use, end-user programmable system: programming-by-example, natural language programming, and templates. In the following pages these approaches will be explained, given examples of and dissected into pros and cons.

2.2. Programming-by-example

Programming-by-example (PBE) or by demonstration, is an approach that allows users to define a behavior without programming. This is achieved by the user inputting an example and the specific output for the example, instead of programming the behavior in general. For the system to capture this example, the user must first explicitly start it. The user then executes a sequence of actions that represent the desired behavior. The sequence of actions is used to turn an example input into an output. The sequence of actions, input and output are an example. From the input examples, it is then extrapolated the desired behavior and the result is a more general program that encompasses the original example intent. The generated program, given the example inputs, will output the correct results. This generated program is also then used, for other inputs other than the example inputs. PBE is an attractive approach for end-users since they naturally express themselves with examples.

2.3. Natural language programming

Natural programming languages allow users to program using a language that the user uses every day, e.g. English. This approach allows end users to create rewards schemes with small to no additional knowledge. On the other hand, it has disambiguation problems, since words and phrases can have different interpretations depending on usage, context, culture and other factors. Even though the biggest challenge in programming is considered to be the interpretation of the problem and the designing of the solution, learning a new language is also a challenge for new programmers, especially because programming languages are very different from natural languages. While professional programmers might be used to programming languages, end-users also need to learn how to work with a strict syntax language, this is what natural language tries to fix.

Before going into more detail about natural languages, we must define its types. Natural programming language, might not be the ordinary English we are accustomed to; To avoid confusion *Foundations of the case for Natural-Language Programming* makes a distinction between two types of natural language, the active and the passive one. Passive natural language is a language that might look like natural language just enough so that anyone can instantly recognize it. On the other hand, unlike active a natural language, it has a rigid syntax so only people that are already experienced in it will be able to write applications using it. The active natural language, on the other hand, allows a user to write using a subset of a natural language for creating applications. So instead of the user only being able to use some pre-defined fixed amount of statements, in a specific way but a bundle of words that can use with the flexibility of natural languages and altered or extended easily. Passive natural language has very few pros; its readability is the main one, but it still maintains the rigidity of programming languages so we dismiss this type of natural languages. Another variation on the natural programming language is in using a keyword style; this takes advantage of the fact that most users are accustomed to using search engines, like Google, by inputting keywords. Just like when inputting keywords into a search engine, there is no strict syntax. From keywords like "add text button" the system would create a text button. When tested with users in *Translating Keyword Commands* [7] the users were able to use the correct keywords for 84% of the tasks, and their first attempts had 72% success rate.

2.4. Visual Programming

A visual programming language is a language where each primitive element is represented with a visual element instead of a textual one. It is also a very high level of abstraction language since each visual element heavily abstracts from its logic. These visual elements are then connected to create a program. There are three types of visual languages: the flowcharts, the non-flowcharts, and the most popular, the spreadsheets. Flowcharts are diagrams that are composed of boxes, of several types, that usually represent functions and arrows that display the logic flow of the program. Non-Flowcharts are diagrams that are simply composed of boxes, of various types, that connect with each other. Without arrows, the connections between boxes could be implemented to force the correct syntax by having boxes only connect if the following box has a matching symbol to the previous box. Spreadsheets are tabular forms that allow the user to specify the column and row meaning and have the cell content display its value. This is a field that does not have many empirical evidences, and most of the theory of its advantages come from the popular belief that visual elements are simply more user-friendly than text or its popularity. Between the few pieces of research that do exist, some conflict among them, indicating that the advantages of visual languages are not very clear cut. Fortunately, it is a field that is being researched more and more.

Visual programming is currently used massively in two fields. In teaching, mostly to teach programming to children and novices. In companies for the budget, cash flow and other types of calculations using spreadsheets. Some examples of teaching with visual languages can be seen in Scratch and Tynker, two projects that are primarily used by youth, ages 8-16 year to program apps and games, using a non-flowchart like a scripting language. Scratch has even been also implemented as a beginner computer science course at Norfolk State University, for at-risk students [8]. To evaluate the results it was compared the grades and student retention between control and target groups. The control groups are simply students that had directly CS1 programming classes. On the other hand, the target group was at-risk students that had taken one semester of CS0 with Scratch. When comparing these, the results show that between two years, the target group shown better overall grades. On retention, around 60% of students that were exposed to Scratch kept going to the course while only 30% of the control group kept going. There were also surveys answered by the target group, where the students wrote that not only they enjoyed CS0 classes but it also provided them with better mindsets and a solid foundation for programming. We can then conclude that visual languages, specifically Scratch, have been applied with success in teaching classes.

2.5. Form Templates

Form Templates, like the name, indicates is a document with a strict layout with several boxes where the end-user can input several types of values. While the layout can change, it is never the duty of the end-user to change it, this is due to the fact that the type of each input box actually hides a logic that is hidden from the end-user, only known by the developer. This results in Form Templates being an approach where the developers can design all the implementation and keep it as a black box to the end-user, while still allowing customization of certain values, also picked by the developer. While it does reduce the flexibility of the end-user in terms of creating new logic, it significantly lowers the entry barrier by removing the logic creation and only opening to the user some values.

Even though the above solutions, are easy to learn when the user is willing, they are not instantly learned. When target user tests were made, the users expressed that they would rather have a simple approach that does not need a big time and work engagement. This requirement tells us that while the previous approaches are more flexible, they are too involved, requiring not only learning how it works.

The Form Templates approach means that there is no flexibility, but it allows the user to not care about the logic since it is already done. By researching the most used shop challenges we can try and hit the needs for most shopkeepers without forcing them to learn visual programming or other complex approaches. These templates still have some customization in that they are form templates. The logic implementation is hidden from the user but still allows the shopkeeper to input values into several boxes that feed into the logic of the template.

To reduce the necessity of the end-user having control over the campaign logic there needs to be a good array of form templates already. For that, it is needed to implement the currently most popular loyalty programs.

2.6. Custom Solution

A custom solution is where each shop owner would contact and hire a developer to create a solution specifically crafted to its own requirements and clients needs. So in reality, this solution is unlike the others since the end-user, in this case, the shop owner has no hand in the logic behind the challenges.

This is a trade-off between time and money. By hiring a professional the shop owner spends less time implementing the reward system or learning how to implement it and instead simply pays the professional to create it.

2.7. Solution Comparison and Conclusion

To select a solution they need to be compared to each other while taking into account the requirements.

First, we compare a custom solution fully constructed by a professional versus end-user programming. By having the challenges created by the end user, most of the implementation cost is gone compared to having a custom work done by a professional that would end up costing thousands. So the end-user programming solution is preferred.

On the end-user programming there are several solutions and to choose one they need to be compared taking into account the requirements. While the flexibility of the campaign creation and scalability are important requirements, the most important requirement to fulfill is the ease of usage for new users. This is because ease of usage will be the one factor that will drive user acquisition, more shop owners will adhere to cycle-to-shop, and user retention, more shop owners will keep using cycle-to-shop. From the Table 1 we can see that the best solution is Form Templates since it has the best ease of usage. To minimize the flexibility problems two approaches are to be taken: create a wide array of different templates and have an advanced mode that allows shop owners to combine parts of each Form Template to create a customized campaign. For scalability, Form Templates are the best approach since it is only required to save an identification number of the template and its input values, while on the other approaches since they are more similar to programming, it would be needed to save the code done by each shop owner. Finally in terms of security, since the logic behind each reward scheme is all implemented on the server side, while on other implementations, the shop owner would be creating the logic and sending it to the back-end, leaving it vulnerable to all kinds of attacks, like for example code injection.

Templates allow the shop owners to instantly set their campaigns without having to learn any programming language or API. Even though they are not as flexible as the previous methods, the requirement of fine-tuned personal customization is not one as important as ease of usage since shop owners, as end-users, do not have the time investment or patience to spend on a fine-tuned system.

Since usability is the most imperative requirement, Visual, Natural Language and By Example Programming are not the preferred solutions, since they trade ease of usage for higher flexibility. Form Template and a Custom solution by hiring a developer, on the other hand, have high ease of usage. On the other hand, by hiring a developer it will have a high implementation cost while Form Templates do not. A Custom solution possibly even requires its own server hosting which adds another cost. Another problem with the custom solution is that for each challenge it will require an extra cost to hire the developer.

This means the best approach to take for the current problem and user base is the template-based.

	VisualP	NLP	PBE	Template	Custom
Usability				x	x
Flexibility	x	x	x		*
Low Cost	x	x	x	x	
Scalability				x	*
Security				x	*

Table 1: Comparison of possible approaches *Since with a custom solution is implemented by a company, the requirement fulfilment may vary.

3. Implementation

The overall system can be divided into three components: the cyclist mobile tracking application, the tracker database and the Reward server. Both the tracker database and the Reward server are hosted and managed by a third party. The Reward server fully implements the shop owner reward system web application and interacts with the other two components as part of the functional loyalty challenge-based programs implementation.

A few examples of how the components interact are:

- Shop owner can use a web interface to create his own reward schemes.
- Cyclist application can query the reward server and retrieve nearby shops, challenges and rewards.
- Whenever a cyclist, using his mobile application, completes a bicycle ride, it sends the route data to the tracking server, which saves it in a database. The mobile app also sends a request to the Reward server in order to update challenges progress with the new ride data.
- Cyclist application requests a reward code for a specific challenge from the reward server. The Reward server then queries the trace database and compares the challenge requirements with the cyclist data.
- Shop owner queries the Reward server with a cyclists reward code, in order to know if the code is valid or not.

An extra component is the shop owner's offline mobile application. This thesis will not tackle this component in depth since the offline application's objective is to provide some of the functionalities of the Reward server without requiring an internet connection.

The Reward server has two main functionalities: allowing the shop owner to implement his own challenge-reward loyalty programs and interacting with the cyclist application to handle the cyclist's challenge progress and completion.

3.1. Cycle-To-Shop Components

The first component, the cyclist mobile track application, has as its main purpose to track the user cycling routes. It also needs to communicate with the other two components. It sends messages to the tracker database with the routes, where the cyclist has been, the paths he took, etc. This does not mean the application needs to be connected to the internet constantly. Instead, the application simply requires GPS enabled when cycling, and once the cyclist has finished his trip, he can connect to the internet and upload the trip data to the tracking server. In case the cyclist does not have an internet connection the mobile application caches the request for later, following an opportunistic approach. On the other hand, the cyclist app also needs to send messages to the reward web server to query if the cyclist has fulfilled a challenge set up by the shop owner. This is used when the cyclist requests a reward or when he wants to see what was his challenge progress. Finally, it also needs to receive messages sent by the reward server to know what shops and challenges exist and display them to the cyclist. The list of shops and challenges received by the application take into account the user position and a user specified radius.

The second component is the tracker database; this is where the tracking data from the cyclist is stored. It uses a graph-based database to easily save positional data and a small relational database to save simple data like kilometers traveled. This database also needs to be queried by the reward server in order to obtain all the cyclists' data.

Finally, the reward server has two main purposes: to communicate with the cyclist application and to allow the shop owner to set up his own reward system for his shop. When communicating with the cyclist application, the reward server needs to send data about nearby shops and challenges, display the progress in fulfilling the requirements of shop challenges and whenever a cyclist has fulfilled such requirements, send a code to the application as a verification method to the shop owner that the cyclist can receive a reward. On the shop owner side, it must display several templates for reward system types, save the reward system customized by the shop owner and provide a way for the shop owner to check if a cyclist code is valid.

3.2. Reward Server

The Reward server was implemented as a web server. This allows the shop owner to create his reward scheme in an environment familiar to him, the web browser. It also allows the cyclist app to be easily easily ported from one mobile platform to another. One tool currently being used a lot to develop web servers is Node.js.

Node.js is an asynchronous event-driven JavaScript runtime that is specially designed to build scalable network applications. So unlike the more common servers that go with the concurrency model where multiple threads are used, these design choices are perfect for scalability and optimized throughput. Node is also easy to work in part due to its module-centric nature, allowing developers to easily use community published modules, and also because it uses JavaScript, meaning that the developer can use the same language both for client and server side.

3.3. Challenge Templates Design

A loyalty program will attract new guests and turn the already customers into more loyal ones, and in this case, it will also try to influence their behavior by rewarding cycling. There are several types of loyalty programs, classified by the rewards and requirements.

The main types of loyalty programs are [9]:

- Type 1: Members receive a price discount off a static amount at the register. Popular implementations are usually a discount of a percentage of the total price or a discount of a specific monetary value.
- Type 2: Members receive one unit free when they purchase n units.
- Type 3: Tiered rewards. Once a customer receives a reward, his following rewards will be of increasing value.
- Type 4: Customer relationship with frequent special offers.

A challenge type system was chosen as the solution. Shop owners create a challenge with certain requirements related both to cycling and their own business. This offers a bigger flexibility than the previous static discount and also uses gamification to make it more interesting for cyclists.

The reward server provides several templates to allow the shop owner to customize his own reward challenge. The template method itself can be done in two ways. Either a single template, that has a multitude of options to customize into one of the several types of challenges, or several simple templates with a single challenge type in mind. Limiting each template to a unique simple challenge type results in a more intuitive system.

To choose the templates, current popular challenges from existing loyalty programs were taken into account. The simplest one is simply to promote cycling. Once a cyclist does a certain number of kilometers, chosen by the shop owner, he can get a reward. The intent of this challenge is to promote events since it simply requires the cyclist to travel and nothing more.

The next template is like the previous one, but the shop owner can choose a city where the cyclist must ride. The objective is the same as that of the previous template, to promote a more active lifestyle but in a specific city. This can be used to promote an event of a specific municipal center for example.

The next template is "Cycling between two dates", which makes it very much like the first one but with a date restriction.

"Cycle and win a raffle", is a template that allows shop owners to create a raffle, giving a more expensive reward to a single user randomly.

"Visit the shop's owner shop while cycling". The visits are tagged by the application when it detects a special beacon on the shop. This template can be used as a visit frequency loyalty program, where the cyclist that visits the shop the most receives better rewards. It can also be used to emulate the type one loyalty programs, by having the discount coupon as a reward. In theory, the shop owner can also only allow the cyclist to detect the beacon after the performing a transaction or using a service, causing this template to work as a type two loyalty program since the cyclist is only rewarded after buying something n times. Which is very similar to how stamp cards work.

"Happy hour", which is a popular discount program to attract clients in bars. The cyclist must visit the shop, very much like the previous template, but in a restricted time frame. It can be used by restaurants to promote client visits during hours where there are not many clients.

Finally, the last template is "Cycle and the number of previously completed challenges". This allows shop owners to create the type three of tiered challenges by creating some simple challenges, then have some harder challenges that require a certain number of previously completed challenges.

3.3.1 Challenge Template Architecture

Challenges are organized on a two-file-per-challenge basis. Each challenge is implemented in two parts: the visual part, and the logical part. The visual part is what appears for the shop owner, containing the interface where the shop owner inputs his values and sends it to the server. The logical part contains the functionality that is called when the reward server needs to verify if a cyclist has finished a challenge. The logic needed to verify if a cyclist has finished a challenge encapsulates: which queries to call on the TraceDB and the RewardDatabase, which and how to compare the values and outputting if a cyclist has finished the challenge. The logic can also output a progression value for how much the cyclist has completed the challenge instead of a simple boolean state of completed or not.

3.4. Rewards

Once the cyclist requests a reward code and the reward server verifies that the cyclist has completed the challenge, it creates a reward code. The reward code is generated by hashing three values: the cyclist identification number, the challenge identification number and the day of the request. This generates a simple four letter code. It is important to be simple since the cyclist will have to easily display or tell the code to the shop owner. The shop owner will also be busy with his usual activities and other customers, so this verification should be simple and short. A simple string is the most straightforward method of authentication, user-friendly for the non-computer-savvy shop owners and it does not require extra equipment like barcodes and QR codes.

The three values used are needed to create a unique code for each person-challenge combination on a specific day. This is important because if the shop owner does not have an internet connection, he can use a separate shop owner specific mobile application. This mobile application also receives the challenge id number, the cyclist id number and the current day and generates a four letter code. If both the shop owner app generated code and the cyclist server generated code match, then it means the cyclist should receive the prize. Unfortunately, due to this requirement, it means that both the offline shop owner application and the reward server need to have the same reward code generation and it must be deterministic thus rendering salt-like techniques to improve the hash security impossible.

Once the cyclist gives the code, and the shop owner verifies its validity through the server or his mobile app the cyclist receives the reward. The reward is saved and displayed as a simple word, as to avoid extra costs for the shop owner that would come by forcing a specific hardware piece that reads and registers specific products barcodes as rewards. The barcode method is also less flexible since it does not allow services, discounts or products that are not sold in that shop. This means that a reward can be pretty much anything from a flat discount on a restaurant, a free meal, free night in a hotel, a free trip, and others.

3.5. Tracking Server and Cyclist Application

Once the shop owner creates his challenges program, the cyclist needs to be able to view them, follow his progress, complete the challenges and receive the reward. These actions complete the interaction cycle between the shop owner and the cyclist and are handled by the reward server via interactions with cyclist application and tracking server.

3.6. Communication

The reward server needs to receive data from several sources: the shop owner, the cyclist mobile application, and the geospatial Trace database.

The Trace database was already created and therefore already had a messaging protocol defined. To query the Trace database and receive a cyclist route data a message has to use a RESTful API and go through the HTTP protocol. The challenge program creation by the shop owner has to be simple and easy to use. Because of this, the shop owner challenge creation application was decided to be made as a web application, accessible in a browser, so using HTTP was a given, and therefore HTTP verbs as well.

The communication with the cyclist mobile application for the sake of consistency also works through HTTP verbs. Another advantage of using HTTP as a communication protocol is that it is implementation independent and therefore platform independent. By using HTTP, the mobile application can be made to work natively, as a web application or a hybrid. The hybrid solution is an application implemented with web technologies like HTML, CSS, and Javascript but hosted inside a native application that works as a full-screen web browser.

Since there is some sensitive information, like the cyclist position or the reward code, messages are sent through HTTPS, Hyper Text Transfer Protocol Secure. HTTPS is actually HTTP over a secure protocol that ciphers the communication, which is usually either TLS or SSL. HTTPS provides security by ensuring authentication, encryption and data integrity. These are implemented by the secure protocol extra handshake at the start of the connection, where the server sends a certificate, this certificate has a key, allowing the server and the client to share a secret and establish a secure connection. This certificate is generated from a certificate authority which ensures authentication.

This is important, since shop owners with potentially big rewards, like a car or a free vacation, will be a target of attacks like man-in-the-middle attacks or simple packet sniffing. By having the sent reward code encrypted, an attacker can not capture it and pretend it is his. Data integrity ensures an attacker with a fake reward code can not give it to the shop owner, waiting for him to request a verification on the browser and have someone tamper the server message turning a "this code is invalid" into "this code is valid" kind of message.

The cyclist's GPS location is sent when the cyclist queries the reward server about nearby shops and challenges. This GPS data is also sensitive data since an attacker can over time create a route based on usual GPS locations and find a cyclist's real location. Since the message is two-way encrypted, the cyclist is also protected from packet sniffing.

From the side of the cyclist mobile application, most messages are cached once it loses internet connection. This is especially important for when it commits a new bicycle trip to the Trace database, requests a challenge progression update and when he requests a reward code.

3.7. Database

Each shop's relevant data and challenges need to be stored so that the cyclist can later receive them.

Each shop owner has associated a shop profile, which the shop owner inputs on the website. The shop profile consists of the shop name, shop GPS latitude, and longitude, identification number of the low range beacon used by the cyclist mobile app and an image logo reference.

Each shop owner can create his own challenges. Challenges are saved in separate tables. On the challenges table, the metadata is saved of each challenge. These are comprised by the challenge name and description, date of set up, type between simple template or a combination of templates, unique id number, and reward for completion. For each of these entries, there are one or more entries on the challengeTemplates table.

This table holds, for each challenge, the template id number, so that later, when the cyclist requests a reward code, depending on the template type, it can use the correct function to verify if the cyclist has completed the challenge or not. It also holds the requirement values chosen by the shop owner, which are required to assess if a cyclist has completed the challenge or not.

Once a challenge is completed and a cyclist requests a reward code, the code is generated and it is saved in the challengeCompletion table that works as a whitelist. In the table, the challenge id number, the cyclist id number, and the reward code are all saved. This table also avoids the same person using the same reward code several times or sharing it with friends as to have multiple uses. Each code entry is only saved for a day.

Once the reward code is shown to the shop owner and verified to be correct its entry on the table is deleted. In its place, it is either added to the rewardUsed if the challenge had not been completed by the cyclist before or incremented. The date of the last time it was completed is so that it resets the progress of the challenge, and the next time a cyclist requests a reward code, the server will query the Trace database how many kilometers has the cyclist done since the last time he completed the challenge. This avoids the same cyclist requesting several times a reward code for the same challenge. This table is also used for the challenge template, that requires the cyclist to complete a certain number of challenges since it counts how many times each challenge has been completed.

For the raffle challenge, there are two extra tables. One saves each cyclist entry on live raffle challenge, and once they end, an entry is selected randomly and saved on the finishedRaffles table while all other entries in the previous table for that challenge are deleted.

4. Results

In this section, the experiments and results of the evaluation of this thesis are presented. The objective of these tests is to determine if the Cycle-to-Shop Reward server achieves the requirements of ease of use and flexibility. Ultimately, we present the experimental results obtained and corresponding conclusions.

5. Requirement 1 : Ease of Use

Ease of use is one of the most vital requirements since it is the deciding factor for user retention. This section examines the experiment and the results, to try and verify how easy to use the template approach and the interface is.

5.1. Experiment

This test tries to determine how easy it is for users to understand and use the interface and template forms. Each test subject is first given a small explanation of what are the roles of the shop owner, the reward schemes and the cyclist. The test subject also has as much time as he wants to first read the main page. Each test subject is then presented with seven tasks that encompass the shop owner's use cases. For each task, the number of mouse clicks are counted and also the time it took to complete the task. Since test subjects were encouraged to think aloud during the task, any opinion or mistake committed were also noted. The objective of this experiment is to attempt to quantify the usability of the interface, pinpoint any interface mistake and verify how easy it is for users to understand how to create a challenge from templates.

The tasks are:

- Task 1: Sign on with Google and create a store called "MyRestaurant", in Av. Rovisco Pais 1 Lisboa, with "This is my restaurant" as it's description and 21334512 as the beaconID number.
- Task 2: Setup a simple challenge where if someone cycles to your store 3 times he gets a 10% discount. The challenge is called "Cycle To Me" with a description of "Everyone should cycle".
- Task 3: Setup a simple challenge where if someone cycles 10 km in April he gets a free meal. The challenge is called "Cycle A Lot" with a description of "Everyone should cycle".
- Task 4: Go see your setup challenges and delete the "Cycle A Lot"/cycling 10 km one.
- Task 5: Setup an advanced challenge where if someone cycles to your store and wins a raffle he gets a 5euro discount. The challenge is called "Visit and Raffle" with a description of "Try your luck".
- Task 6: Setup an advanced challenge where if someone cycles 6kms in your city and visits during happy hour gets a free drink. The challenge is called "Cycle and drink" with a description of "Be happy cycling".
- Task 7: A cyclist enters the store and tells you his reward code is 6772, verify if it is true.

The first task is the initial contact with the website and encompasses both the sign on using Google and the setup of the shop. The two following tasks require the subject to create a challenge from the description by picking the correct template. For each pair of challenge creation tasks, 2-3 and 5-6, the first of the two is more straightforward than the second. The fourth task is simply to view the previously created challenges and delete one of them. The fifth and sixth tasks are similar to the second and third, but it requires the test subject to understand how the combined templates work. These also increase in difficulty. The final task is to verify if a cyclist code is valid or not.

Testing was done in rounds, and between each round, the interface was improved taking into account the results and user feedback. Each round tests five users, with three cycles done total. This is due to how usability problem detection shows diminishing returns when testing more than five users [10, 11]. Because of the diminishing returns, it is better to do three rounds of five users with improvements in between rather than testing fifteen users at once [12].

5.2. Results

The results from the third round of user tests can be seen in table 2. In each task the time taken in minutes, the number of clicks and number of mistakes were noted and the time and clicks were averaged.

Task	Avg. Time	Avg. Extra Clicks	Total Errors
1	1:47	0	0
2	1:05	0	0
3	1:21	0	0
4	0:18	0	0
5	1:59	1	1
6	1:26	0	1
7	0:18	0	0

Table 2: Results of the third round of user testing.

From these results, we concluded that is fairly fast for users to setup their own challenges. Users had no problems with the interface with exception of the advanced challenges where the first contact is not easy to understand and might require a different approach to the interface. From user feedback and notes that were taken thanks to the "think aloud" methodology, the template and forms interface were simple to understand and easy to use.

6. Requirement 2 : Flexibility

While usability is vital to not scare away potential shop owners, flexibility is still required to allow shop owners to implement the reward schemes of their choosing. With templates, this means striking the balance between not overwhelming shop owners with multiple templates and choices while still providing useful and meaningful options.

There are currently six popular implementations of reward schemes [13]. These are: Point based, Tiers, Premium, Customer values, Business partnership and Gamification. These six are more specific implementations in the implementation section. From these six, four are easily implemented using Reward servers templates. These are the: Point based, since both the distance traveled could be thought of as points, but the shop owner can also define points as a reward. Tiers can be easily created by using the template that takes into account how many other challenges the cyclist has finished. This makes possible the creation of a reward scheme where only the most loyal cyclists have the bigger benefits. Customers values, is partly implicit by the nature of reward-2-cycle since it will inspire customers that value health and the environment. By creating rewards tailored to these customers the reward scheme is even more based on customer values. Gamification like the customer values is also implicit in the reward server. This is due to the server working with challenges and progression values, very much like games do with missions and experience. The two remaining reward schemes can not be implemented using the Reward scheme.

When comparing with the reward schemes in existing solutions, the Reward server templates allow most of the reward schemes implemented in these existing solutions to be reproduced. From the total eleven existing solutions, there is a total of fifteen reward schemes. These are:

- Four implementations of "Customer must travel a specific number of km".
- Six implementations of "Customer must visit the shop a specific number of times".
- Three implementations of "Visit the shop for the first time".
- One implementation of "Raffle".
- One implementation of "Happy hour".
- One implementation of "Liking the shop Facebook page".
- One implementation of "Customer Birthday".

From these, the shop owner can easily implement the two most popular just using the templates provided: "Count the distance traveled by cyclist" and the template "Count the number of times the cyclist came to your shop". The third most popular, while not possible through the simple templates, it is possible when using the advanced challenge creation. The process is the combination of the number of visits template with "Count the number of times the cyclist completed other challenges", allowing the shop owner to create a challenge that triggers if the cyclist visits once and has never completed a challenge with the shop owner, which would be the first visit. The next two challenges have their own templates, making those also simple to be created. The remaining two, on the other hand, do not have a way to be implemented by the shop owner. Instead of these last two, it was preferred to expand the basic "distance traveled" template since the main objective is to motivate people to cycle more.

Besides the normal template selection, the advanced challenges further expand the Reward server flexibility. The advanced challenges allow combining several templates into a single challenge. From the seven existing templates, 127 unique challenges can be created as seen in the expression 1.

$$\sum_{i=1}^{n=7} {}^{n}C_i = 127 \qquad (1)$$

This allows a vaster array of challenges to be created when comparing to the simple template approach. Since the advanced challenge creation is aimed towards veteran users it can also hold more than the normal seven templates. This results in an exponential increase in unique challenges, while not flooding the advanced users with information.

It can be concluded that while somewhat limited to the templates, the Reward server is able to create the currently popular reward schemes and provide a wider array of challenges through the combination of templates.

7. Conclusions

The objective of this thesis was to provide an easy way for shop owners to create reward schemes in order to get new customers. These reward schemes, composed of challenges, would be focused at cyclists, in order to promote people to cycle more. Several solutions were explored to allow shop owners to easily create their own reward schemes. These were programming-by-example, natural language programming, Visual programming, and Templates. The template forms strikes the best balance between usability and flexibility while having a low cost to implement for the shop owner. The Reward server was then implemented as a simple web application with templates. Once implemented it was then user tested. Experimental results prove that the template approach for the Reward server achieves the usability requirement while maintaining a good amount of flexibility.

In a real world versions there are several tools and functionalities that would be deemed useful that were not implemented since they were out of scope. One of these features would be to provide the third party hosting the Reward server with an interface. This interface would allow the third party to verify if a shop owner is in fact a shop owner or to control and limit the shop owner challenges deployed.

Acknowledgements

I would like to thank my supervisor, Professor Paulo Ferreira and Professor João Barreto for the support, guidance and feedback provided throughout this thesis. I also want to thank my parents, for their love, patience and resilience. Not only during this thesis process, but also in the time spent at Instituto Superior Técnico. Finally, I want to acknowledge all the support provided by friends and colleagues. Thank you.

References

[1] Oja P., Titze S., Bauman A., de Geus B., Krenn P., Reger-Nash B. and Kohlberger T. (2011) Health benefits of cycling: a systematic review. Scandinavian Journal of Medicine & Science in Sports.

[2] De Hartog, Jeroen Johan, Boogaard Hanna, Nijland Hans, and Hoek Gerard. Do the Health Benefits of Cycling Outweigh the Risks?. Environmental Health Perspectives 118.8 (2010): 1109-116. Web.

[3] Haubold, Holger. Shopping by Bike: Best Friend of your City Centre. ECF.

[4] The ride2school handsup! system. https://www.bicyclenetwork.com.au/general/programs/350/ Accessed: 2015-10-30.

[5] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The state of the art in end-user software engineering. *ACM Computing Surveys*, 43(3):1–44, 2011.

[6] Shreenivasarao Prabhakararao, Martin Main, and Mike Durham. Software Engineering for End-User Programmers. *Test*, pages 1–8, 2005.

[7] Greg Little and Robert C Miller. Translating keyword commands into executable code. *Proceedings of the 19th annual ACM symposium on User interface software and technology UIST 06*, page 135, 2006.

[8] Mona Rizvi and Thorna Humphries. A Scratch-based CS0 course for at-risk computer science majors. *2012 Frontiers in Education Conference Proceedings*, pages 1–5, 2012.

[9] Ilona Reinekoski The Pursued Benefits Of Customer Loyalty Programs Lappeenranta University Of Technology School Of Business

[10] Jukob Nielsen and Rolf Molich HEURISTIC EVALUATION OF USER INTERFACES Technical University of Denmark Baltica A/S

[11] Jakob Nielsen and Thomas K. Landauer A Mathematical Model of the Finding of Usability Problems Bellcore

[12] Jakob Nielsen User Testing https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ Accessed: 2017-01-10.

[13] HubSpot How To Use Customer Loyalty Programs for Your Business