Improving Sales
Benefits of API Development

Axel Boström
Abstract

Improving Sales

Axel Boström

Application programming interfaces (APIs) are on the rise, drastically increasing in numbers in just a few years. For this thesis work, a remote procedure call API was developed and implemented for Responder, an online survey service. This thesis is intended to act as a base for future research on how the implementation of an API affects two key performance indicators, customer retention and customer conversion. The API was designed specifically to be integrated with a task automation service to further enhance the experience. Currently, the API is released and available to the public via the service’s website. In addition to the implementation of an API, a literature survey was conducted on mostly non-academic literature, due to the limited selection of academic works on the subject. The literature survey suggests that the implementation of an API has a positive effect on both customer retention and customer conversion.
## Contents

1 Introduction  
2 Background  
   2.1 Marketing  
      2.1.1 Customer satisfaction  
      2.1.2 Customer retention  
      2.1.3 Customer conversion  
   2.2 Application programming interfaces  
   2.3 Task automation services  
      2.3.1 Zapier  
3 Requirements  
4 Design  
   4.1 Tools and frameworks  
   4.2 API architecture  
   4.3 Visual consistency  
   4.4 Versioning  
   4.5 Security  
      4.5.1 Authentication  
      4.5.2 Authorisation  
      4.5.3 Error handling  
   4.6 Zapier application  
      4.6.1 Integration with the website  
      4.6.2 Usage  
5 Delimitations  
6 Evaluation  
   6.1 Summary  
7 Literature Survey  
   7.1 Miscellaneous  
   7.2 Zapier integration  
   7.3 External APIs  
8 Discussion  
9 Future work  
   9.1 Measurement  
   9.2 System  
   9.3 Literature survey  
   9.4 Analysis
1 Introduction

Money makes the world go round, and for businesses providing a service, acquiring and maintaining customer relationships is a successful way to make money [1]. In addition, technological advances play a major role in changing the human experience [2]. As technological advances are made, the technological ability of the average person increases, and businesses must adapt to the technological landscape in order to survive and flourish.

One such technology that has been on the rise is the Application Programming Interface (API). Over nineteen thousand APIs have been added to the online API directory ProgrammableWeb since 2005 [3], and more are added every month.

An API provides standardised access to computing services or functionality [4]. APIs used for communicating with Web services and applications are called Web APIs [5], and the purpose of a Web API is to give developers a way to remotely utilise the data or functionality that is provided by a specific service or application. For the purposes of simplicity, the term API is used in this report to refer to Web APIs.

The increasing number of available APIs combined with the pressure on businesses to adapt to an ever-changing technological landscape is an interesting subject. What impact does the design and implementation of an API have on the success of a service in terms of customer retention and customer conversion?

The thesis work is split into two distinct parts. The first is the design and implementation of an API, which was intended to act as a basis for future work on how the API inclusion affected the service as a whole. The other part comprises literature survey, which sheds light on how API implementations have affected other services.

To simplify future work, two key performance indicators were singled out as most interesting to the product owner, customer retention [6, 7] and customer conversion [8]. These two will thus be the focus of the remainder of this thesis, i.e. design and development decisions are made to improve these key performance indicators. Customer retention is a key performance indicator referring to the ability to retain customers over a period of time, i.e. the percentage of customers that return to the service. Customer conversion, another key performance indicator, is defined as the percentage of users that act in a desired manner, e.g. page visitors becoming paying customers.
The API discussed in this thesis was built for Responster, a small company that provides software as a service allowing users to create and manage online surveys. To enhance the service, the API functionality was designed to allow users to automate the distribution of existing surveys, which previously had to be done manually. The idea was to both enhance the experience of customers but also to see if it was possible to attract customers on the basis of the niche functionality of the API, but motivate them to stay for the simplicity of the service as a whole.

The API was developed to be used on the Zapier platform, a task automation service [9] that lets users combine predetermined functions of different services. Thus a Zapier application was built to handle the communication between Zapier and the API, while also providing the API with a user interface.

The quality of the API was evaluated using the ISO 25010 standard [10]. This standard contains a set of eight characteristics that can be used to describe the quality of a given software system.

While research has been done on customer retention, customer conversion, and APIs separately, little research has been done on the combination. Similar research can be found, but it mostly comprises blog posts, online articles, and case studies, see Section 7. Since the academic literature is quite limited, informal sources have been used. In addition, even though two specific key performance indicators are the focus of this thesis, other key performance indicators are included in the literature survey to highlight the benefits of API development.
2 Background

Using technological advancements for marketing purposes is a self-evident strategy; as new products are developed, innovative features or upgrades in quality are highlighted to entice customers, as explained in the book *Core Concepts of Marketing* by Burnett [11]. Burnett discusses the creation of new products, and states that the addition or alteration of features can be sufficient to call a product new.

New features are highly correlated with the success of a product, as concluded by Cooper and Kleinschmidt [12]. In their work, the authors present and review the correlation between the success of a product and a multitude of factors. The factors most correlated with success is product advantage, i.e. the product being superior in the eyes of the customer, by means of unique features, increased quality, reduced cost, or some similar property. Another factor with high success correlation is presented as project definition or protocol, meaning a detailed understanding of the target market, customer needs, product specification and requirements, prior to commencing development.

2.1 Marketing

Marketing can be divided into four strategies, as explained by Ansoff [13].

- **Market penetration**
  Increasing business performance by either finding new customers or increasing sales to existing customers.

- **Market development**
  Adapting an existing product to be able to compete in other markets.

- **Product development**
  Developing new products with different, generally better, characteristics to advance in the same market.

- **Diversification**
  A combination of market development and product development, i.e. developing new products to compete in new markets.

The efficacy of these strategies varies depending on the current market and product situation and are in reality often used simultaneously to ensure growth. Diversification, as presented by Ansoff, is different from the other three as it often requires the acquisition of new skills and facilities, while the other three build on existing resources.
2.1.1 Customer satisfaction

In *Customer Satisfaction, Market Share, and Profitability: Findings From Sweden* [14], Anderson et al. discuss how customer satisfaction affects Swedish firms. They note that the monetary value of customer satisfaction is not immediate, as the effects come in the form of future customer behaviour. Relatively small increases in customer satisfaction can result in relatively large increases in returns, meaning that attempts to increase customer satisfaction should be treated as investments, not expenses.

Anderson and Sullivan [15] define customer satisfaction as a function of perceived quality and so-called disconfirmation, i.e. customer expectations not being met. These two factors are not equally weighted, as disconfirmation damages overall satisfaction more than perceived quality strengthens it. In addition, disconfirmation is more likely to occur when the quality of the product is easy to gauge.

2.1.2 Customer retention

Customer retention is based on a customer-business relationship in which both parties benefit [16], and is linked to customer satisfaction [17]. The relationship between customer retention and customer satisfaction depends on a third property, relationship quality. This includes the perceived quality of product or service, trust in the product-or-service provider, and customer commitment to the relationship.

According to Anderson and Srinivasan businesses are only able to influence the perceived quality of a product or service and the trust the customer has for the provider of that product or service [18]. To ensure customer satisfaction, and in turn customer retention, the perceived value of customers must continuously be improved.

The average retention rate of software as a service businesses is based on the annual contract values (ACVs) of these businesses [19]. ACVs are the annualised revenue of customer contracts [20]. A contract refers to a specific service being provided. Since a customer can have many contracts simultaneously, an ACV refers to the amount of revenue a single customer contract provides. Smaller ACVs allow for lower acceptable retention rates [21].

2.1.3 Customer conversion

Customer conversion is the percentage of users that act in a desired manner, i.e. shoppers converting to buyers, or page visitors converting to paying customers [8].
In the information systems success model [22, 23] the terms system quality and service quality are defined. According to Kuan et al. [24, 25], system quality is linked to customer conversion, while service quality is linked to customer retention. System quality refers to properties such as functionality and usability, while service quality refers to properties such as reliability and support.

In a report from 2017, the average customer conversion rate for software as a service businesses was around three percent [26]. As seen in Figure 1, this noticeably lower than other kinds of services, such as e-commerce and media. The difference can be attributed to the subscription requirement for the software as a service models.

![Figure 1: The conversion rates of different service models.](image)

### 2.2 Application programming interfaces

Application programming interfaces, or APIs, are a standardised way for technologies to communicate [4]. The most common API architecture [27] is the RESTful architecture [28], based on Roy Fielding’s Representational State Transfer architectural style, followed by the Remote Procedure Call (RPC) architecture [29]. Around eighty percent of APIs are RESTful, while only around ten percent are RPC.

These architectures refer to how data and functionality is provided and accessed. The RESTful API architecture requires the correct use of HTTP verbs [30], causing URL differences to be the key factor in which data is accessed. The RPC API architecture requires knowledge about the procedures available on the server, which allows for more fine-grained functionality to be executed remotely [31].
In an article from Harvard Business Review [32], Bala Iyer and Mohan Subramaniam write about the strategic value of APIs. Using real-world examples they argue that there are a multitude of ways that APIs can generate growth for companies and that all CEOs will soon need to align their growth strategies with APIs. This is also emphasised in the 2017 report *State of APIs* [33], where it is shown that a large majority of companies view APIs as strategic in nature. In addition, the report shows that successful companies focus on implementing APIs based on use cases, while less successful companies put more focus on technical aspects of the API.

### 2.3 Task automation services

Task automation services [9] are online services that allow for service integration and task automation. Service integration refers to the ability to connect services and their functionality which would be difficult to integrate otherwise. Task automation refers to the ability to automatically execute a task in one service whenever a specific event occurs in another service.

#### 2.3.1 Zapier

The task automation service Zapier [34] refer to service integration instances, i.e. application integrations via the Zapier service, as Zaps. Like other task automation services, Zapier divides task execution into multiple steps. Zapier refers to these steps as Triggers and Actions. When a trigger event occurs, an action is executed automatically.

While Zapier is free to use, certain functionalities are improved or added by adding a monthly subscription plan. For example, using the free subscription plan, Zapier executes tasks automatically every fifteen minutes. By adding a subscription plan, that time decreases to five minutes. In addition, a user can only have a finite number of active Zaps at a time, and those Zaps can only be executed a finite number of times. The actual number of times these events can occur is dependent on what subscription plan the user has elected to pay for. If the user chooses a more advanced subscription plan, the ability to re-execute failed tasks become available.
3 Requirements

Requirements Engineering [35] is the process of defining the requirements of a system. These include, but are not limited to, the goals, functions, and constraints of a system. Eliciting requirements can be done in a multitude of ways [36], such as prototyping, surveys, and group elicitations such as brainstorming and workshops.

Eliciting requirements was outside the scope of this project, as the product owner had specified the necessary functionality in advance. Thus, the requirements stated above were given.

The required functionality of the API described and developed in this work was automatisation of survey distribution, which in turn contains the following functional requirements:

- Ability to authenticate the user.
- Ability to fetch surveys from the database.
- Ability to fetch users from the database.
- Ability to verify the user’s access to a given survey.
- Ability to send emails given correct arguments.
- Ability to authorize the Zapier integration.

The Zapier application is also based on functional requirements, to both cohere with the Zapier framework and to provide the API with the necessary data. The key functional requirements were:

- Ability to authenticate the user via the API.
- Ability to request inputs from the user.
- Ability to call the API using the required data.
4 Design

Software design can be defined as the activity following requirements specification before programming [37]. This definition is closely related to software architecture, defined by Jansen and Bosch as the set of architectural design decisions about a system [38]. There is a strong correlation between design quality and overall software quality [39], and an appropriate software architecture is necessary to ensure the longevity of any system.

Designing APIs is explored by Stylos and Myers [40] who define the two most basic qualities desired as usability and power. Usability refers to characteristics such as ease of learning, simplicity, and consistency, while power refers to characteristics such as performance, potential use cases, and lack of bugs.

When designing the API, an effort was made to maximise these qualities. Some aspects, such as ease of learning and simplicity were almost completely moved from the API to the Zapier application, described in detail in Section 4.6, as it acts as the user interface to the API.

![Sequence diagram depicting the flow of the system.](image-url)
4.1 Tools and frameworks

For the purposes of consistency and simplicity, the language, framework, and tools that were chosen for the API closely mirror those of similar projects. This also makes both code review and debugging easier, as the code reviewers are experienced in the chosen framework.

The programming language chosen for the API was PHP using Zend Expressive, a middleware centric [41] framework that allows for quick development using the provided project skeleton. This approach follows the object-oriented paradigm, and an effort was made to adhere to some of the related principles, such as the single responsibility principle [42] and the open/closed principle [43].

4.2 API architecture

Specifying the architecture of an API, as they were introduced in Section 2.2, requires an understanding of the purpose and capabilities of the system.

While the RESTful API architecture is the most common and is at times powerful, the functionality of this API was too complex to correctly utilise the HTTP verbs as required. In addition to this, the nature of the Zapier integration causes the URLs used when calling the API to be hidden inside the Zapier application. This mitigates the inherent problem with the RPC architecture having to explicitly know the remote procedures available. For these reasons, the API was implemented as an RPC API.

4.3 Visual consistency

It was decided to keep the same visual style as the original website’s login page for simplicity and consistency. To achieve this the CSS of the original login page was replicated and used in the design of the login page of the API. In addition, the emails being sent follow the same visual characteristics as emails which can be sent directly from the Responster website.

4.4 Versioning

One important aspect of API design is versioning [44], i.e. how updating the API is handled. Developers using the API would need to manually update their implementations as new API versions are released. However, due to the design of this API, the versioning is hidden behind the Zapier application, removing a layer of complexity for users.
4.5 Security

As the API accesses details about the user and related surveys, it was important to ensure both the account security of the user and the integrity of the service. General measures were taken to achieve this, such as using HTTPS instead of HTTP and displaying appropriately formulated application error messages rather than displaying server-side errors to the user. In addition, authentication and authorisation were implemented.

4.5.1 Authentication

Authentication in web services refers to the process of reliably associating a user or client, connecting the session to an existing account [45]. There are multiple types of authentication [46], each of which utilising one or more authentication factor to validate the authenticity of the user. These factors are:

- **Knowledge factors**
  These refer to factors that the user knows, e.g. a password or a personal identification number (PIN).

- **Ownership factors**
  These refer to factors that the user owns, e.g. a bank card or an identification card.

- **Inherence factors**
  These refer to factors relating to who the user is, e.g. fingerprints or facial recognition.

The Responster service uses single-factor authentication, meaning only one of the above factors is necessary to successfully authenticate the user. It requires an email and password combination, i.e. a knowledge factor. It was decided to mirror this form of authentication in the API, which enabled the use of the login functionality provided in the Zend Expressive framework.

4.5.2 Authorisation

Authorisation refers to the process of granting a client access rights to specific resources [47]. This can be done implicitly during the login process or explicitly by means of some authorisation framework mechanism.

For this API, authorisation needed to be explicitly handled, as the idea was to only require the user to authenticate and authorise the Zapier client once. This allows the Zapier client to execute tasks automatically without external intervention while continually being able to prove its authorised status to the API.
Zapier has support for a limited number of methods for authentication and authorisation. It does, however, support the OAuth 2.0 authorisation framework [48], but only when paired with the Authorisation code Grant [49], which inserts an additional step compared to other grants. Before being issued an access token, users must authorise themselves to be issued an authorisation code. This can then be traded in for an access token, which is used to gain access. As the OAuth 2.0 authorisation framework has been used by the developer team in previous projects, it was the obvious choice.

4.5.3 Error handling

Another key aspect of API design is properly handling errors [44]. An API is essentially a black box for developers apart from the defined functionality, making properly defined error messages important for both development and debugging.

When an error is encountered by the API it returns an HTTP status code [30], which is then interpreted by the Zapier application and displayed to the user via the Zapier userface. The three main status codes used are:

- 200 – all requests that do not encounter any errors,
- 401 – when the client is unauthorised to access the requested resource,
- 404 – when the requested resource is not found.

4.6 Zapier application

In addition to the development of the API, an application for the online service Zapier explained in Section 2.3.1, was built. This application was written in JavaScript using the Node.js environment, allowing for server-side execution as required by the Zapier service.

The purpose of this application was to serve as the interface between the user and the API. As explained in Section 4.5.2 above, the Zapier application was responsible for authenticating and authorising the client. This was done as specified by the Zapier documentation, redirecting the authenticating client to the necessary URLs.

Zapier applications follow a specific pattern [50] to ensure that the application adheres to the structure of the Zapier website, meaning the Zapier application required fewer design choices than the API.
4.6.1 Integration with the website

Each survey on the Responster website has a corresponding share section, containing functionality to help the user share their survey. A link to the Zapier integration was added to this share section, together with an explanation of how to use it.

To ensure that the quality of the API was satisfactory, a decision was made to provide the service to all Responster users on release rather than just paying customers. This allows for more rigorous testing as the functionality will reach more users, but will be changed in the future to only be available to paying customers.

4.6.2 Usage

Using the Zapier application and API requires three things, a Zapier account, a Responster account with a published survey, and access to the Zapier application. At the moment, accessing the Zapier application requires a direct invitation, which is required by Zapier until the application has a minimum of ten active users.

On the Zapier website, the user first creates a new Zap, described in Section 2.3.1. The user chooses a suitable Trigger and configures it as required. The user then chooses the Responster Action, authenticates the client as instructed, and configures the properties of the email to be sent. These properties include the wording of the email, a potential logo to be included, and the URL of the survey to be sent. During each step of the Zap creation process, the user can test the chosen Trigger or Action, allowing for easy configuration according to needs.

Identify need for automatic surveys
Trigger exists on Zapier and you want to use it?
Create a Zap with trigger and Responster action
Provide Responster action with information
Finish Zap and enjoy automatic surveys
No
Implement trigger on Zapier
Choose a trigger
Create account on Zapier and log in

Figure 3: A broad depiction of the user workflow.
5 Delimitations

Initially, the API functionality was intended to include two procedures, one being automation of survey distribution. The other targeted automatically fetching new survey results, making them available to other integrated services. This idea was excluded from the scope of this thesis due to time limitations, and the need to have a product to test and launch in a reasonable amount of time.

Another idea that was initially intended for implementation but was ultimately discarded due to time constraints was the ability to fetch all available surveys via the Zapier application and display them to the user when choosing which survey to send. This would allow the user to bypass accessing their Responster account entirely.

Unit testing was also considered outside the scope of the project, as the two parts of the system, the API itself and the Zapier application, were difficult to unit test. As the functionality was so niche, the individual parts of the system all depended on each other in some manner, making it easier to just test the system as a whole.
6 Evaluation

According to the International Organization for Standardization, or ISO, software quality is based on eight quality characteristics [10]. These characteristics are functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. Each of these characteristics is divided into sub-characteristics, which should be present in order to ensure that the needs of the stakeholders are met.

What follows is a thorough evaluation based on the software quality model described above. As the API is integrated with the Zapier application, they will be evaluated together. The definitions of these characteristics are summarised on the ISO 25000 portal [51].

**Functional Suitability**
This characteristic refers to how much of the required functionality is provided by the system.

- **Functional completeness**
  The API provides users with one accessible procedure, usable via the Zapier application. As this was the only required functionality of the system, the API can be considered functionally complete.

- **Functional correctness**
  The API is only required to send an email to a recipient, given the correct inputs, which it does. As these inputs are either required from the user or inferred by the Zapier application the API can be considered functionally correct.

- **Functional appropriateness**
  As the API provides new functionality to the service, i.e. automating the distribution of surveys, the API can be considered to accomplish that task.

**Performance Efficiency**
This characteristic refers to how effective the system is, i.e. what performance it achieves in relation to the resources it uses.

- **Time Behaviour**
  Because the API is meant to be activated automatically, and because of the implicit timing issue with Zapier as explained in Section 2.3.1, the API does not have any time constraints on execution, as long as it executes. Thus the timing behaviour of the system can be considered correct.
• **Resource Utilization**  
As the expected traffic through the API and Zapier application is low, the expected usage of resources is minimal.

• **Capacity**  
If the system were to be used at its maximum limits, the mail server used to send emails would probably be affected the worst, as the messages sent and received by the API and Zapier application are very small. However, the mail server is not within the scope of this project.

**Compatibility**  
This characteristic refers to how well the system interacts with other systems.

• **Co-existance**  
The API is able to co-exist with other systems, given that these systems do not overload the server.

• **Interoperability**  
Because of the narrow scope of the API, and that its only procedure is handled largely as a side effect, i.e. something not returned to the requester, it is currently incapable of operating with other systems.

**Usability**  
This characteristic defines how well the system achieves the specified goals in the context of efficiency, effectiveness, and satisfaction.

• ** Appropriateness Recognizability**  
Because of the niche nature of the system, with users most likely being developers at businesses that utilize the Responster service, it is very likely that if they are using the API, they also recognise that it is appropriate for their needs.

• **Learnability**  
As the user interface of the API is provided by the Zapier application, learning how to use it is the same as any other Zapier application. The wording of the Zapier application functionality was chosen for clarity and consistency.

• **Operability**  
Similar to the learnability sub-characteristic above, operating the API is largely outsourced to the Zapier application.

• **User Error Protection**  
The inputs that are required by the API are either required by the Zapier application or have default values, i.e. they cannot be empty. In addition, all inputs are filtered, custom error pages are implemented, and the authentication functionality provides accurate error messages on incorrect credentials.
• **User Interface Aesthetics**
  This is provided by the Zapier application, with the exception of the authentication functionality which mirrors the aesthetic characteristics of original Responster service.

• **Accessibility**
  While the API is usable by anyone, it does require some effort. Users are required to register to Zapier and Responster.

**Reliability**
This characteristic refers to how the system performs over a certain period of time.

• **Maturity**
  Not enough testing has been done in a live environment to provide an accurate evaluation of how the system performs under normal operation.

• **Availability**
  There are two aspects necessary for the API to be available, the server on which the API is hosted and the Zapier service. These are under normal circumstances always online, making the API always available.

• **Fault Tolerance**
  If the server or the Zapier service are put offline, the API will not function, making fault tolerance an unavoidable weakness.

• **Recoverability**
  If an error is returned to the Zapier application by the API during automatic execution that particular set of inputs is lost [52]. Users can try to remedy this themselves by becoming paying customers of Zapier, which allows them to re-execute tasks.

**Security**
This characteristic describes how well the system handles data access needing authorisation and how well it protects information.

• **Confidentiality**
  Every execution checks if the requested survey belongs to the authenticated user, i.e. confidentiality is upheld.

• **Integrity**
  Authentication and authorisation is explained in Section 4.5. As there is only one procedure, there is only one level of authorisation.

• **Non-repudiation**
  Zapier keeps track of previously executed tasks, meaning no repudiation can take place.
• **Accountability**
  There is currently no logging of API requests on the server side, which should be implemented. On the client side, Zapier keeps track of the tasks being executed.

• **Authenticity**
  Authenticity is achieved by authenticating the user. The data the user provides is, while filtered and in some cases required to be on a certain form, entirely optional and mutable.

**Maintainability**
This characteristic describes how easily and efficiently the system can be changed to adhere to new environments or requirements.

• **Modularity**
  The API is mostly built on modular parts, attempting to adhere to the single responsibility principle, as mentioned in Section 4.1. However, some parts are built to be used in conjunction. Changes to functionality need to be mirrored in the Zapier application. As the API functionality is hidden behind different URLs, adding procedures is easy.

• **Reusability**
  Since the API implementation is mostly modular, reusing components, or parts of components is easily achieved.

• **Analysability**
  As the API is both in its infancy and relatively small in scope, analysis of faulty code and potential subjects for modification is effective.

• **Modifiability**
  Given that the expected inputs and outputs of previously existing functionalities are correctly changed in accordance with the modifications, the mostly modular nature of the API ensures that modifying the implementation is efficient and effective.

• **Testability**
  While tests can be written for the Zapier application itself, the RPC functionality of the API is difficult to test on its own. For this purpose, as stated in Section 5, unit tests were not included in the project. However, the system flow has been manually tested by providing input data to the Zapier application and verifying that the email is sent correctly.

**Portability**
This characteristic describes how easily the system can be transferred to another environment, be it hardware or software.
• **Adaptability**
  The API is hosted on an Apache server, from which migrations to other server types are possible.

• **Installability**
  As the system is web-based, it is usable from anywhere as long as it has a functional browser to set up the required Zap.

• **Replaceability**
  The API is built ad hoc to be integrated with Zapier, meaning that there is currently no software to replace.

6.1 **Summary**

The system comprises two entities, the API, and the Zapier application. The API provides the functionality and the Zapier application provides the user interface.

The visual properties of the system in its entirety are largely inherited from the Responster and Zapier services, i.e. how the login page and the user interface provided by the Zapier application looks. Some of the quality characteristics mentioned above, such as learnability and operability, are thus largely outsourced to the Zapier service.

Internally the system is structurally sound. The functionality is correct and complete according to the requirements, errors are handled properly, and the modification and testing of the system are easily done. The server running the API is directly linked to the original Responster service and is thus outside the scope of this project.

No internal logging of the execution of tasks is currently being done, though that is provided by the Zapier application. In addition, the niche nature of the API makes it difficult to integrate it with other systems in ways other than the intended ones.
7 Literature Survey

This literature survey comprises mostly non-academic articles, as the academic literature relating to API development and key performance indicators is limited. It was also conducted while prioritising customer retention and customer conversion, as these are the key performance indicators of interest. However, works relating to other key performance indicators are included as well.

7.1 Miscellaneous

In a Nordic APIs blog post from 2014 [53], Bruno Pedro lists five benefits APIs will have for a business’s revenue. Among these are increasing customer retention and creating "upsell" opportunities. The term "upsell" in this case refers to the act of persuading a user to purchase the required access to one or many features, i.e. customer conversion. In a similar article from 2016 [54], increasing retention and conversion was the number one benefit of adding an API to a service.

The keys to successful API strategies are discussed in an article by Mark Boyd from 2015 [55]. These include, among other things, developer support, developer community size, API discoverability, and tools to make the API easy to use and learn.

In June of 2009, only fifty-five percent of Twitter users directly used the actual service, while the remaining forty-five percent of users used some third party application connected to the Twitter API [56]. The most used third party application was TweetDeck with around twenty percent of the user base. In 2011, Twitter bought TweetDeck for around forty million dollars [57].

7.2 Zapier integration

In the Zapier blog post Do Apps That Integrate With Zapier Have Lower Churn? by Christopher Peters [58] the churn rate of Typeform, a company similar to Responster, was examined. The churn rate, i.e. the inverted retention rate of users that use only Typeform, was compared to users that combine Typeform and Zapier. The results show a ten percentage point difference between the two after one year, with the Zapier and Typeform combination having a retention rate of eighty-five percent. The increase in retention is one of the properties that Zapier explicitly state as a positive aspect of using the service [59].
7.3 External APIs

In a case study of the company AwardWallet implementing the Rewards API [60], the API was credited with improving the quality of the service by improving security, overall reliability, and processing speed. As explained in Section 2.1.1, service quality is linked to customer retention, which leads to the logical conclusion that the implementation of the API had a positive impact on the customer retention of the service.

The company Next Jump, specialising in improving workplace culture, enjoyed a three-time increase in customer conversion [61] when they started using the Walmart Open API. This was done by removing several steps of the purchasing process, improving the perceived quality of the system and in turn conversion rates as well.

Another company that noticed an increase in conversions when including an external API was Dynamic Creative [62]. They observed a 150 percent increase in conversion when integrating the AdWords API into their advertising platform.

JustGiving, a company that provides an online charity platform, implemented an API in 2011 [63]. Since that time, about twenty-five percent of annual revenue is driven via the API, and over fifty charities utilise the API globally.

With the inclusion of an API, FightMetric, an online provider of mixed martial arts data and statistics, was able to ensure reliable data access to important customers [64]. They also increased the total number of website hits from 250 thousand to twenty million over the course of three years, between 2013 and 2016.
8 Discussion

This thesis has focused on the design, development, and evaluation of an API. It is intended to act as a case study for evaluating the effect API development has on a business from a sales and marketing perspective. The API, which is based on an RPC architecture, included one procedure, namely the ability to send surveys via email. In addition, a Zapier application was built to be used as the user interface for the API.

In addition to the development of the API, a literature survey has been conducted on works relating to similar technologies and marketing aspects as this one, to highlight the effects API development can have. Works relating to the key performance indicators customer conversion and customer retention have been prioritised, but other works have been added as well to paint a wider picture for the reader.

The architecture of the API is RPC, meaning it contains specific procedures that are called remotely. As only about ten percent of available APIs are RPC while around eighty percent are RESTful, most related works are studies on how RESTful APIs impact businesses, not RPC APIs. It is not clear whether the studies on RESTful APIs can be translated directly to RPC APIs, as RPC APIs are more niche in nature and are thus more likely to have a niche user base.

While the API utilises an RPC architecture, it is built specifically to be interfaced by a Zapier application. This abstracts away some of the niche nature of the API, as the user is never allowed access to the API directly, but must access it through the Zapier user interface. As mentioned in Section 7.2, the service Typeform is similar to Responster in its functionality, but has a RESTful API that is interfaced by Zapier. The user base that uses this integration has a retention rate of eighty-five percent, while users that do not use Zapier only have a retention rate of seventy-five percent.

This increase in retention in the Typeform integration with Zapier shows that at least RESTful APIs benefit from integrating with Zapier. However, it does beg the question of why the retention rate increases with the integration. There are two possible reasons for this.

1. The integration provides a sufficiently large quality improvement that users who try it are inclined to continue using the service at a higher rate than users who do not try it.

2. Users that use the integration are already committed users of the service, and the integration is just another usable feature, essentially just a bonus.
The difference between these two is that the first reason implies that the API integration itself is the cause of the increase in retention. The second reason implies that users who utilise the API integration are already devoted to the service and that they as a group would have a higher retention rate regardless of the API implementation.

Albeit, both of these reasons likely play a role. It could be argued that the API architecture plays a part in which reason is more prominent. RESTful APIs are often more generally usable than RPC APIs, and thus are more likely to be considered just a bonus feature in the eyes of the users. RPC APIs, on the other hand, are more likely to be specifically requested by users, i.e. requests for niche or ad hoc features. When the requested features are implemented they improve the perceived quality of the service, which in turn is linked to customer retention.

The potential power of APIs, when used correctly, was shown by the service called TweetDeck, see Section 7.1. By using Twitter’s own API, the user experience it provided was different enough that twenty percent of Twitter users used it instead of the regular service. This immense user base caused Twitter to buy the company for around forty million dollars, showing that correctly utilised APIs can be the root of success.

As mentioned in Section 7.3, the company Next Jump had their conversion rate tripled by using an API that simplified the purchasing process. However, as opposed to Responster which uses a subscription-based payment method, Next Jump users buy specific products and services directly. This difference in payment method could play a large role in how the conversion of a service is affected by the inclusion of an API. One possibility is that API inclusions have a larger effect on e-commerce based companies such as Next Jump, as the commitment required is much lower than for subscription-based services.

Both Next Jump and the company Dynamic Creative, see Section 7.3, noticed an increase in conversions when integrating an external API with their service. Using external APIs allows services to focus on providing their specialised functionality at the highest possible quality while outsourcing other aspects to APIs that do the same. The root of the perceived quality increase can be assumed to be the addition of high-quality functionality in place of low-quality functionality or lack of functionality. This translates to the Responster API quite well, as the inclusion of the API is a pure quality increase to users who wanted it, while others see another feature being added, which can also be considered an increase in perceived quality.

Regarding the works found in Section 7, out of the ones that are case studies purely on APIs, many are case studies on one of two things. They can ei-
ther be on how integrating an external API has affected the service, or how contracting another company to create and provide an API has affected the service.

The integration of external APIs is essentially the key aspect of APIs in general. As the functionality provided is so distinct, service providers are able to pick and choose functionality from different APIs to produce the best possible service. However, when the functionality required is too specific to the service, the other alternative is available, i.e. contracting another company to create the API for the service. This is essentially what Responster did, as the functionality required was so specific to the service.

An issue with using non-academic literature is bias, especially when utilising case studies, as they often written by the company trying to promote their service. However, while the rhetoric may be misleading, the numbers produced are not. In addition, non-academic literature is less likely to be written at all if the endeavour in question was a failure.

Analysing the findings of the literature survey as a whole, it could be argued that the API implementation together with the Zapier application should increase retention. This is due to the fact that a similar situation has been documented, albeit using a RESTful API instead of an RPC API. However, whether the API will increase conversion is less clear, as there is a severe lack of studies on this particular situation, i.e. a SaaS-based service integrating an RPC API.

Given this, and given the knowledge on what causes increases in retention and conversion, both retention and conversion should increase, as the overall experience of the Responster service is likely to improve. The addition of more functionality should improve the perceived quality of the system, even to users who do not intend to use it.
9 Future work

As this thesis is meant to act as a basis for future research in how the development and implementation of an API affects a service, the most logical direction to take in the future is the analysis of the finished product, i.e. measuring how the relevant key performance indicators have changed after a sufficient amount of time has passed.

In addition to the key performance indicators that are the focus of this thesis as per the product owner’s request, more key performance indicators could be added to the analysis of the system to paint a clearer picture on the benefits of API development.

Apart from the analysis of the system, there are two possible avenues for future work for this thesis. These are the improvement of the system and the improvement of the literature survey.

9.1 Measurement

While the API is live and available on the Responster service, it does require a certain amount of time before the acquired data can be considered accurate enough to measure the relevant key performance indicators. The absolute minimum amount of time that the product owner is comfortable with is three months, but the more time that passes, the better.

There are essentially two ways to measure these key performance indicators, one directly related to the API and one more related to the service as a whole. The same measuring strategies can be used for both key performance indicators.

1. Compare the key performance indicators of the group of users that use the API with the group that does not. This is the direct way of measuring the effect the API has on the key performance indicator, and should thus be prioritised.

2. Compare the overall key performance indicators before and after the implementation of the API. This is the indirect way of measuring the effect the API has on the key performance indicator, as it is not necessarily the case that the API has a clear influence on this measurement. However, it is interesting to see if the API causes a change in the trend of the key performance indicator.

One thing to note about the measuring of the key performance indicators is that the service currently provides the API to all users rather than to only paying customers. This means that, while retention can be measured
as soon as a sufficient amount of time has passed, the measuring of conversion requires the API to be added to a payment plan, so that only paying customers are provided access.

9.2 System

Potential improvements to the API as a system come in many forms. While some require a large amount of work, some features can be added quite easily.

- Allow Zapier to automatically fetch available surveys, see Section 5. This could be included in all other Zapier applications that handle specific surveys.

- Allow users to automatically fetch new survey results, see Section 5. It would be reasonable to include some functionality for fetching all available survey results on first execution.

- Provide functionality for simple, personalised surveys to be created automatically through Zapier. This could, in turn, utilise the current API functionality, i.e. automatic survey distribution.

- Provide functionality for automatic and dynamic survey alteration, i.e. allowing users to alter their surveys without having to explicitly visit the Responster website.

In addition to the aforementioned improvements, which are just added functionalities and can thus be added quite rapidly, a RESTful section of the API could be added. As the RPC architecture narrows the usage of the API to only be usable via Zapier, a RESTful section could be added which could be used without going through Zapier. This would require a large amount of work, but would also add flexibility to the system and would most certainly add to the overall experience.

Adding a RESTful part of the API would also align the Responster API more with the APIs found in the literature survey in Section 7. This would improve how well the literature survey applies to the effects of the actual implementation, making it easier to make accurate assumptions on how the key performance indicators would change.

9.3 Literature survey

The nature of literature surveys, at least the kind presented in this thesis, makes it so that more literature can always be added. For this reason, there are two possibilities for improving this particular literature survey.
1. Improve the literature survey depth-wise. This would entail the addition of new literature that is released while also delving deeper into how the particular key performance indicators are affected by similar projects.

2. Improve the literature survey breadth-wise. This would entail being more liberal in terms of what literature is considered to be relevant.

The alternatives to improving the literature survey, i.e. improving the system itself or improving the analysis of the system, have greater potential impact, as conclusions have already been drawn from the literature survey. Thus, it would be wise to prioritise these over the improvement of the literature survey.

9.4 Analysis

Improving the analysis of the system can be done by adding key performance indicators to more accurately gauge the effects of the system. Only customer retention and customer conversion were chosen because those were the key performance indicators of interest to the product owner of the system, but any key performance indicators can realistically be used as well.

One such key performance indicator that could be of interest is unique website visitors, which refers to how many unique visitors the website has over a set period of time. This would be interesting in order to gain an impression of if the Zapier application being visible to other Zapier users has an impact on the service.

Adding more key performance indicators to the equation would elicit some alteration to the literary study breadth-wise, as described in Section 9.3 above.
References


