POLITECNICO DI TORINO

Corso di Laurea in Computer Engineering

Tesi di Laurea Magistrale

Chatbot integration within Sitecore Experience Platform

Relatore:
prof. Guido Albertengo

Riccardo Di Vittorio
matricola: 231694

Supervisore Aziendale
Luigi De Martino

Anno accademico 2017-2018

Summary

Nowadays users want to feel themselves considered important and to receive a dedicated treatment from technological experiences as well as between people in the real life. The the main purpose of the thesis' project is the development of a chatbot to integrate within a Sitecore's website in order to increase the user engagement with respect to the brand, exploiting personalization mechanisms.
The work included the analysis of the Sitecore technology, in order to understand how it works as Content Management System and how it is able to deliver a personalized experience in the websites that use it. Than has been analyzed how a chatbot should improve the user experience within a website, looking at their state of art, and the possibilities for developing a new one. Finally has been studied and exploited the Microsoft Bot Framework to speed up the chatbot's creation and development.
The thesis is composed by five chapters in which are illustrated the state of art of the technology studied and how it has been used for the development of the targeted project. In the first chapter is done an overview of the Content Management Systems, what are, how they work and how they can be categorized in different types, focusing then on Sitecore and its main characteristics. In the second one is presented the chatbot's world: starting from the first bot developed until nowadays, explaining their history, how they are evolved, the used technology and how they are used, distinguishing different ways to work. Going on, the next chapter is about the Microsoft Bot Framework, a powerful tools for creating, developing and managing chatbots, with a

detailed description of the main components of this framework, in order to understand how it works. Having done an overview of the state of art of the technology used in the work, the fourth chapter shows the project implementation, how the shown technology has been used and how it was chosen to operate, the choices done and what has been developed, with the addition of code snippets for a more detailed view on the work done. Finally, the last chapter is dedicated to the conclusions and further works, with the summary of what has been done, the goals achieved and an analysis of what is missing and what could be added in future.

Contents

List of Figures

Chapter 1

Content Management Systems

The development of a website passes first of all from the definition of what will be its use in terms of quantity and complexity of the information to be represented, the need or not to have updated contents and how the user can interact with them. Hence the choice between a static or a dynamic site. The first one is characterized by an HTML markup language, through which little flexible pages are created as far as updating and maintenance are concerned, but which at the same time have reduced loading times. The user can use these pages only in a consultative way, without a real interaction that leads to the modification of the pages themselves. The second one, instead, offers the possibility to manipulate the content dynamically going to query the databases in real time in order to obtain the required content, using languages such as ASP.NET, PHP or AJAX. In this case the loading times will be longer due to latency between the data request and the response from external databases but it will be possible to establish an interaction with the user. While on one hand the dynamic sites allow better content management and greater involvement of the user, on the other hand, the construction and maintenance of this type of sites requires a greater implementation effort due to the need to control and update the material provided in them. Traditionally the task

of uploading and updating site content is fell to the IT department, but sometimes a web page may consist of several areas handled from different departments in the company, so the additional difficulty of managing the access to the same resources by several people must be considered. To overcome this, specific applications were born for the management of websites without the need for high technical connotations, called web content management systems, or CMS. A content management system is a software package that provides some automation to the tasks that are required to maintain content. It is composed by many parts, like an editing interface, a repository, and several publishing mechanisms, even if from outside it might appears as a single block for non-technical users. A CMS is usually server-based, multiuser software which interacts with the content stored in a repository in the same server or in another one allowing editing, controlling and reusing it. Based on the content type is possible to distinguish four categories of CMS, though they are not exactly separated and independent among them:

- Enterprise Content Management (ECM): is a combination of strategies, methods and tools used for the organization of general business information as employee resumes, memos or incident reports, that has to be used by a designated audience.

- Digital Asset Management (DAM): is a software solution that allows enterprises to store, organize and share digital content, like images, audio, video and presentations.

- Records Management: it concerns the creation and management of records that are part of the business transactions, identifying relevant information to be used for record classification.

- Web Content Management: manage material designed to be published on a website. [3]

Core functions of all Content Management Systems are the possibility to establish the necessary permissions to editing, deleting or simply see

the content and keeping published content separated from the repository where editors work, so that revised content remains unpublished until it's ready to be launched. At the same time the system automatically tracks all the changes and allows to see when it was done the last modify and how the current version is different from the previous one. Furthermore it encourage content reuse and provides dependency management, so that when a content is modified or deleted, also all the references to it are modified accordingly.
Going into more detail about Web Content Management Systems (WCMS), these typically are web applications that provide browser-based user interfaces in order to allow to maintain the content, and its presentation, of one or more websites. The primary goal of these systems is to provide means by which even non-technical users can maintain the content of their website, including the consistency of what is shown, as well as the reuse of the content not only in several pages but also in different websites, adapting the presentation to the different characteristics of the various devices (for example different browsers could use different markups, and the same content will be arranged differently depending on whether you use a smartphone rather then a tablet [26].
Actually many different CMS are available on the market, for example:

- WordPress: an open source content management system launched in 2003 and based on PHP and MySQL.
- Drupal: another open source software written in PHP, that can be used to easily create and manage websites that include blog, forum, e-commerce sites and social networks.
- Joomla!: it is a free open source content management system based on Mambo (the previous CMS) and is written in PHP, while stores data using MySQL.[18]
- Sitecore: a powerful CMS built on ASP.NET, that enables web content editors and marketers to have full control over all aspects of their website.

Among those listed above, we focus on Sitecore technology.

1.1 Sitecore

Sitecore is a global software company founded in 2001 as spin-off of another consultancy company, and become one of leading providers of customer experience management, digital marketing and e-commerce software. It offers a complete ASP.NET development platform and a web content management system that uses a flexible and hierarchical data storage, that give to the developer the possibility to speed up the website creation and maintenance.
Starting from pure web content management software, the company has expanded its offering concerning digital marketing allowing Sitecore's customers to personalize and tailor the provided content to individual visitors on different channels. The importance of the user experience in the sales process has been confirmed by several studies, including a survey made by American Express and reported by Glance in which states that «91% of customer who had a bad experience won't willing do business with your company again», thus the importance of investing to improve the user experience and establish an ongoing relationship with him. In support of this we can refer to another survey done by Gartner in 2016, who reports that «89% of companies expect to compete mostly on the basis of customer experience, versus 36% four years ago».
To meet these expectations Sitecore offers solutions leveraging on artificial intelligence and machine learning to help its customers to design marketing activities that are increasingly targeted and personalized for individual visitors [7], including the Sitecore Experience Platform launched in 2015.

1.1.1 Sitecore® Experience Platform™

«Marketers need three key elements to deliver relevant, personalized experiences,» said Michael Seifert, CEO, Sitecore. «First, they must understand their customers, who they are, what they care about, and what they are doing. Second, they need a single content management

system that can share contextualized content across channels to customers. And third, they need to deliver communication automation, the ability to map profile and real time behaviors with on the spot decision making to customize the experience as it happens across touch points.»
Sitecore Experience Platform (XP) helps businesses to know better their customers, using key attributes like location, device used and purchase history in order to offer them targeted content, selected based on the acquired information, and present them in the most adequate [21]. It is a customer experience management software containing the Sitecore CMS, the Experience Database (xDB) and Experience Marketing applications that allow the unification of activities across multiple channel, like campaigns, visitor activity and performance measurement.

Sitecore's architecture

Sitecore's architecture can be split into five main blocks: Channels, Management, Sitecore AIDA, Database and Integration.

Channel Layer Users interact with a brand by means of channels. Those available are web, e-mail, mobile, social, commerce, print, apps and federated. Sitecore provides useful tools for managing the content available on each channel, such as an Email Experience Manager, that can be used for creating personalized email campaigns and tracking user interaction with them, the Print Experience Manager, for dynamically creating print assets like manuals and brochures by means of familiarly design tools, the Federated Experience Manager, for adding Sitecore content and tracking visitor interactions on external non-Sitecore websites, the Social Connected, for social networks integration and statistic tracking, in addition to the Sitecore Commerce Connect, that allows e-commerce solutions to use customer engagement to provide personalized experiences thanks to the customer's behaviors tracking.

Figure 1.1. Sitecore Experience Platform

Management Layer This is the core of the Sitecore Experience Platform. It gives the possibility to manage the content of the website, configure the personalization rules and view the engagement scores of the customers. To access these functionality, Sitecore provides tools such as the Media Library, the Content Editor and the Experience Editor, that users with log-in credentials can use through the Launchpad.

Sitecore AIDA Layer Sitecore AIDA (Analytics, Insights, Decisions, and Automation) is a framework that allows to collect and analyze the behaviors of the users that interact with the brand across the several channels. It gives the possibility to perform different tests of the customer experience, track how users have interacted with the website and may how they have achieved some defined goals. All these information are then available in different reports, from which you can extract meaningful user's characteristics and use them to associate visitors to available pattern, delivering personalized content based on their

interests. Using rules is also possible to dynamically create customer segmentation based on real time information. Finally the automation process allows to establish a one-to-one relationship with all the users responding immediately to their actions.

Sitecore Database Layer In order to improve user experience delivering personalized, channel optimized content, a large amount of data has to be stored in several big databases. Sitecore's flexibility allow to have several deployment options, concerning low-traffic solutions as well as high-traffic solutions with higher availability. However, in all possible solutions the key components of the Sitecore Experience Database (xDB) are always the same:

- Content delivery server: this Sitecore application server is the one aimed to deliver website content serving incoming HTTP requests from the network. For improved scalability and better performance is possible to have multiple instances of this server in different geographic locations in order to offer the website content to a high number of visitors.

- Content management server: this type of server enables content editors to create, modify and publish content on the website.

- CMS databases: are three SQL Server databases that host data available on the website. Going more in detail there is the Core database, that contains all configurations and settings about users and roles, the Master database, where are stored all versions of all website content and where it can be created modified or deleted. Finally the Web database is the one responsible for the live site, indeed all editing actions done on the content are not propagated to the website until content author publish it.

- Session state server: this component is a session state store used by the delivery server for the personalization process. Really it is an ASP.NET session state store provider.

- Collection database: it is the main repository for storing interaction, contact, history and automation data as well as devices, location and triggered events. This database is implemented with MongoDB, so a NoSQL document-oriented database where is possible to store unstructured data and retrieve them faster then in SQL databases, improving availability and scalability.

- Reporting database: it contains aggregated data from the Collection database. This data are generated by the Sitecore aggregation pipeline and then used by Sitecore reporting applications. When processed, data are stored in several tables organized in a star schema where at the center there is a fact table containing foreign keys for joining the dimension table around the schema.

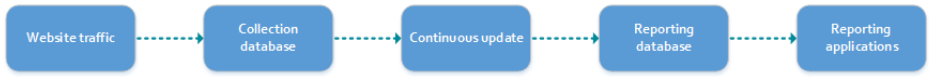
- Processing server: this is a server connected both to the Collection and to the Reporting databases, aimed to process and aggregate data from the first one and store them in the second in a form suitable for the reporting applications. Several types of processing data are available:

  - Aggregation: this processing extracts data from the Collection database, then groups and reduces it before storing it in the Reporting database.

  - Continuous update: is a continuous process that starts with the launch of Sitecore and keeps the Reporting database up to date with the most recent interaction as long as the Experience Database is running. Latest interactions are saved into the Collection database, and added to the processing pool waiting for the aggregator, which pushes them into the processing pipeline where they are transformed in a form suitable for the Reporting database and finally merged with the existing data into it.

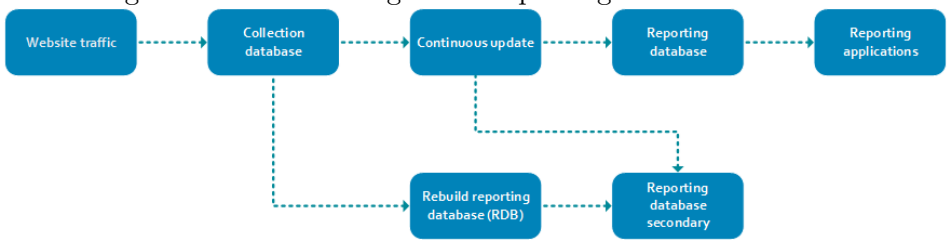
  - Rebuilding the reporting database: is a process that can be requested for ensuring to have the Reporting database

Figure 1.2. Continuous update processing overview



updated with the latest interactions. This operation completely override the content in the Reporting database and require some time, thus, in order to minimize the interruption of reporting functionality, is performed with the help of a secondary database (reporting secondary) that will replace the existing Reporting database once the rebuild process has finished.

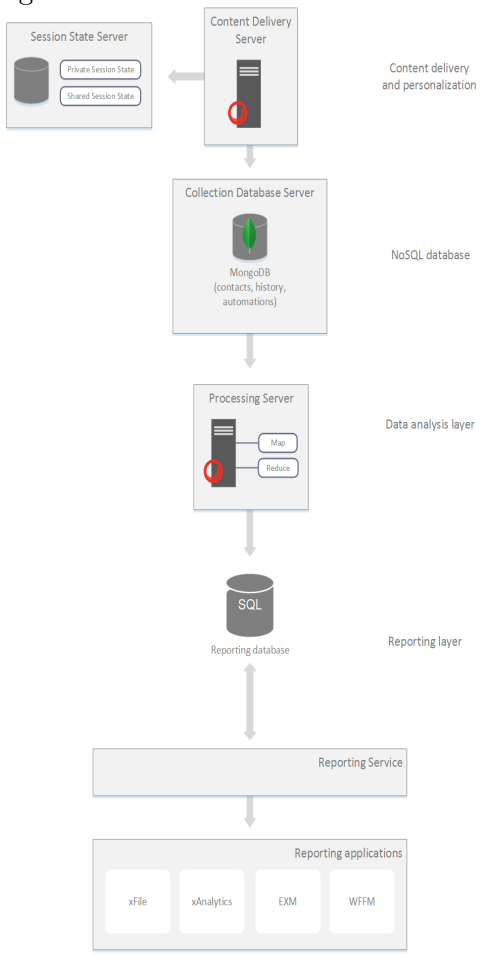
Figure 1.3. Rebuilding of the reporting database overview



  - Maintenance: this process concern maintenance tasks on the Collection database. [20]

- Reporting service: this allows to retrieve information from the Collection and the Reporting databases using the Reporting Service API. However, is recommended querying the Reporting database because it contains information in a form optimized to be queried, so that writing and reading operation are very fast. These queries are performed by Sitecore's reporting applications like Sitecore Experience Profile (xFile), Engagement Analytics reports (standard Sitecore reports), Executive Insight Dashboard (overview analytics data reported as charts and dashboards), Email Campaign Manager (ECM) and Web Forms for Marketers (WFFM).

Figure 1.4. xDB architecture overview



When a user starts navigating through the website, depending on his geographic location he is redirected to contact an appropriate Content delivery server in the closest cluster to which the user will remains connected also if he will switch device or use another browser, as long the session will not expire. During the session, information concerning the user, like interactions, used device and visited pages, are retrieved and stored in either a private or a shared session state until the session

will not end. At the end of the session these information are pushed on the Collection database and scheduled for the processing done by the Processing server. This aggregates customer data in a form suitable to be queried and stores them in the Reporting database, available to the Reporting service. [19]

Integration Layer The Integration layer provides connectivity with third products such as CRM (Customer Relationship Management), custom databases or e-commerce solutions. Thanks to its modular architecture is possible to make Sitecore the hub of external digital activities that have only to push data into Sitecore and use it as central repository for their marketing content, while it will take care of the content management.
In order to make integration available, Sitecore provides several useful tools such as CRM connectors, ERP connectors, Sitecore Commerce Connect, SharePoint connectors and social connections. [27]

Sitecore Launchpad

In order to make easy acceding to Sitecore Experience Platform functionality has been made available the Sitecore Launchpad, a collection of shortcuts to commonly used Sitecore applications.
The Launchpad was introduced with the Sitecore 8 release and provides a unify access to Sitecore functionality including campaign management, view of Analytics reports and performance measurement. It is split into four main areas:

- Marketing Applications: It contains useful applications such as the Marketing Control Panel, from which is possible creating profiles that corresponds to users segments of the website, in addition to goals and events for measuring the engagement, or the Experience Analytics, for seeing interactions of visitors.

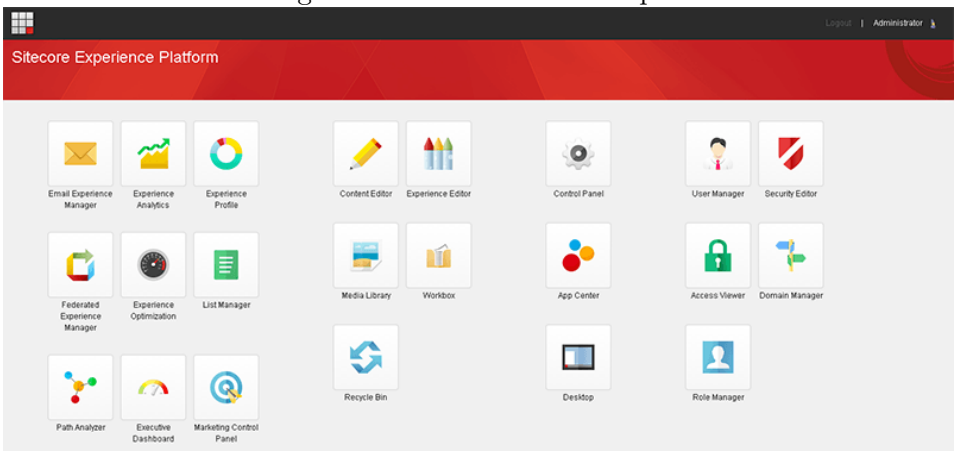
- Content Editing: Applications in this area allow to manage the website content, adding, editing and deleting pages or single item

inside them, in addition to the possibility of categorize each component with respect to users segments.

- Control Panel: It allows to manage system and personal configurations, like a language registration, password change or licenses management.

- Access Management: It concerns the management of the roles and permissions inside the project. Administrator can use these applications to manage the registered users or to modify the permission for acceding the content management.

Figure 1.5. Sitecore Launchpad

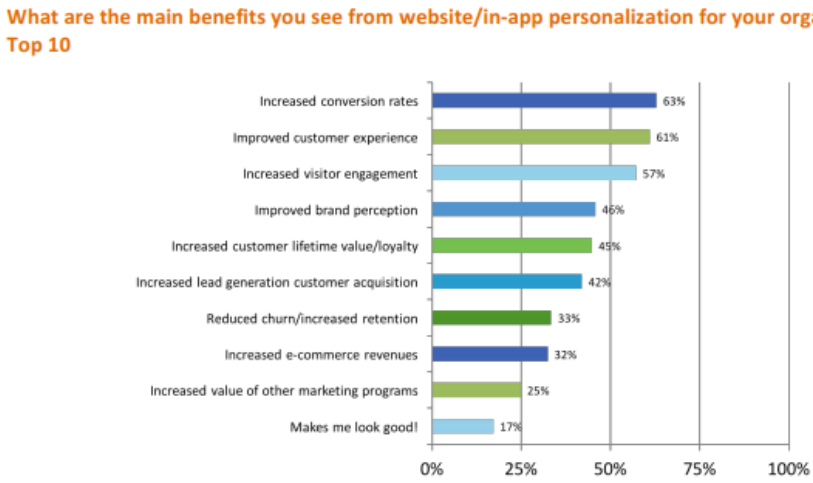


1.1.2 Experience Personalization

The technological growth has considerably increased the consumer expectations, leading them to desire for a personalized treatment in the digital experience, as well as in the real one. Social networks, newsletters, e-commerce sites and several other digital services aim to the customer loyalty providing him personalized content, in order to give the impression to have always the right solution to his needs.

In a survey conducted in 2017 by Evergage and Researchscape International, on 206 organizations of all sizes, about their marketing operations and implementation of personalization, marketers as a whole (88%) say their customers expect an experience, across digital properties, that's personalized to them. The majority of respondents typically see a lift of over 10% from their personalization efforts while 63% indicate increased conversation rates. Moreover, 61% see benefits in the improvement of the customer experience and 57% in the increasing of the visitor engagement.

Figure 1.6. Evergage and Researchscape International survey



Sitecore, with its personalization, provides targeted, relevant content to the users based on their characteristics and behavior, like location, visited pages and interactions with the website. For example you may want to hide some content to a targeted type of user or show additional content to those user that comes from a given country.
Sitecore provides two main ways to enable personalization: Rule-based personalization and Predictive personalization.

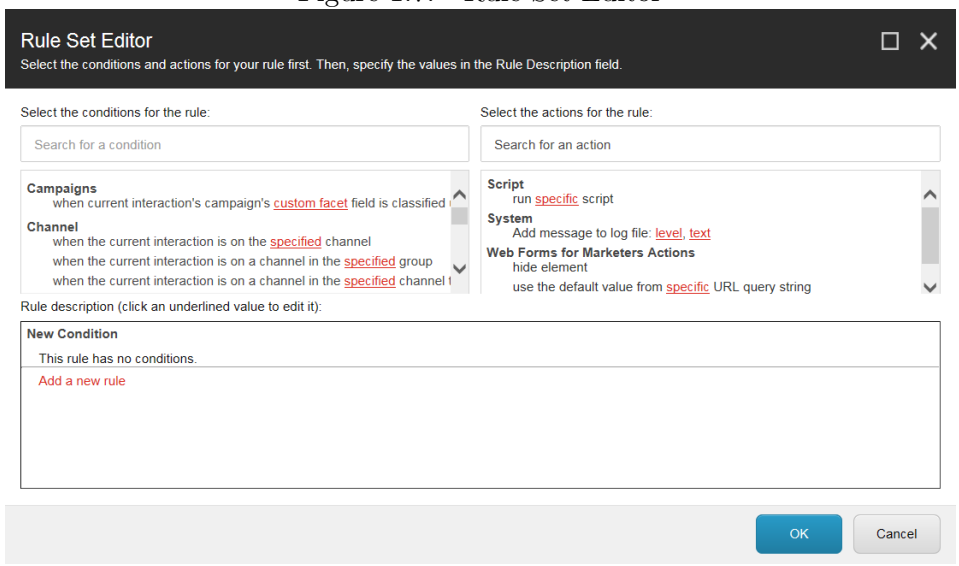
Rule-based personalization The Rule-based personalization allows to modify the rendered content by means of a set of rules formulated by the developer, who can choose between the various suggestions offered by the system or create other personalized ones. These rules are nothing more than if/else statements evaluated in a cascade order from the top to bottom, going through the list until you find a match. They follow the policy of the *first come, first served*, then usually the default rendering is the last of the list, so as to be shown in case no customization is applied. This functionality can be achieved though the Experience Editor tab in the Launchpad. It is a WYSIWYG(what you see is what you get) editor that allows to make changes to the items shown in the page, as well as to their single fields. Once selected the component that has to be modified, is possible to use the Rule Set Editor tool (Figure 1.7) to create conditional renderings and controlling the user's experience. These rules are composed by three main elements:

- Conditions: they are logic statements that determine when a condition is true. For example the **when the current interaction is on the *specified* channel** condition is true, it is triggered if the user interacts with the website through the specified channel.

- Actions: they are logical steps executed when one or more Conditions in a rule are true. For example the **Redirect to *page*** redirects the user to the specified page if the related condition is true. As well as for the conditions, also for the actions Sitecore provides a number of default choices, but the developer can also implements his own.

- Rules: they are associations of one or more conditions with one or more actions, concatenated by means of logical operators as *And* and *Or*. For example if you want to create a rule that hide a component to those user that comes from Canada and Mexico, you have to state a rule where the action hides the component, while the condition contains a double check for the country.

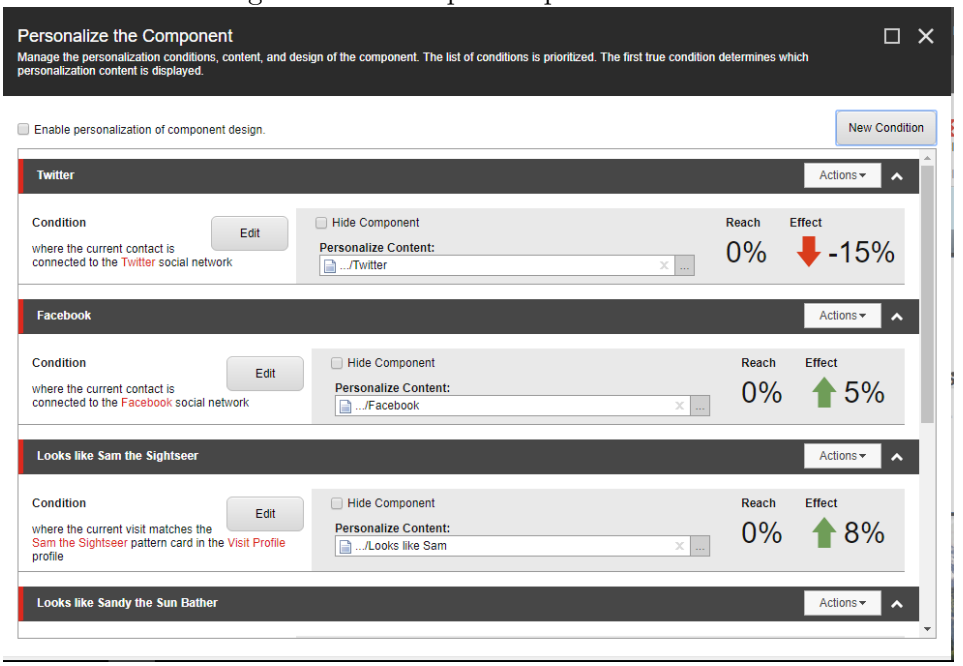
Figure 1.7. Rule Set Editor



Once you have personalized a component you can also check changes obtained by the stated rules, indeed the Experience Editor allows to check for each component inside the page, which rules exist on it and if they are useful or not, giving to the developer the opportunity to change those rules that did not meet expectations (Figure 1.8). For each version of the component, Sitecore provides the *Reach* field, that indicates the percentage of visitors who met the condition, out of all visitors who viewed the component along the time[1], and the *Effect* field, that is calculated as difference, in percent, between the trailing value per visit of the personalized experience and the default experience for the visitors who meet the condition.

[1] if the rule is under test, the time considered by the report is the duration of the test, otherwise it corresponds to the last 30 days.

Figure 1.8. Component personalization



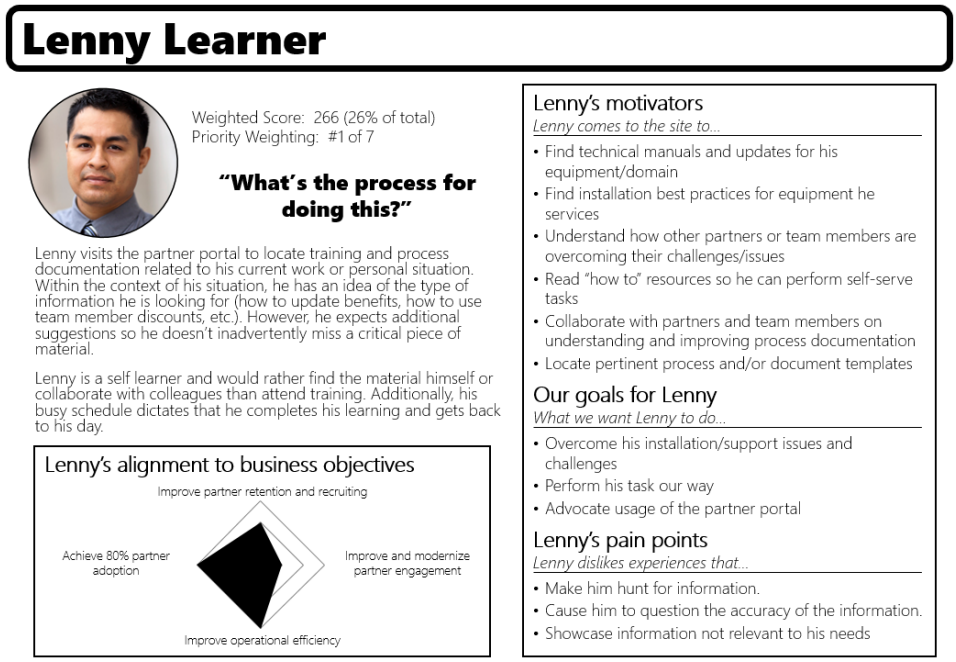
Predictive personalization The Predictive personalization is based on the concept of *Persona* and is the most robust personalization method of Sitecore. This type of personalization uses the information retrieved by the framework about user's interactions within the website, such as visited pages, completed goals and events, in order to categorizing him and finding a match to one of the profiles prepared by marketers, whose features are well known. Assigning one or more *profile cards* also to a component, its easy to implement a rule that hides or shows it to those visitors that match a given profile, avoiding to write complicated rules.

At the base of the personalization concept used by Sitecore, there are several technical terms such as *Persona*, *Profile card*, *Goal* and so on, therefore, in order to clarify what we are talking about and going deeper through the process of personalization, first of all we have to

explain these concepts.

Personas Lars Birkholm Petersen[2] states that «Personas are "archetypical" visitors/users of a website that represent the needs of larger groups of users, in terms of their goals and personal characteristics. They act as "stand-in" for real users and help guiding decisions about functionality and user experience design», since that «they identify user motivations, expectations and goals responsible for driving online behavior»[16]. To define Personas, developers and marketers cooperate

Figure 1.9. Marketing Persona



for developing a fictitious person able to represent a segment of customers. This usually leads to write a full description for the archetype,

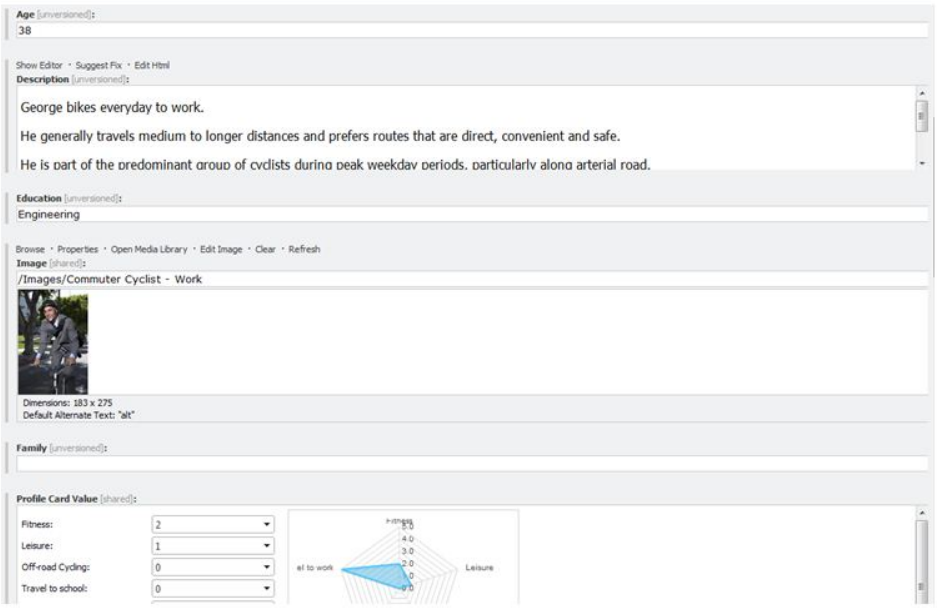
[2]Lars Birkholm Petersen is a web content management expert, leader of Sitecore Business Optimization Services

equipped of name, picture, story and relevant features, such as the reasons why he is using the website and information that might interest him.

Profiles Profiles are just categories of *profile cards* useful to define criteria used to track visitor's behaviors. Thus, for each profile, developer has to define some *profile keys*, whose values will distinguish visitors belonging to it.

Profile cards A *profile card* is akin to a *persona*, but it refers only to the aspects of a single *profile*, while a persona describes life, habits, background and interests in detail. A profile card is a persona whose key/value pairs have been defined for each profile key characterizing proper profile.

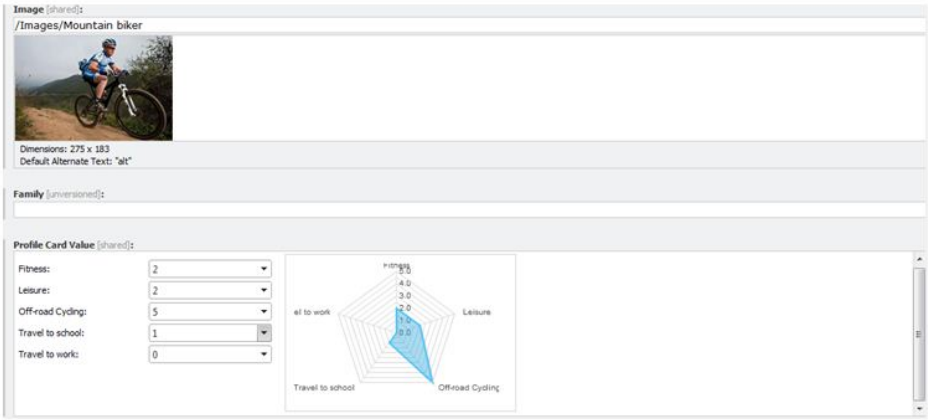
Figure 1.10. Profile Card



Profile keys Profile keys represent the behaviors of a profile card and are associated to numeric values within a defined range. This keys are used by marketers to define relevant parameters on which assign values for categorize profiles.

Pattern cards The pattern card concept is very close to that of the profile card but it concerns the current visitor. Once defined the profile cards, Sitecore allows to set *content profiles*, that are profile cards assigned to the content. In this way is fast and easy to categorize each page of the website associating it to one or more profile cards that represent users segments interested to that type of content. Doing this, when a visitor navigates through the website and reach a profiled content, he accrues the values associated to it. These values contribute profiling the user, indeed Sitecore will assign him the pattern card, whose values are most similar to those accumulated.

Figure 1.11. Pattern Card

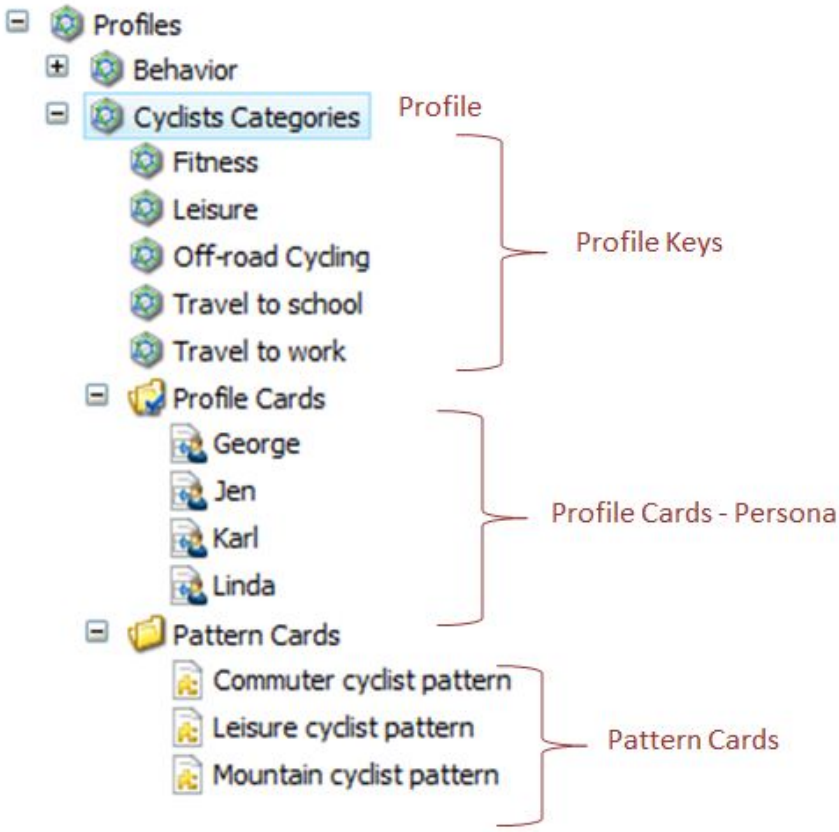


Events Events are like actions that reflect the reasons that led the user to use the website. Looking at Figure 1.9, you can think events like the listed points in the motivation section and so how the digital experience meet the user's needs. Sitecore allows to create and

edit events through the Marketing Control Panel, assigning them engagement value points according to the relative importance. Then, the framework tracks user interactions with the website and the triggered events. Example of events are searches, downloads and registrations.

Goals Goals are similar to events but represent those actions that the brand want the user to do in order to go deeper down the funnel. For example, referring to a e-commerce website, if a possible event is the visit of a particular page, a possible goal is the purchase of a product, or the registration to the newsletter. Given the importance of goals, usually they have a higher value then events, so tracking them, marketers have a relevant measure of user's engagement.

Figure 1.12. Profiles node in Sitecore



User profiling and tracking As visitors start navigating through the website, they are assigned to the content profile values that developers have defined for each visited item. These values are accumulated along the navigation, helping profiling users.
To make this possible, first of all marketers and developers have to collaborate for creating one or more profiles, that will correspond to users segments that interact with the website. Looking at Figure 1.12, that reports an example that can be used within a website of bikes, it might be useful to define a profile for categorize users based on their

use of the bike. The *Cyclists Categories* profile will be composed by profile keys (Fitness, Leisure, Off-roads Cycling, Travel to school and Travel to work) that represent the criteria used for defining different categories of cyclists. After having created the profile keys, developers have to create the profile cards, that will be used to assign values to the content of the website. An example is the "George" profile card, that corresponds to a person that usually uses the bike to go at work and sometimes for fitness, as shown in its diagram in Figure 1.10, and can be assigned to a page of the website that sells products useful to this type of use. When the visitor will navigate to this page he will accumulate the values associated to George. This values will help to categorize the current user and bring him back to the most similar pattern card.
Despite usually pattern cards and profile cards are very similar, with the same names and values, is also possible to define them differently. In the Figures 1.12 and 1.11 have been defined three different pattern cards that don't match exactly the previous profile cards. Among them there is for example the "Mountain Cyclist" card that categorize those users that love go off road, rarely use the bike for going to school and never to go at work.
Along his navigation through the website the visitor accumulates key values associated to the visited items and is categorized with the closest pattern card. This tells to Sitecore the kind of the current user, but is not enough to show personalized content. Indeed, this goal is achieved using the rule-based personalization, that allows to show, hide or change the source of the page content with respect to the current user pattern card.
Together with the profile of the users, Sitecore tracks also their interactions with the website providing detailed reports that can be accessed through the Experience Analytics application inside the Sitecore Experience Platform. Several types of reports are available for helping marketers to understand if the choices made are leading in the desired direction, to check the usage of each channel and how people engaged with the website through the defined goals. Indeed, among all reports available, the most relevant are:

- Online interactions by visits and value per visit: It shows week by week the number of visits and for each visit the value accumulated.

- Channels group by visits: It shows week by week the number of visits for each available channel. This allows to understand who are the preferred channels and so where to put more effort and reach a larger number of users.

- Top pattern matches by value per visit: A pie chart shows which pattern cards are generating the most value. Thanks to this report marketers may decide to get more value out of lower performing personas, updating the personalization strategy.

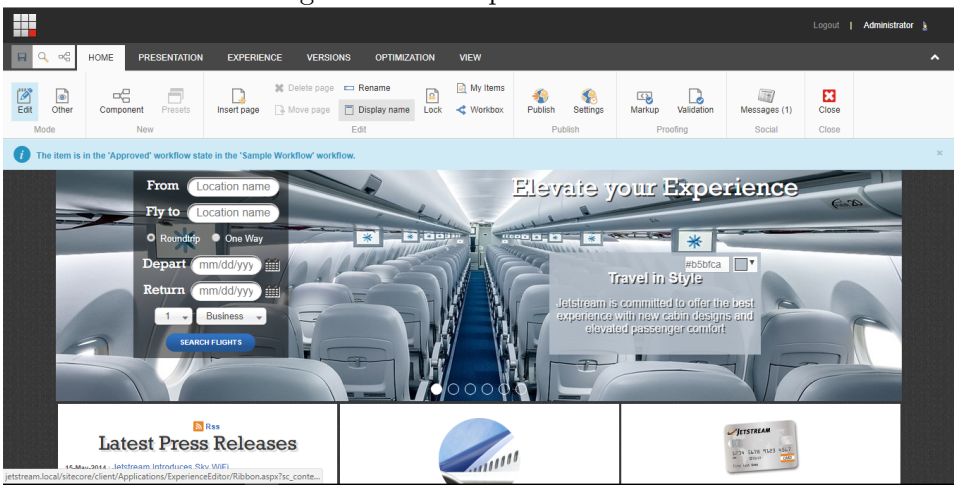
- Top Goals by Conversions: This report shows how goals are engaging the visitors and which are the most used.

1.1.3 Content Management

The content management is the capability of creating, editing or removing content of a website and is a key functionality of Sitecore. The framework provides two tools for editing content, both reachable from the Content Editing section of the Sitecore Launchpad: the Content Editor and the Experience Editor.

Experience Editor The Experience Editor is a WYSIWYG ("what you see is what you get") editor easy to use also for non technical user, such as the marketers. It allows doing modify the content, such as changing an image or editing a text, and seeing immediately the result.

Figure 1.13. Experience Editor



In the Figure 1.13 you can see the several tabs that take part to the ribbon:

- Home: This is the first tab to use to edit the content. You can insert a new page or a single component on an existing page, edit components and then publish the changes to see them on the live site. In addition, the social button allows you to create, edit and publish messages on the desired social network, such as Twitter or Facebook. An important button is the lock/unlock. When you decide to change a component, you must always lock it so that other users can not touch it while you are working on it, and unlock it once the changes are over.

- Presentation: It allows to edit directly the layout of a targeted component, giving to the developer the possibility to choose if applying changes only to the selected object in the current version of the website, or, in case of shared layout, to extending it to all version of the website in all the languages.

- Experience: In this section you can toggle the several available devices and see how the website appear in each one of them. A

very helpful functionality is the one concerning the date field. Sitecore allows to set a publishing future date and hour, so that developers can work at the same time on different versions, without affecting the live site. This is helpful if for example you want a Valentine's day version of the website that is different from the one available all other days. You can set the 14 of February as publishing date for the Valentine's day version and the following day for the main one, and Sitecore will automatically provide the swap operation.

- Versions: From the Version tab developers can manage different versions of the website content, adding new one, removing another or comparing two already existing. In addition they can also change the language of the website.

- Optimization: Here developers can choose among available goals and attributes, such as the content sharing on social networks, the registration to the newsletter or the log-in operation, and associate them to the content of the website. They can also view the report on the applied personalization rules and the effects achieved for the user experience. Finally in the test section is available the list of tests performed on the current page, one list for the suggested tests and another where can be found pages already tested.

- View: It simply provides some facilitation for quickly show or hide controls, the control bar and the navigation bar. In addition it lets the possibility to separate the design phase (component addition and removal, changing of components properties and so on) from the editing one.

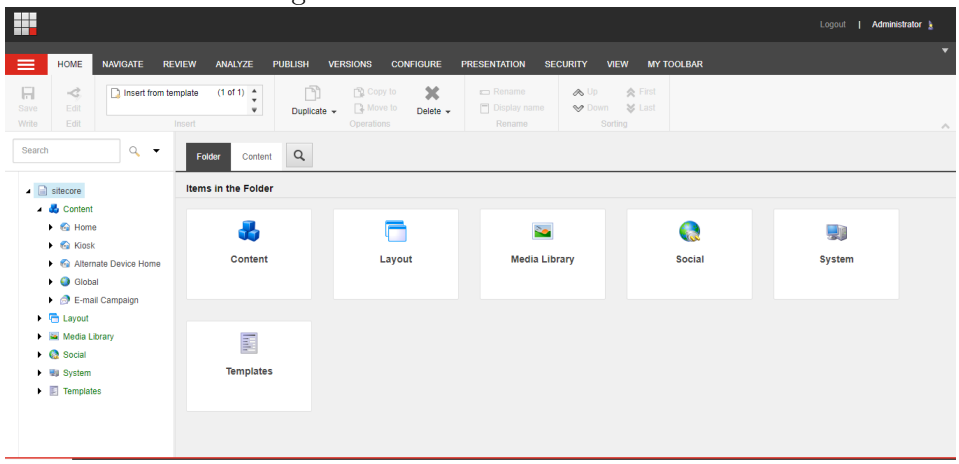
Content Editor The Content Editor is an editing tool designed for users who are familiar with Sitecore. It allows acceding the website content tree, Figure 1.14, whose main components are the following:

- Templates: Everything in Sitecore is an item and every item is based on a template. A template is itself an item that represent the structure and the behavior of other items. Differently from the template concept usually used, in Sitecore it refers not to the presentation of a component, thus the template of a item doesn't tell nothing about how it will be shown. Developers can create their custom templates and inherit from those already available. The data template is used to represent the schema of an item and can be seen like a class whose properties represents the fields of the template. Every field has a type and a standard value, that is its default value, equal in all the items that use the template without override the inner values.

- Layouts: Layouts tell Sitecore where to render a component inside the page and components tell Sitecore how the data template and the content instance should be rendered. A layout contains one or more placeholders, that can be simply thought of as areas in the page where to place components.

- Content: Is the main node of the Content Editor tree and where is located each item of the website. Developer can move within the content node to the desired location and create a new item based on the specified data template in few clicks. Navigating to each single item of the website, is possible to see all the information available about it, such as the used template and the value of the each field associated to it. These fields can also be edited and then republished on the live site. For each item there is also the possibility to see which is the associated profile card and change it with another one. In this way when a visitor, navigating through the website, will arrive on the page concerning this item, he will accumulate the values of the associated profile card.

Figure 1.14. Content Editor tree



Chapter 2

Chatbots and Artificial intelligence

Robots able to recognize the voice of the master, play the role of butlers interfacing with other smart devices around the house in order to adjust the air conditioning, turn on and off the alarm and project films on TV. If in the past all this might seem like pure science fiction, nowadays it is a reality and joins several other robot cases dedicated to ordering online, helping people to shop at the supermarket and even playing the role of babysitter. The artificial intelligence (AI) is increasingly present in everyday life, as evidenced by the debut of a human-like robot in April 2018, for the conduction of a Japanese television news.

One of main goals of the AI is to create a computer able to have conversation with user in a human-like way, so that it can pass the *Imitation game* proposed by Alan Touring[1] in 1950 and that later will be known as the *Touring test*, from the inventor's name. The purpose of the test was to measure the level of intelligence of a computer, by means of an interrogation submitted by a judge to two questioned, a human and

[1] Alan Touring was a British computer scientist lived between 1912 and 1954, widely considered to be the father of artificial intelligence.

a machine, using a text type interface similar to a modern messaging application. The judge communicated with each participant from separated rooms and without knowing who was the human and who was the machine. His task was to submit to each respondent a set of answers trying to identify the machine.
Chatbots (also known as *virtual assistant*, *online chat program* or *software agent*) are software applications aimed to interact with users in a conversational way by means of text messages. Their presence on the market has seen an exponential increase in the last years but the first attempts to realize such technology go back to a distant past considering the technological evolution

2.1 History of chatbots

The first chatbot was developed at MIT by Joseph Weizenbaum and shown to the public in 1966. It was a program written in MAD-Slip for the IBM 7094, called ELIZA, thought to interact with humans by means of a text messages. ELIZA simulated conversations based on provided scripts, including the one concerning a Rogerian psychotherapist and that made the machine famous. User interacted with the bot typed in some statements using normal punctuation and sentence structure (except the question mark, who could not be used for MAC system compatibility), than ELIZA analyzed them searching for keyword and printed out a statement retrieved from an internal list and modified using some transformation rules related to the found keyword, without really understanding the conversation[25]. An example of rule, based on [10], could be resumed as the following:

keyword: i
  pattern: * i am *
    answer: Is it because you are (2) that you came to me?
  pattern: * i don't *
    answer: Why don't you (2)?
  pattern: *
    answer: Can you elaborate on that?

When the bot finds the keyword, it tries to match the received statement with the related patterns in the script (* represents wildcards that always match) and, in case of success, returns the associated text in which (2) represents the second wildcard, otherwise returns the default answer.
Despite the purpose of Weizenbaum was not to simulate the human cognitive process many patients thought to speak with a real person. Given its success, ELIZA has been at the base of many other later chatbots, including PARRY, developed by Kenneth Colby in 1971 with the meant of simulating a paranoid schizophrenic patient. Like the previous, it is rule-based and has similar structure, but it is also able to perform "emotional responses" triggered by a weight applied to the inputs. PARRY was also the first bot to pass the Turing test, in a modified version where the jury was composed by psychotherapists whose aim was to discover if the patient was a human or a machine.
An interesting originality was introduced in 1997 with Jabberwacky, an artificial intelligence created by Rollo Carpenter, aimed to simulate natural human chat with the capability to *learn* as the humans. Differently from the previous it was not bound to hard-coded rules but, based on an internal knowledge-base containing a great amount of conversations and a pattern-matching algorithm, it was able to formulate an answer. As already said Jabberwacky learns. This is possible because the bot stores each conversation, in each language, so that later time will know something more[5].
Later, from the hands of the same creator was born Cleverbot, an advanced version of Jabberwacky. As the previous, it learns from the people who chat with, storing each conversation in a huge database which is scanned each time a new response has to be provided in a conversation. The capability to provide a response searching among all the conversations done (from its launch to 2011 Cleverbot has exchanged 65 million conversations), and so having a huge amount of human statements as source of inspiration, is the key of its success in the Turing test done in 2011, in which 59% of its human interlocutors thought to converse with another human[28].

Moving close to nowadays, in 1995 Wallace developed ALICE (Artificial Linguistic Computer Entity), become later, in 2002, an open-source chatbot. It clearly separate the *bot engine* from the *language knowledge model*, so that is possible to change one without affecting the other.
ALICE doesn't have any semantics or NPL modules but works with a huge set of rules, maintained in a AIML file[2], containing AIML objects composed by topics and categories. A topic is a top-level element, which internally has a name attribute and a set of categories related to it. Each category, the basic unit of knowledge, is a rule composed by the couple pattern/template that represents user input and related chatbot template for the answer. When the bot receives an user input, it searches for the longest pattern matching using a depth first technique in order to retrieve the template for the response.
ALICE relies on more then fifty thousand categories and can count several Loebner[3] competitions won.
With the arrival of twenty-first century, research for creating a chatbot able to replicate a human-like conversation proceeded with the development of artificial intelligence and machine learning algorithms. However, instead of chatbots with the purpose of maintain general conversations with humans, was preferred to invest in machines focused on specific purposes, able to cover the virtual assistants role and using data sources for answering questions[2].
The first example of this new trend was SmarterChild, a chatbot developed by ActiveBuddy in 2001 with the aim of providing the user with useful information such as sport scores, movie quotes, weather forecasts and so on. It lived inside AIM (AOL Instant Messenger)

[2]AIML, or Artificial Intelligence Mark-up Language, is a XML derivation developed by the Alicebot community during 1995-2000, that enables people to provide input patterns into the chatbot.

[3]The Loebner Prize is an annual competition in artificial intelligence that awards prize for the most human-like computer program based on a simplified Turing test.

so that users could simply communicating with it by means of textual messages, from which the bot, relying on its linguistic artificial intelligence, was able to extract the context and the type of response to deliver. Smarterchild relied on a huge database, which allowed to query and obtain the desired information in a shorter time than in any other website. In addition it was also one of first chatbots to deliver targeted content to the user, indeed it asked for name, age range and zip code in order to contextualize the user and delivering him the right information about weather, news and so on [14].
An example of conversation with this chatbot could be the following:

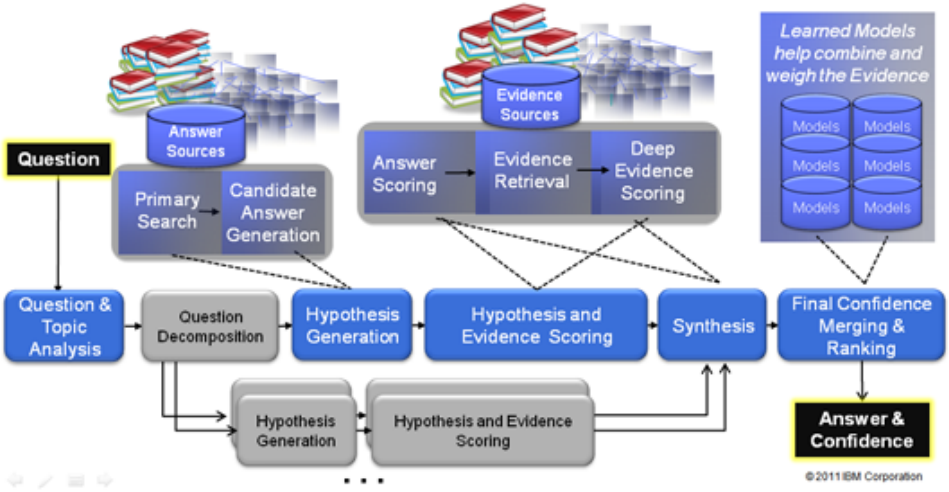
User: What movies are playing?
SmarterChild: For what city or zip code would you like the movie listings?
User: 10036
SmarterChild: Movies playing in or near New York, NY (10036) on Tuesday, October 23rd:

1. The Game Plan [PG]
2. 30 Days of Night [R]
3. The Comebacks [PG13]
4. The Heartbreak Kid [R]
5. Michael Clayton [R]
6. We Own the Night [R]

SmarterChild achieved a great success, accounting for 5% of global instant messaging traffic and laying the foundations of modern virtual assistants, such as Siri and Alexa, but the user's involvement was insufficient to guarantee long-term success. This, combined with the scalability problems encountered with Neuro-Linguistic programming, led to the dismissal when in 2007 Microsoft acquired the company ActiveBuddy[4].

Another example of development in this direction was IBM Watson, a question answering system born in 2006 with the purpose of taking part to a television quiz called Jeopardy!, in which participants have to answer to general questions. Technically, IBM Watson is not really a chatbot but, in a refashioned version, it provides now a set of intelligent services used in chatbot development. Indeed, the aim of the Watson project was not only about playing Jeopardy!, but mostly doing research in natural language understanding and machine learning. The IBM Watson used for the game was developed on the DeepQA software architecture, that allows advanced natural language processing, informational retrieval, reasoning based on evidences and machine learning (an high-level view of the architecture is reported in Figure 2.1). The first processing task is to analyze the given question using

Figure 2.1. DeepQA Architecture



natural language processing techniques in order to extract relevant information, such as the entities involved in the phrase, the requested response type and so on. Then the process continues with the research of information sources where could be found the desired response. The research is done both on structured data like databases that on unstructured data like encyclopedias, dictionaries, news articles and other

textual material coming from the web. Once retrieved the source documents, several possible answers are collected using information extraction algorithms that are able to find relevant terms and entities inside them. In order to select only the right answer between all the possibilities, various analytics strategies and scoring algorithms are used to create evidence on which draw up a ranking to the retrieved possibilities. The ranking strategies can exploit information like the expected category of the answer, geographical reasoning or the *lexical overlap*[4]. Finally, thanks to machine learning techniques that use historical collected answers and the aggregation of all scores calculated with the various strategies, the final rank is computed and top scored answer extracted[9].
The effort put into developing this technology was rewarded in 2011, when IBM Watson challenged the best champions of the game and won. After obtaining the desired success, the company decided to open Watson technology to the rest of the world, developing services able to use it in the most diverse areas, such as medicine, technical support and industry.
In the wake of the success obtained by SmarterChild few years before, in 2011 Apple launched Siri, an intelligent assistant consisting of machine learning, natural language processing and Web search algorithms. The purpose of this software program is to help users doing tasks such as making calls, writing and reading messages and creating remainders, simply by requesting them verbally.
Siri can be split in three different layers: voice processing, grammar analysis-context learning engine and services.
First of all when the user gives a command, the device used collects analog voice and translates it to a file audio. This task is not simple because of Siri has to interfacing with people peaking different languages,

[4]The lexical overlap technique takes in account how many keywords are in common between the question and the supporting evidence for each candidate answer

in different accents, thus every interaction with Siri is recorded by Apple and used to improve the later translations. Then the speech is sent to the Apple servers, where is translated to text and natural language processing techniques are used to understand the meaning of it, by looking at the syntactical structure. This step allows to extract the core of the request, independently of how it is formulated: thus phrases like "I want a pizza" or "I would like to eat a pizza" will be evaluated in the same way. Finally, once Siri knows what is the user intent, it has to interact with the application chosen to accomplish the desired task.
With the advancement of voice technology and following the Watson and Siri example, in last years several other companies launched their own voice-based personal assistants, such as Cortana for Microsoft (2013), Alexa for Amazon (2014) and Google Assistant for Google (2016). On the other hand, starting from Telegram in 2015, several messaging platform such as Slack, Skype and Facebook Messenger opened to chatbots allowing developer to deploy their bot on them, leveraging the possibility to reach a huge consumer base to which providing multiple services via chat.

2.2 How the chatbots work

Starting from the Alan Turing and the first chatbot ELIZA, until nowadays the research in the human-machine conversation has made significant progress, with the development of several bot able to interact with users in a conversational way by means of text-messages or voice and in some cases also taking part to public challenges with human participants and win them.
In literature the term chatbot has been associated to several definitions and synonymous, such as chatterbot, conversational agents, software agents, virtual agents, intelligent personal assistants. Together with the different definitions, in order to siting them in categories, were born also various classification criteria, such as how to interact with the user (by means of messages or voice), the purpose for which they

were created (carry out targeted activities or support dialogues with humans), rather than the technique used to respond (based on hand-written rules or using natural languages components).
Bots has become increasingly smart, able to answer a question logically and solve requested tasks, however at the same time not all modern bots can be considered smart with respect to the Turing concept of intelligence, that concerns the capability of a machine to maintain a conversation with a human without he realize that is talking to a robot. The needs of the market have in fact led to develop chatbots able to understand the user's needs and carry out simple tasks in his place, rather than maintaining a conversation with the sole purpose of entertaining. Thus, a first distinction is between task-oriented chatbots and non-task-oriented chatbots, then in both is possible to made an additional discrimination between rules-based and intelligent chatbots.

Task-oriented chatbots

This category includes those chatbots based on a *domain ontology*, a knowledge structure representing the kinds of intentions the system can extract from user sentences[11]. Belong to it, those personal agents available on mobile devices such as Siri for Apple, Cortana for Microsoft and Alexa for Amazon.
Simplest bots carry out only the tasks belonging to a target domain and discard all the others, like booking a fly or an hotel, while advanced ones are able to recognize and accomplish intentions concerning different domains, like the personal assistants on the smartphone that let you set a reminder, make a call or search something in the web in addition to several other operations.
Each domain ontology defines one or more *intents*, each one composed by a set of *slots*, and the values that each slot can take. The set of slots defines what the system need to known in order to accomplish its task. For example the travel domain can include a Booking-flight intent for representing the user intention to book a flight ticket. In order to satisfy the request of the user the system can define the following slots with their relative value types, where City type means that the

slot requires the name of a city, while Date type could be a hierarchic type including integers for day, month and year.

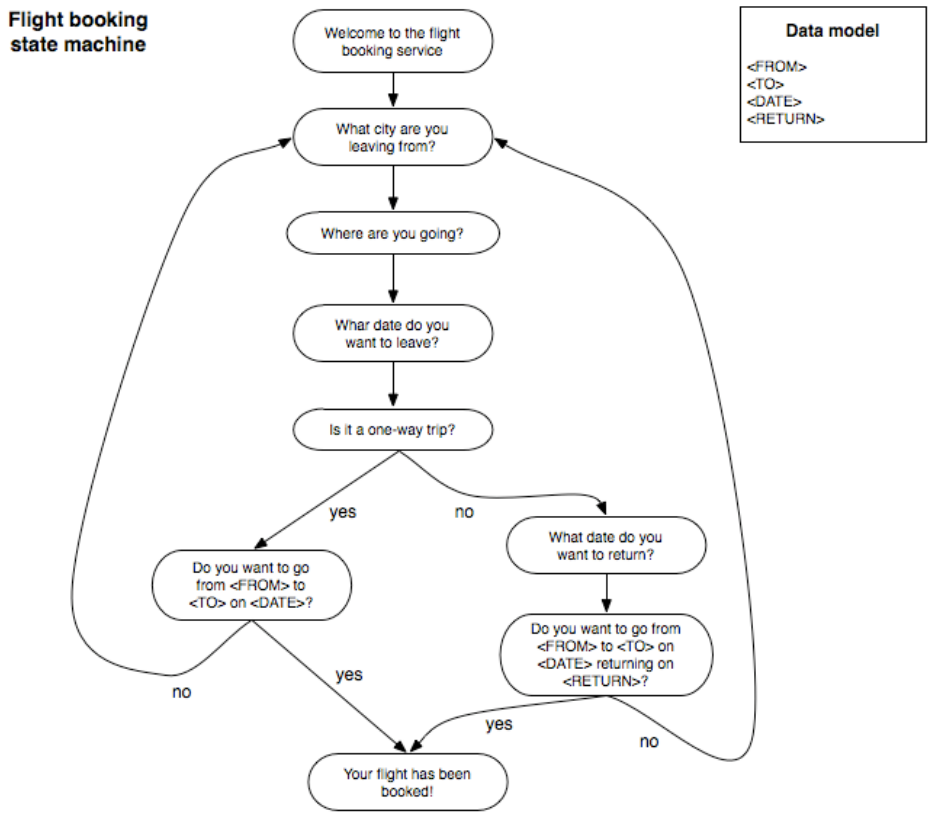
Slot	**Type**
Departure city	City
Destination city	City
Departure date	Date
Arrival date	Date

The way of handling the dialogue, formulating questions and retrieving information from the user utterance, in addition to providing response, depends from the three main components of the chatbot: the Dialog Manager, the Natural Language Understanding (NLU) component, the Output Generator.

Dialog Manager The Dialog Manager component is the one who manages all the aspects of the dialog and coordinates activities of all other components, thus is the core of the dialog systems. It has to recognize if all necessary information have been retrieved from the user, or if something else should be asked, interacts with external knowledge databases for searching and providing query results and internally, control the conversational mechanisms such as multi-party dialog and back-channels.
There are three main models to choose in the Dialog Manager implementation phase concerning the way used by the system to converse with the users[1].

- Finite-state model: This is the simplest model. Here the Dialog Manager is represented as a finite-state machine, where each state corresponds at a part of the conversation and the arcs linking the states are the streets that the conversation can follow. Thus, along the conversation the user is guided through a set of predefined steps according to his responses.

Figure 2.2. Finite-state machine for task-oriented chatbot



The chatbot proceeds by placing the questions associated with each state of the FSM, following the arcs corresponding to the user's responses, until the task is completed. Internally it stores the dialogue state and the values of the already completed slots. The mainly advantage of this method is that the user is forced to respond in a predictable way, as is the system that has the initiative for the most of time. On the other side, the drawbacks are the rigidity of the conversation, which is less natural, and the need of having a pre-built model according with all the possible street that the dialogue can follow. So, despite its simplicity, this

model is not suitable to open-domain dialogues. An example of a finite-state machine is the one reported in Figure 2.2.

- Frame-based model: It is widely used to compensate the lack of flexibility of the previous model. Indeed in the finite-state model the chatbots asks the questions to the user based on the slots to be filled, one at a time, without possibility to extract more information from the same utterance. The frame-based approach instead uses a frame template including all the information that the system have to retrieve to accomplish a task for the targeted domain. Differently from the finite-state model it allows the user to responds with more degree of freedom, being able to recognize multiple entities in the same utterance. For example, always looking at Figure 2.2, if at the request

  "Where are you going?"

  the user responds

  "I want fly to Rome on Tuesday."

  the bot has to recognize the *Departure city* and the *Departure date* slots, storing them and avoiding to request the departure date again.
  Clearly the advantage of this approach is to have a more natural dialog, with the possibility to get more information in less turns, however these approach is not suitable in complex dialogues and is restricted to those systems aimed to collect information from the user based on static templates.

- Plan-based model: Also known as *Agent-based model*, is a more flexible approach originated from artificial intelligence research and the use of *acts* in order to analyze dialogues in terms of goal to achieve, beliefs and desires of the users.
  A "speech act" is defined as the function of an utterance, such as confirming, requesting, informing or greeting. In order to define speech acts, further the syntactic and semantic analysis of the sentence, also an higher level study is needed, including its

context with the aim of understanding speaker needs and beliefs. In fact the purpose of a sentence can be implicit and not clearly stated. For example the utterance "it's cold today" may not just be an observation, but the desire of turning up the heating or closing the window.
Systems that follow this approach are able to recognize the user intentions also if they are not explicitly exposed and can skip from one topic to another as in a dialog between two humans. An example of dialog of this type, taken from [13], is reported

User: "I'm looking for a job in the Calais area. Are there any servers?"
System: "No, there aren't any employment servers for Calais. However, there is an employment server for Pas-de Calais and an employment sever for Lille. Are you interested in one of these?"

In the example is clear how the system goes further the semantic request formulated by the user, and instead of simply answer "yes" or "no", it provides some additional information to keep alive the dialogue. Although this is quite normal in a conversation between humans, is not obvious when a machine is implied. In fact agent-based models are aware of the dialog context, tracking its evolution and associating it to a purpose dynamically, looking at what the user is saying, what has already said and what information are missing in order to achieve the targeted goal. In this way the system adopts a mixed initiative, where the user is asked for information concerning the retrieved purpose of the dialog, but he can also take the control and introducing a new topic, that will be recognized by the machine, who will adapt to the evolved scenario.
This approach is more flexible than the finite-state or the frame-based models, that are not suitable for complex dialogues. Furthermore is clearly closer to a human-human conversation where the really purpose of some assertions is not explicit but has to be inferred from the context. However, this type of systems requires

more complex technology and advanced natural language understanding capabilities, thus they are not common on the market but are limited to the laboratories for research purposes[13].

Natural Language Understanding (NLU) The role of analyze the sentence for discovering the targeted domain and the relative intent is in charge of the *Natural language understanding* module. The first task of this component is to define the domain of the user request, understanding if he is talking about booking a flight, setting an alarm, searching for an hotel and so on. Then it has to determine the intent, the general goal of the request.
At the current state of Computational linguistic, the input analysis can be done using several techniques, such as Part-of-speech Analysis, Named Entity Recognition, Syntactic and Semantic Parsing.

- Part-of-speech (POS) Analysis: This technique concerns the classification of each word to a part of the speech, such as "noun", "adjective", "verb" and so on. Each word is tagged with respect to the labels contained in the choose tag set, among the several available. The labeling task is done by a POS tagger that has to be trained on some data before, paying attention that, as the tag set used depends from the language of the input utterance, also the training data set to use depends from the future data that the parser will analyze. In fact, taggers that use a stochastic tagging approach relies to the probability of a certain tag occurring, looking at the training corpus used, thus parser trained on data from newspaper article will lose accuracy when used on other data sources. Alternatively to the stochastic approach, the rule-based one can be used, consisting in a set of *if<some pattern> then<label>*.
  An example of POS analysis is the utterance "The grand jury commented on a number of other topics." that is evaluated, using the Peen Treebank POS tag set[5] as

[5]Among the several available, the Peen Treebank POS tag set is the most used.

The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

The POS analysis is easy to use and gives helpful information on the text, however it can suffer ambiguity.

- Named Entity Recognition (NER): This approach is similar to the POS, but instead of tagging each word, it aims to tag only those single words or chunks of words that are relevant for the context, as their recognition is crucial for the intent analysis. An example of this technique applied on the text "John think that Turin is a beautiful city" return the following classification

  [John] (Person) think that [Turin] (Location) is a beautiful city

  The methods used for the Named Entities Recognition (NER) can be divided in three main categories:

  – Hand-made Rule-based NER: This approach use a set of hand-written rules for recognize entities within the text, exploiting grammatical, syntactical and orthographic features such as the part of the speech, the words precedence and the use of capitalization, or using gazetteers automatically learned on annotated corpora containing named entities. This method works well for restricted domains but is not portable and is too domain and language specific.

  – Machine learning-based NER: Is a statistical classification method that use patterns and relationships in the text with statistical models and machine learning algorithms. Based on the machine learning model used, it can be further divided in supervised and unsupervised machine learning. The first means that the program learns to classify the text using a set of labeled data and is guided to label correctly by a "supervisor". Instead the unsupervised model learns,

It includes a list of the available tags and their descriptions.

without any feedback, to autonomously create representations from data. These approach can be used in different domains but need a huge set of data on which learning classification.

  - Hybrid NER: It combines machine leaning and rule-based strongest points, achieving better results that the previous approaches, however is still suffers the domain dependence as the rule-based approaches[12].

- Syntactic and semantic parsing: This approach aims to construct a structural representation of the sentence recognizing structural units of the text at an higher level that part-of-speech or chunks of words because it is applied at the sentence level. The process involves a syntactic parser and a semantic one, relatively employed to extract the utterance constituents as words or chunks of them together with their role (noun, adjective, verb and so on), and the relationship between them.
  The syntactic parser is first of all used to extract the constituents from the input utterance and placing them in a tree structure, labeled with their syntactic role. For example the phrase "The vehicle proceed quickly along the road" is parsed in the following tree structure, where brackets include words dependent on each other

  ```
  (ROOT
      (S
          (NP (DT the) (NN vehicle))
          (VP (VBZ proceeds) (ADVP (RB quickly))
              (PP (IN along) (NP (DT the) (NN road))))))
  ```

  Is possible also that the parser constructs a tree with the whole sentence (S) as root and then divides it in blocks who internally contain words dependency, such as *along the road*, that is grouped. In order to analyze syntactically an utterance, the parser need first of all a grammar to use in its task, then a set

of hand-parsed sentences from which to learn.
Then the semantic parser is employed to understand the meaning of the the input phrase, that is the existing relationships among the phrase constituents according to the semantic framework used[15]. One of most used is the First Order Predicate Calculus (FOPC), based on which the sentence "John is a footballer" would look like

footballer (j)

where "footballer" is a predicate referring to *j*, that is John.
In addition, the framework allows to use special operators such as the quantifiers "∀" and "∃", or the negation "¬".

Output generator Once the Dialog Manager has triggered all the necessary sub-tasks, it may want to provide the retrieved information encapsulating them in a message for the user. Also in this case, the task can be done in several ways, ranging in terms of simplicity and closeness to the formulation that could be made by human being.
The output message, according with the system type can be implemented as a speech, a simple text-based message or including some graphic aimed to rise the user engagement, related on the encapsulated information. This phase is fundamental within the interaction with the user, as the system can be able to understand the intention stated by means of natural language and retrieve all the needed information, but a non-smart way of presenting them, may can lead to a frustration of the work done.
There are three main options for constructing the output message:

- Pre-stored text: this approach is the simplest and concerns the use of canned responses. According with the information to provide, a canned response could be selected among those available, without additional effort, however in this way the responses provided are repetitive and so less natural.

- Template filling: more flexible and personalized that the previous, this method relies on message snippets with variable parts

to fill according with the information retrieved. An example of this approach could involve the use of the user name, if available, together with the information requested, in order to make the message more personal. Despite the increased flexibility, also in this case the message could seems repetitive.

- Natural Language generation: this is the most advanced approach, able to achieve the best result. It concerns the output planning at an abstract level, then processed by a dialog pipeline similar to the one used by the Natural Language Understanding component, in order to mimic human responses more naturally, removing dependencies from rule-based approaches. Several architectures has been defined to accomplish this tasks, however they all refers to the same sub-tasks, implemented in different ways along the pipeline. Basically they can be resumed as

  - Content determination: deciding what information to include and what not in the response. The included content strongly depends from the domain in which the system is operating.
  - Text structuring: Once the system has decided what information to provide, then it needs to order them so as to expose them in a sensible way. Indeed, is quite obvious that, looking for example at news in newspaper article, if some sentences are extracted and shuffled randomly that lost sense.
  - Aggregation: after the text structuring stage, the system has a list of information expressed in separated sentences. At this point is necessary to reorganize them by combining multiple messages in a single one, in order to achieve a more fluid output.
  - Lexicalisation: it concerns with choosing to right word or phrase in order to correctly express the retrieved information. This step, therefore aims to transform the message's

building blocks into natural language representation, being aware of the context and of the possible several ways to expose the same concept.

- Referring Expression Generation: this step is similar to the previous but is aimed to determine how referring to entities in the text avoiding repetitions, by using pronouns, short defined nouns or other attributes.
- Linguistic Realisation: in this step, all the words and phrases decided before have to be combined to form a well-formed sentence. It can concern a reordering of the constituents inside the sentence, adding conjunctions, auxiliary verbs and so on. The system has to decide which syntactic form to use, exploiting human-crafted templates or statistical approaches, ensuring the correctness of the resulting text, both syntactically and morphologically[8].

Non-task-oriented chatbots

Non-task-oriented chatbots, also known simply as chatbots, are those bots not thought to accomplish some task on request, rather to carry on extended conversations with users miming human dialogues.
Chatbots can be divided in two categories based on the way they act in order to respond to the user input: rules-based chatbots and corpus-based chatbots. The first group includes earlier bots, who follow a pattern-matching approach, while the second is more recent and is powered by artificial intelligence and machine learning algorithms.

Rules-based chatbots These bots was the earlier to be developed, including ELIZA, the first chatbot shown to the public, created by Weizenbaum at MIT in 1966, and ALICE, its modern version conceived by Wallace in 1995.
As introduced in 2.1, ELIZA worked using rules and a pattern matching algorithm dependent from the script used among the several available, including the one of the Rogerian psychoanalyst, the most known.

Going more in detail, when the chatbot received the user input, it scanned the text from left to right searching for a keyword. The research concerned the use of a rank, calculated looking up each word against a keywords dictionary, so that only the keyword with the higher value was kept, while the others found were discarded. Finally, if a keyword was found, it was tried to associate one of the related rules until one was successful or the search was terminated without any correspondence, in which case it was provided to apply some default rules. At the end, the computed result was printed out to the user.
The structure of the script could being resumed as

(K x ((D_1) ($R_{1,1}$) ($R_{1,2}$) ... (R_{1,m_1}))
(D_2) ($R_{2,1}$) ($R_{2,2}$) ... (R_{2,m_2}))
.
.
.
(D_n) ($R_{n,1}$) ($R_{n,2}$) ... (R_{n,m_n})))

Where K is the keyword with the x value for the rank, while D_i is the ith decomposition rule associated with K and $R_{i,j}$ is the jth reassembly rule for the ith decomposition rule.
So the transformation rules were composed of a decomposition rule and a composition one. For example, as reported in [25], a decomposition rule was

(0 YOU 0 ME)

and the reassembly rule was

(WHAT MAKES YOU THINK I 3 YOU).

In the mentioned rule, associated with the keyword "YOU", the "0" in the decomposition rule stood for an undefined number of words, while the "3" in the reassembly rule indicated that the third component of the decomposed sentence has to be placed in that place. This type of rule could been applied to the sentence "It seems that you hate me", that would be decomposed in *It seems that-you-hate-me* and recomposed as "What makes you think that i hate you".

In addition to simple rules of decomposition and composition, ELIZA was also equipped with substitutions rules designed to transform some words as they were analyzed, and a numerical index for each decomposition rule, updated when the latter was used, and exploited then to access the related replacement rules in a cyclic manner.
A big step forward was done in 1995, when Wallace developed the AIML (Artificial Intelligence Markup Language) and ALICE (Artificial Linguistic Internet Computer Entity), the most famous chatbot based on it.
ALICE could be considered the advanced version of ELIZA, as also in its case a pattern-matching technique is used to respond to the user. It was developed using AIML, a derivative of the XML language used to store the knowledge-base data on which the bot relied. Contrary to ELIZA, ALICE was an open-source project involving nearly five hundred developers, that leading to the bot a "brain" of over forty thousand pattern-template pairs.
AIML consists of AIML objects, that basically are tags, each one containing a command. From all those available, the worth tags are *<category>*, *<pattern>* and *<template>*, organized as explained below. At higher level there is the *topic* tag, that is an optional top-level element that includes one or more *<category>* representing rules, each one made up of a *<pattern>* and a *<template>* used for matching the input and transforming it into the output. More in detail, the basic unit of knowledge in AIML is the category, who is composed of a possible input of the user, called pattern, and a related answer to provide called template.

```
<category>
   <pattern>HELLO</pattern>
   <template>Hi!</template>
</category>
```

However, covering each possible user input is almost impossible, so in order to avoid creating an indefinite number of pairs trying to infer each possible stimulus, developers can exploit some tricks, such as the

possibility to use wildcards, recursive mechanisms and *divide and conquer* approaches, to accelerate and make smarter the covering process. Indeed the AIML vocabulary is composed of simple words, spaces and two special characters, "*" and "_", called wildcards, both used to match single words or multiple-words strings with the only difference that the first has lower priority than the latter. This is useful in all those situations in which similar stimulus lead to similar answers from the bot.
An application example can be the case in which the chatbot wants to show to approve some user statements.

```
<category>
   <pattern>I LIKE *</pattern>
   <template>I like it too!</template>
</category>
```

that will match each input starting with "I like", rather than listing all the possibilities for continuing the sentence. Furthermore, is possible also capture a particular text fragment contained in the user input with a wildcard using the *<star>* tag, and than use it in the response. Taking in account the following example,

```
<category>
   <pattern>I LIKE *</pattern>
   <template>I like <star/> too!</template>
</category>
```

if the user will say "I like football", then the bot will responds "I like football too!".
Another important tag is the *<srai>*, that allows to target different pattern models for a single template tag. It finds application for the use of synonymous, keywords detection and the *divide and conquer* approach. Indeed the content of this tag is a reference to a pattern, whose template will be used for the response. So looking at the previous examples, by adding

```
<category>
   <pattern>GOOD MORNING</pattern>
```

```
    <template><srai>HELLO</srai></template>
</category>
```

if the user types “Good morning”, the response is took from the “HELLO” pattern, so the chatbot will states “Hi!”.
In addition to these ones just presented, several other tags are available and can be combined to create the desired rules.
AIML software then stores all the categories in a tree object similar to the computer file system where, starting from the root, common pattern prefixes are accumulated under the same nodes, allowing to perform a fast back-tracking[24], deep-first search in which the visited path corresponds to an existing pattern that leads to the bot answer.

Corpus-based chatbots The main rule-based chatbots advantage is the behavior predictability, as the back-end engine searches for matching the user input against its knowledge base, retrieving the prepacked answer if the research success, or a default sentence otherwise. The drawback is the need to have a knowledge base able to support the conversation, that has to be periodically updated, requiring a non indifferent effort.
Thanks to the progress in the artificial intelligence science, corpus-based chatbots represent an alternative approach that exploits on the machine learning techniques in order to provide conversations human-machine that mine those human-human.
As the name suggests, these chatbots rely on a huge source (corpus) of conversational data coming from chatting platforms as Twitter. The way in which they use those data for answering to the users allow distinguish them in two model categories: Information retrieval and Generative models.

Information retrieval models The basic idea behind information retrieval chatbots is to respond to an user input with a human response extracted from the corpus data. These approach works well as big is the data set available, because an higher number of conversations

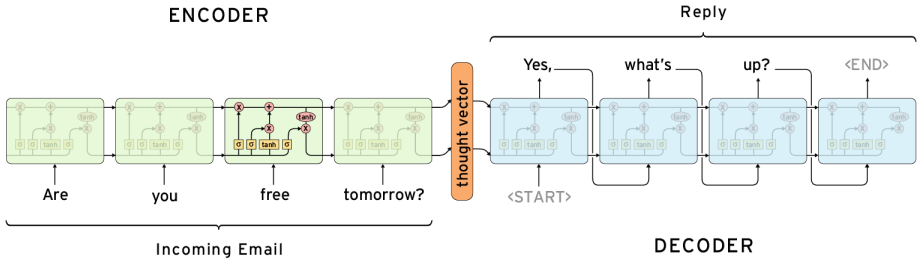
stored implies an higher probability to find the right response, however following this approach chatbots can't generate a new response, they can only provide an existing one.
Several methods can be used in order to search a suitable response but they can summarized in two categories: those searching for most similar question, and those searching for most suitable response.
The former method imply to take the user input and retrieve the most similar question stored in the corpus using whatever evaluation algorithm, such as the cosine similarity, then return the related response. Although this is the most intuitive approach, it doesn't work well as the other mentioned, that is the most used. Its idea is to search within the corpus, the answer that is most similar to the user input, using for example the cosine similarity method as for the previous. This method is less intuitive but relies on the supposition that a good response shares some words with the question.
Having the user input, searching for the most similar question in the corpus doesn't ensure that the response is appropriate, instead searching for the most similar answer is more effective but doesn't consider those available good responses that don't share words with the input[11]. In order to improve this techniques, can be done a research that, in addition to the selected turn[6], considers also the previous ones and the user context.

Generative models While information retrieval models answer to user inputs searching the most suitable response within a set of human conversations, generative models use these data for training and then generate response by themselves.
The initial approach for accomplish this task was proposed in 2011 by Ritter, Cherry and Dolan[17], who present a data driven approach that exploited phrase based Statistical Machine Translation (SMT),

[6] A turn is a sentence of arbitrary length, including from one to many words, representing an interaction inside a dialogue. Speakers' turns alternation composes the dialogue.

for mapping user input to a system response. Indeed, relying on the strong relationship that often occurs between two sequential conversation turns, they realized a way for constructing a response, by analyzing the user sentence.
This method outperformed the information retrieval approach and in 15% of cases automatic generated responses was preferred to human ones, however it had to cope with some inconvenient. First of all the system tended to generate answers very similar to the input, because of the language similarity between initiator and respondent. These was solved by filtering those generated sentences that were too similar to the input. Then, a second problem was the word alignment that occurred in the long phrases and that was partially resolved using the Giza++ alignment tool created by Och and Ney.
Starting from the Statistical Machine Translation method, in 2014 Kyunghyun Cho introduced the sequence to sequence (seq2seq) model for response generation. This model is based on two recurrent neural networks (RNN) encoder-decoder architecture. The first RNN is an encoder that takes in input an arbitrary-length sentence and encodes it in a fixed-length vector of symbols, who is then taken by the second RNN, which is a decoder, and decoded into another symbol sequence of arbitrary length[6].
At each step the encoder transforms a word into a symbol and update an internal state that influences later translation within the sentence. At the end the encoder produce a final hidden state, also called context, that is the input of the decoder, which at each step generates a new word, influenced by the context itself and the output of the preceding step. This model allows to provide good responses, however it has a tendency to produce repetitive answers and it is not able to handle the overall conversation context. Recent studies, have shown how to deal with these issues modifying the decoder and using reinforcement learning [11].

Figure 2.3. Sequence to sequence model for answers generation



Chapter 3

Microsoft Bot Framework

The increasing importance and diffusion of chatbots has led to the born of several frameworks aimed to guide developers to create and managing their own bots by means of dedicated services and tools. Among all those available is worth mentioning IBM Watson, Amazon Lex, Chatfuel.
In addition to the mentioned ones, a leading role is covered by the Microsoft Bot Framework. It is a complete suite for building intelligent and powerful bots, deploying and testing them, composed of a Bot Connector service, a developer portal and a Bot builder SDK. Combined with Azure Bot Service, it allows to simplify and speed up the chatbots development in a purpose-built environment.

3.1 Azure Bot Service

Microsoft Azure, also known simply as Azure, is the cloud computing service offered by Microsoft in order to help developers to build, deploy and manage their applications without worrying about managing necessary hardware and computing resources.
Traditionally companies have their own infrastructure, including web servers and all hardware necessary to provide their purposes. However this requires to have qualified personnel to estimate the required

hardware, purchasing it and then maintaining over the time, adding resources when necessary and providing an internet performance adequate to support the estimated network traffic. Cloud computing has revolutionized everything. Indeed it allows organizations to access to a massive pool of computing resources for fee, without need to purchase their own hardware and maintaining it, and using software services on demand.
Azure is part of those cloud computing platforms classified as *Platforms as a Service (PaaS)* for the way its cloud resources are available to end users. That is, it provides both infrastructure and run-time environment to deploy applications without letting to the user the possibility to control the infrastructure, so without worry about managing web servers or database servers with maintaining and updating operations. Thanks to this architecture, developers have only to provide their application and the PaaS vendor will provide the means for deploy and run them.
An important service offered by Azure is the *Azure Bot Service*, an integrated environment purpose-built for bot development that enables to create, deploy, test and manage bots. Following a guided procedure, shown in Figure 3.1 developers can choose one of the two available language to use (C# and Node.js) and then select a template, among the five provided, on which to base the chatbot:

- Basic bot: this is the bot template with basic functionality, able to respond to users input simply back to the users the same words they have typed in.

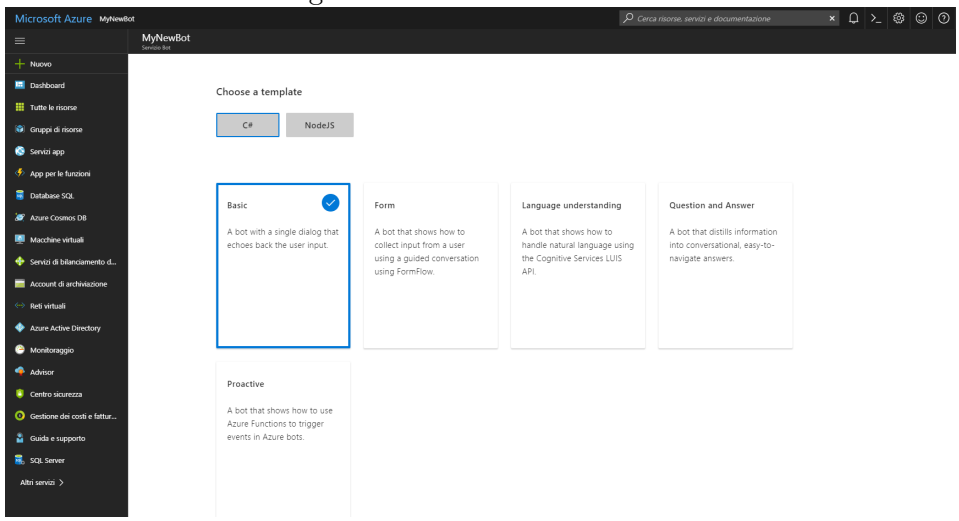
- Form bot: is a template specifically designed to build a guided conversation with the user, in order to retrieve some information useful for filling a defined form.

- Language understanding bot: this is a bot that uses language understanding models to retrieve the user sentence meaning in a natural language conversation. Leveraging on LUIS (Language Understanding Intelligent Service), the bot is able to recognize intents and entities in the user utterance and behave accordingly.

- Question answer bot: this template allows developers to create bots able to respond to repetitive questions present in its own knowledge database. It leverage on the QnA Maker service able to parse questions and providing desired answers.

- Proactive bot: usually bots react to user input and behave accordingly to its latest message. Instead the proactive bot is able to send messages not directly related to user recently messages and not belonging to the conversation context. An example of this type is a bot able to send a message to the user for a reminder.

Figure 3.1. Azure Bot Service



Azure Bot Service speeds up the bot development leveraging on provided components such as the Bot Builder and the Microsoft Bot Framework connectors in order to make available the bot on several defined channels.

3.2 Bot Framework components

The Azure Bot Framework provides a set of means useful to simplify the task of the developer in managing, testing and publishing his own chatbot. It can be divided in three main components: the Bot Builder SDK, the Bot Connector Service and the Bot Developer Portal.

3.2.1 Bot Builder SDK

The Bot Builder SDK is an open source framework available for both Node.js and .NET, with features that make the interaction between the bot and the user simpler and a emulator for debugging the bot once created. For .NET it is available as NuGet package and on GitHub, so it is quickly retrievable from the Visual Studio IDE.
Bots can communicate with users by means of simple text based messages, rather that rich text messages including text, image and other components such as buttons. However, in any case developers have always to face with common problems like managing Input/Output (I/O) operations, connecting the bot to the user and providing language and dialogue skills for support the conversation. All these features are allowed by means of the core components of this SDK, that are listed below.

- Channel: is the connection between the Bot Framework and the bot application. Developer has to configure the bot to connect to the channels he wants it to be available on. Skype and Web Chat are pre-configured, but many popular services such as Bing, Cortana, Direct Line and several others are available. For example, a bot connected to the Skype channel can be added to a contact list and people can interact with it in Skype.

- Connector: is a library that allows to access the Connector Service for delivering messages from bot to channel and from channel to bot.

- Activity: the connector, in order to exchange information back

and forth between the bot and the user, uses activity objects that are essentially the body of the HTTP request sent. Different types of activity are allowed, like the typing activity, the ping or the endOfConversation, but the most used is the message. This one is used to exchange information with the user and can be simple plain text, or may can have richer content such as media attachments, buttons, and cards or channel-specific data.

- Dialog: when a bot is created using the Bot Builder SDK for .NET, it can use dialogs to model a conversation and manage conversation flow. A dialog can be composed of other dialogs to maximize reuse, and a dialog context maintains the stack of dialogs that are active in the conversation at any point in time. A conversation that comprises dialogs is portable across computers, making the bot implementation able to scale. The conversation state (the dialog stack and the state of each dialog in the stack) is automatically stored, enabling bot's service code to be stateless, much like a web application that does not need to store session state in web server memory[23].

- State: the Bot Builder allows the bot to store data associated to a user and reuse them for enabling preferences and customize next conversations. To store state data a developer can either use the Bot Builder Framework's in-memory data storage, or custom data storage like Cosmos DB or Azure Table storage.

- FormFlow: Within the Bot Builder SDK for .NET FormFlow can be used for building a bot that collects information from the user. For example, a bot that takes sandwich orders must collect several pieces of information from the user such as type of bread, choice of toppings, size, and so on. Given basic guidelines, FormFlow can automatically generate the dialogs necessary to manage a guided conversation. It is less flexible that Dialogs but it greatly simplify the process of managing a guided conversation. To create a bot using FormFlow, developer must specify the information that the bot needs to collect from the user. To

do this you can define a form by using either a C# class or JSON schema.

3.2.2 Bot Connector Service

The Bot Connector Service enables the bot to communicate through the channels defined in the Bot Framework Portal, using REST and JSON over HTTPS. Communication requires an access token obtained providing the Azure App ID and password, indeed a bot communicates with the Bot Connector service using HTTP over a secured channel (SSL/TLS). When it sends a request to the Connector service, it must include information that the Connector Service can use to verify its identity. Likewise, when the Connector Service sends a request to the bot, it must include information that the bot can use to verify its identity.

3.2.3 Bot Developer Portal

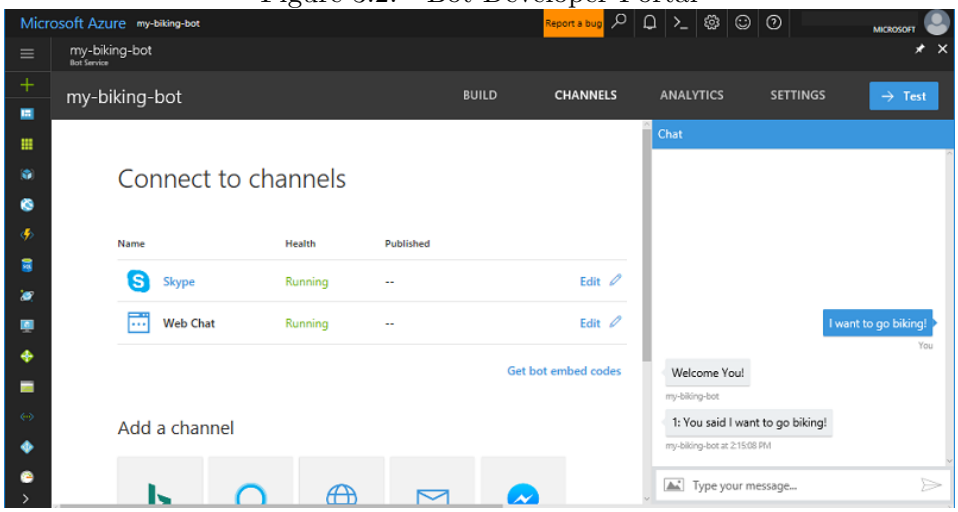
The Bot Developer Portal, accessible at https://dev.botframework.com/, is a central place used by developers for managing, testing and deploying their chatbots. After logging into the portal, you can access your bots and for each one of them exploit the offered service, reported in Figure 3.2, whose sections are listed below.

- Build: is where is possible to download the zip file containing the source code, in order to develop it locally with the favorite IDE. It is also available an online code editor that can be used to make quick changes on the code and see them applied instantly.
- Channels: is where is possible to configure the bot to connect to the channels you want it to be available on. In this section channels can be managed by adding, editing or deleting theme and is easy for the developer to see associated configuration parameters.
- Analytics: is an extension of Application Insights, an extensible Application Performance Management (APM) service for

web developers that can use it to monitor their live web application. While Application Insights provide service-level and instrumentation data like traffic and latency, Analytics provides conversation-level reporting on user, message and channel data[22]. If you are not interested to the wall data report but only to a part, you can filter it on the channels and time period of interest.

- Settings: in this blade the developer can access and modify bot settings such as Display name, Icon and Application Insights.
- Test: the Bot service provides a Web Chat control to help developers to test their bot. In this way is simple to see and improve behaviors of the bot without need of any other external debug tool.

Figure 3.2. Bot Developer Portal



Chapter 4

Project Implementation

The purpose of the project is the development of a chatbot to integrate into a Sitecore website, with the aim of increasing the user engagement and his navigation experience. Today's customers search for more than the traditional one-side commerce communication services. They want to connect and engage with their brands in order to build a personalized experience, that is user-context aware. Sitecore features allow tracking user activities on the website and manipulate them to understand more on who use the service, in order to provide him the right content with respect to he his looking for, but a relevant improvement can be achieved by letting the user actively interact, and making him feel part of the service. This is the reason why a chatbot could be a huge opportunity of increasing the user experience within the website.

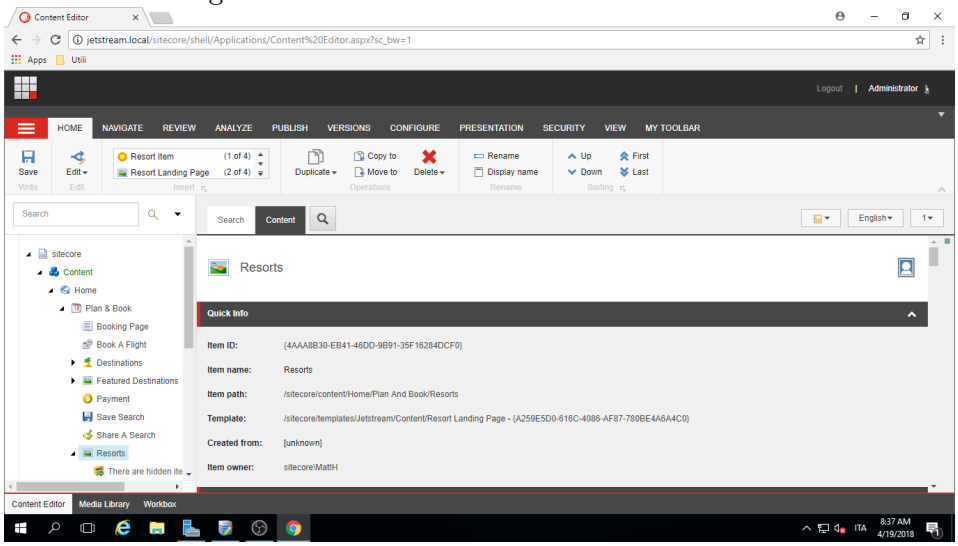
4.1 Jetstream website integration

For the project purpose has been chosen a Sitecore website dedicated to travels, named Jetstream. This choice has been guided by the fact that travelers can easily be divided in categories based on the types of destinations they are interested in, for example a user who focuses his research on exotic places and beautiful beaches is probably a lover of

the sea, while a user who seeks resorts with entertainment and activities for children, will be a family father and so on. This subdivision of users allows to take advantage of the features of Sitecore to provide targeted and dedicated content. Jetstream is a Sitecore's demo website developed with ASP.NET framework that exploit the personalization Sitecore's engine within a travels site. It is not a complete website but is a working solution that lets users to access many services present in other sites, such as navigate through the pages of several destinations, book for a resort or look for available flights in order to reach it.
Behind to the website the Sitecore's engine works for categorize the visitors in order to provide a personalization of the content delivered and achieving a great user experience. As explained in 1.1.2, Sitecore allows to create a set of rules to associate to each item within the website in order to personalize it following the concept of the pattern cards and user profiles. Indeed, together with the set of predefined items which constitute the content of the website, also some standard personas has been defined, equipped of profile cards and pattern cards. In this way when a visitor arrive on a page labeled with a targeted profile card, he automatically accumulates the values associated to it. Then if some personalization rule is triggered, it is applied on the related component, adapting it to the user.
Exploiting the Sitecore Experience Platform and its *Content Editor* section, is possible to see the data stored by default in the Master database and managing it. This section is organized as a treeview, who can be visited trough each folder until to reach leaves that represent items part of the website. Indeed it can be seen as a different representation of the website and so all the sections and pages reachable navigating through the website, are also present in this treeview.
The chatbot purpose is to suggest to the users the most suitable resorts according to their profiles, so has been necessary studying those that are present in the website, their features and how they can be mapped to the users tastes. Resort items are stored in a *Resorts* folder within the Content Editor, as shown in figure 4.1 and each one of theme, is composed by a representative picture, a description and a set of key/-value pairs that helps to categorize it under several aspects.

At the begin of the project the majority of those resorts, apart of the

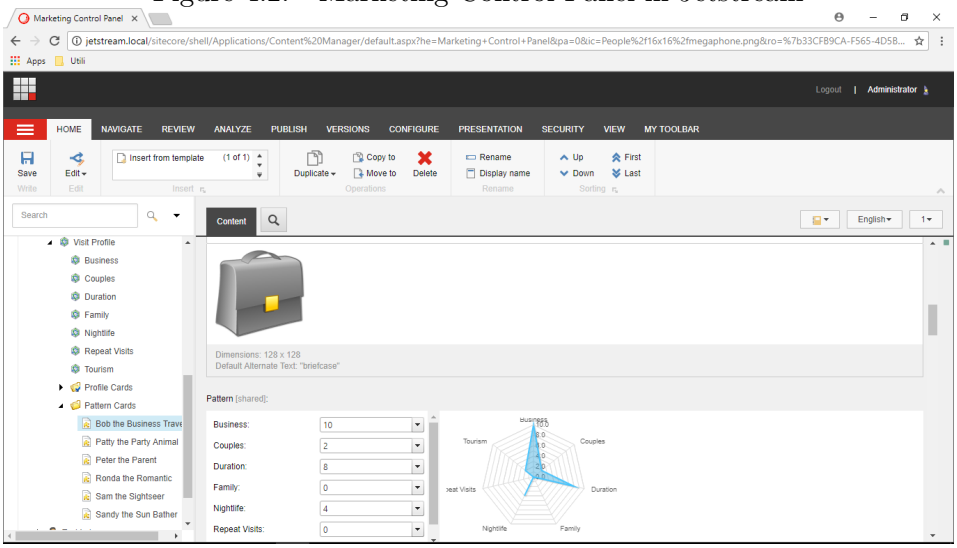
Figure 4.1. Content Editor view in Jetstream



picture and a brief overview, resulting as unclassified because to the default value associated to each key, the zero, so it resulted useless for the personalization process. Therefore has been necessary to navigate through each resort item, and according to the present description , defining the values associated to it, in a range from zero to five.
However the existing key/value pairs were not enough to completely categorize the resorts according to the features of the user profiles, so other fields has been added and set in order to have for each resort a set of values that can be used to check its compatibility with the user behaviors. Acting in this way resorts has been categorized as sea, lake or mountain destinations, for their inclination to host young people as well as families, for the presence of nightlife and the possibility to practice sports, as well as several other parameters.
On the other side, in the Experience platform's *Marketing Control Panel* section, under the *Visit Profile* folder are visible the predefined user profiles and pattern cards, as reported in figure 4.2. There are six

different profiles and the associated card, equipped of a brief description aimed to explain the persona's behaviors and a graph showing the values associated to each key of the profile card, in a range from zero to ten. These profiles are the core of the personalization, as users along their navigation experience accumulate points and are compared to these profiles. For this reason, a study phase acting to fully understand each profile and its possible preferences has been done in order to choose how each one can be considered in a different way from the chatbot though a personalized interaction. In fact, different profiles have different interests and so the chatbot has to aware that a question or a suggestion can be very suitable if posed to one of theme, but useless, if not counter-productive, if posed to one other.

Figure 4.2. Marketing Control Panel in Jetstream



The personalization adopted by Sitecore is useful to increase the user experience, however to make feel him at the center of the site, is necessary to increase his engagement letting him actively interact with it. For this purpose a chat has been embedded within the website, so that to establish a conversation aimed to achieve more information about the user and suggest a destination suitable for him. Using JavaScript,

a web chat control, that is a widget for communicating with the chatbots, has been added into the website exploiting the Direct Line API. In this case the web chat control access the mentioned API using a JavaScript class called DirectLineJS that allows to share the channel with the page of the website in which it is hosted, allowing to sending and receiving massages.
The chat integration is fundamental first of all to avoid redirecting the user to another page dedicated only to the bot, which would lead him to think of it as a separate thing from the site itself, then to allow the data exchange between the Sitecore engine and the chatbot. Indeed this one has to be aware of the profile associated to the user which is interacting both for the destination suggestion that for the conversation. Hence, once a given page is loaded and so the profile score for the current user updated, using the Sitecore Analytics API the user's profile is retrieved together with its related pattern card and sent to the chatbot by means of JavaScript and AJAX, used for calling a purpose-built web service that initialize a new conversation. All the process is made exploiting the communication channel exposed by Direct Line in a way completely invisible to the user, as shown in the following code snippet.

```
[HttpPost]
public void StartConversation(string id, string
   patternProfile)
{
    ConversationID = id;
    PatternProfile = patternProfile;
    DirectLineClient client = new
       DirectLineClient(DirectLineSecret);
    client.Conversations.ReconnectToConversation(Conversati
    var message = "userprofile: "+ PatternProfile;
    ChannelAccount ca = new
       ChannelAccount("userid");
    Activity a = new Activity(text: message,
       fromProperty: ca, type: "message");
    // send message
```

```
    client.Conversations.PostActivity(ConversationID,
        a);
}
```

Once received the current pattern card the chatbot takes the initiative, opens the web chat control and sends a welcome message to the user. This is a not the common approach used by chatbots but has been adopted for emphasizing the bot and encouraging to interact with it. Indeed in the majority of cases is the user that starts interacting, typing some message or enabling it for example with a click on a button that opens the chat, while in the current case is the chatbot that starts the conversation.

4.2 Azure Cosmos DB

All Sitecore website content is stored in the Sitecore databases mentioned above, that can be accessed querying them by Sitecore APIs, however the developed chatbot doesn't use directly this method to obtain data on the available resorts. Indeed, having the necessity to manipulate resorts data without changing the content of the website and given the fact that not all information are useful, has been decided to export those relevant for the purpose.
To do this, an external application was written using a custom ASP.NET Web API that reads the relevant resort data and uploads it into an external database. Internally this custom Web API, queries Sitecore databases for all available resorts, then extracts targeted information and adds other custom fields of interest. Going more in detail, this service accesses the Master Sitecore database and, using some Sitecore APIs, navigates through the folders of the website up to the one that acts as the common root for all the resorts. Indeed the Sitecore.Context class allows to access the context database, automatically determined by the framework, and positioning on the item identified by a provided path, as in the following code

```
var db = Sitecore.Context.Database;
```

```
var resortRoot =
   db.GetItem("/sitecore/Content/Home/Plan And
   Book/Resorts");
```

Starting from there a deep recursive search was carried out to reach every single object of the resort. In fact, in Sitecore, any type of data, like a page, a paragraph or a multimedia file, is an item, which is characterized by some information but can also contain other items. Based on this, Sitecore lets the possibility to use the abstract class *Sitecore.Data.Items.CustomItem*, as base of an own class in which there is a property, with getter and setter, for each field of the item that you want to access. Thus a purpose built class, has been created implementing the abstract class above, so that for each resort, key properties, such as the unique identifier and those attributes needed to characterize the specific element, have been retrieved as reported in the code snippet below, and additional fields, such as the url of the resort and the its image, has been added.

```
[JsonObject(MemberSerialization.OptIn)]
public class MyCustomItem :
   Sitecore.Data.Items.CustomItem
{
        public MyCustomItem(Item innerItem) :
           base(innerItem){}

    public static implicit operator
       MyCustomItem(Item innerItem)
    {
            return innerItem != null ? new
               MyCustomItem(innerItem) : null;
    }

    public static implicit operator
       Item(MyCustomItem customItem)
    {
        return customItem != null ?
           customItem.InnerItem : null;
```

```
    }

    public string Title { get { return
       InnerItem["Name"]; } }

    public string CountryCode { get { return
       InnerItem["Country Code"]; } }

    public Dictionary<String, String> Attributes
    {
        get
        {
            string a = InnerItem["Attributes"];
            string[] splitted = a.Split('&');
            Dictionary<String, String> result =
               new Dictionary<String, String>();
            foreach(string s in splitted)
            {
                string so = s;
                string[] keyValue = s.Split('=');
                if(keyValue.Length == 2)
                    result.Add(keyValue[0],
                       keyValue[1]);
                else
                     result.Add(keyValue[0], "0");
            }
            return result;
        }
    }
}
```

Once retrieved all necessary data about resorts, is time to upload them in an external database. The one chosen is Azure Cosmos DB, that is a Microsoft's globally distributed, multi-model database. It supports multiple data models, including but not limited to document, graph, key-value, table, and column-family data models. For query these document types it offers API in different programming languages, such as

SQL API, MongoDB API, Cassandra API and others. Once again i used the Azure Portal to create this database. Indeed through the portal in few clicks is possible to create a new Azure Cosmos DB account using the Azure marketplace, create a database, add a new collection, or add simple data to an already existing one. Once an account has been created and a database filled, is possible to use the Data Explorer tool in the Azure Portal to navigate through its documents (they appear as JSON objects) or to write a personalized query on the fly.
All these operations can be done from the portal but also through coding. It requires to download the *Microsoft.Azure.DocumentDB* NuGet package in order to access the *DocumentClient* class, that allows interacting with your Cosmos DB account. Indeed, providing the endpoint url and the related key of the account, the DocumentClient class is able to manage databases, as well as collections of documents and the individual documents. In fact the DocumentClient exposes the *CreateDocumentAsync* method to create documents (modeled as JSON) and store them in a database. To do this it requires a class that models the object within the database, so has been created a class named JSONResort, shown below, where each property represents a piece of information that is stored for each document.

```
public class JSONResort
{
    [JsonProperty(PropertyName = "id")]
    public string Id { get; set; }

    public string CountryCode { get; set; }

    public bool HasSpa { get; set; }
    public string Continent { get; set; }

    public string CountryName { get; set; }

    public string ItemURL { get; set; }
    public string ImageURL { get; set; }
    public int Young {get; set;}
```

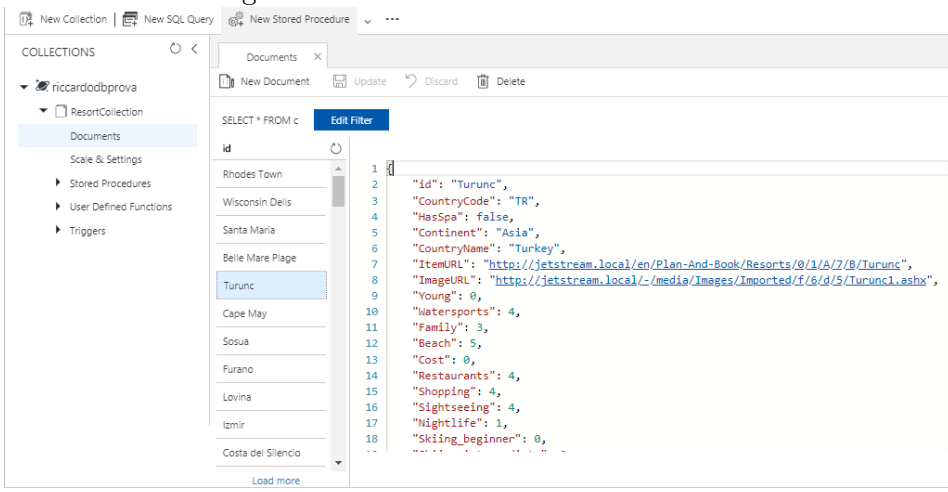
```
    public int Watersports { get; set; }
    public int Family { get; set; }
    public int Beach { get; set; }
    public int Cost { get; set; }
    public int Restaurants { get; set; }
    public int Shopping { get; set; }
    public int Sightseeing { get; set; }
    public int Nightlife { get; set; }
    public int Skiing_beginner { get; set; }
    public int Skiing_intermediate { get; set; }
    public int Skiing_advanced { get; set; }
    public int Snow_reliability { get; set; }
    public bool Lake_mountains_chk { get; set; }
    public bool Beach_chk { get; set; }
    public bool Ski_chk { get; set; }
    public float Latitude { get; set; }
    public float Longitude { get; set; }
    public int BusinessValue { get; set; }
    public int SunBatherValue { get; set; }
    public int RomanticValue { get; set; }
    public int FamilyValue { get; set; }
    public int PartyingValue { get; set; }
    public int SightseerValue { get; set; }

    public override string ToString()
    {
        return JsonConvert.SerializeObject(this);
    }
}
```

In the reported class, the *Id* property is required and used as unique identifier of the document, but apart this detail, all other properties was stored to enable the research mechanism of the chatbot. Acting in this way at the end of the process the target database include a collection of documents, each one representing a resort, identified by name and characterized by a list of key/value pairs that are the values associated to each field of the item that is in Sitecore, as shown in

Figure 4.3.

Figure 4.3. Azure CosmosDB Collection



4.3 JetstreamBot chatbot

This is the core of the project. Chatbots have gained lot of popularity in recently years and several companies have relied on these virtual assistants to offer a greater supports to their customers, with the purpose of increase their experience with the brand.
JetstreamBot is a chatbot, integrated in a travel website, aimed to help users along their navigation experience, to find the resort that is most suitable to their characteristics. This solution avoids the user having to navigate the site searching for a destination, with the risk of getting tired and abandon, in fact it will be the bot to do it for him. In addition, it puts the user at the center of the research, as it establishes a dialogue with him, aimed to capture additional information to those already obtained by Sitecore, so to better outline his profile and understand what are his intentions.
So the first study phase has been the definition of the chatbot purpose

and how it has to interact with users. JetstreamBot's purpose is encouraging users to visit one of the resorts present within the Jetstream travel website, interacting with them in a conversational way, in order to increase their engagement with the brand and the awareness that the proposed destination is tailored on their features. Moreover, also the chatbot approach to the conversation changes with respect to the type of user that navigate the website, thus different users will receive different question about their ideal destination and according to their answer, the conversation will proceed in several ways. This is fundamental in order to provide a great user experience, because the customer will be more satisfied if he feels understood and treated in a particular way.
In the conversation's personalization a key role is played by the Sitecore engine, who is able to define a user profile for the current customer, as well as a set of personalization rules for the website components. Thus, is in charge of the chatbot interacting both with the user and with Sitecore in order to being aware about the person who is using the website.
Once explained the chatbot role within the project, is time to see how it has been developed. Given the increasing usage of chatbots, several platforms are available to help developer in the bot creation and management, such as IBM Watson, Wit.ai or Api.ai, but i choose to use the Microsoft Bot Framework because, as explained it is a very powerful platform for building bots that lets the possibility to use the C# programming language, exactly as Sitecore, and because it provides the Direct Line REST API, which you can use to host the conversational agent on a website.

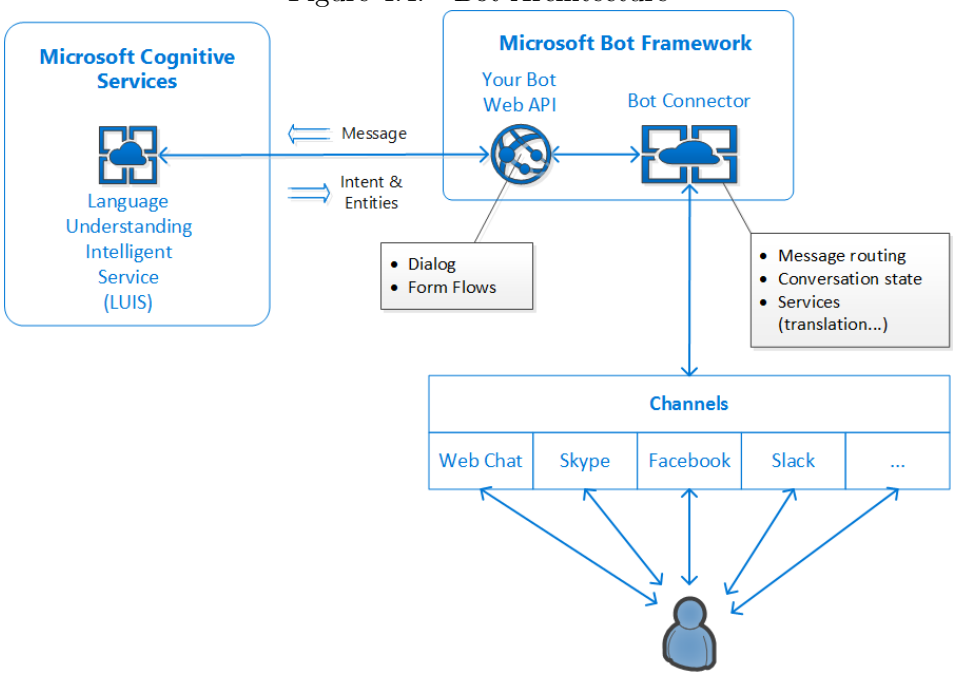
4.3.1 Language Understanding Intelligent Service (LUIS)

Inside the Microsoft Bot Framework, the Azure Bot Service speeds up development by providing an integrated environment that is purpose-built for developing chatbots. Using this service i created a C# bot

based on the Language understand template. This allows the application to understand what a person wants in its own words because it relies on LUIS to interpret the meaning of the messages and behavior accordingly.
LUIS is a service exposed as REST API, and consumable with an Azure subscription, that uses Natural Language Processing (NLP) techniques to process incoming user sentences and infer their meaning. It is based on several natural language concepts, including some particularly important, like intents, entities and utterances that are at base of LUIS understanding process. The prior function of the service is extracting intents from a user sentences, that are the goals, purposes, and motivations enclosed within these, regardless of the way user expresses them. Indeed, in natural language the same concept can be exposed in several ways but has to be recognized by who analyze the different sentences, for example a user can say "I want to fly to Milan" or "I'm looking for a flight to Milan", but the underling concept is the same.
A LUIS application, that can be created at https://www.luis.ai/, has to be able to recognize intents for which it has been thought, within a specified context, so that if the application is implied in forecast, it has to recognize intent like *search for forecast in a location*, *search forecast of a day*, or *search expected temperature of a day* and so on.
As well as the intents, also entities are important, and without them, finding intents is quite trivial and useless, because they represent the parameters of the intents, as nouns, subjects and objects. For example in the utterance "What's the weather in Turin?", the *search for forecast in a location* is recognized as intent and Turin as location entity, that is fundamental to accomplish the task. Entities can be pre-built and provided by LUIS, as number, dimension, datatime and several others, but can be also customized by the developer that define them.
Once defined intents and entities, LUIS in not able yet to classify incoming sentence, but it has to be trained. This phase is in charge of the developer, who has to provide some simple sentences, called utterances, aimed to learn the LUIS model. It is a delicate task from which depends the accuracy of the future classifications, since the model has to be trained to recognize all possible ways in which a user expresses

an intent.
At the end the overall architecture is the one reported in Figure 4.4, with the Bot Connector that takes care of delivering user messages to the bot, who immediately forwards them to the LUIS service endpoint developed for the application. Here messages are evaluated and transformed in intents and entities, then returned to the chatbot. All this is possible using the Dialog model provided by the Bot Builder SDK. When the Language Understand template is used, the frame-

Figure 4.4. Bot Architecture



work automatically creates an empty LUIS application to associate to the bot, with only the predefined *None* intent, that you can use to handle all those cases in which any other intent doesn't matches incoming data. In order to recognize the pattern cards delivered by the website i created a *SelectUserProfile* intent able to understand when an input contains a user profile and associating it to a purpose-built

PatternCard entity. LUIS service provides a response including intents and entities, in addition to their reliability score, as JSON document, hence to understand it, the client that consumes the service has to parse the incoming data and extracting information from the parsed JSON object.
The easiest approach to parse the data and reuse them is to exploit the Bot Builder SDK and a special Dialog class provided by it, the *LuisDialog<object>*. This class automatically takes care of parsing the incoming data and deserializing JSON, lifting the developer from this task. To use these functionality, programmer has only to write a class inheriting from the LuisDialog<object> one, and to provide a method for each intent that the service is able to recognize. Each one of these methods has to be decorated with the *LuisIntent* attribute, who takes an additional string parameter representing the name of the intent, indicating that the targeted method has to be used as handler of the associated intent. Thus i modified the basic class that inherits from LuisDialog<object> and that is already provided with the template, by adding the method reported in the following snippet of code.

```
[LuisIntent("SelectUserProfile")]
public async Task
   SelectUserProfileIntent(IDialogContext context,
   LuisResult result)
{
    string message = "Hi ";
    string profile = "";
    foreach (var e in result.Entities)
    {
        profile = e.Entity;
        message += profile;
    }

    PatternProfile = "userProfile: " + profile;

    PossibleResorts.Clear();
    DistinctContinents.Clear();
```

```
        ExtractedResorts.Clear();
        DistinctCountries.Clear();

        if (PatternProfiles.Contains(PatternProfile))
            {
            await context.PostAsync($"{message}");
            GeneralForm gf = new GeneralForm(profile);
            context.Call(gf.BuildFormDialog(FormOptions.PromptInStart
               FormComplete);
        }
        else
        {
            await context.PostAsync("Hi, I'm your
               virtual assistent. I cannot help you
               yet, please first visit the site");
            context.Wait(MessageReceived);
        }
}
```

When this handler is invoked, the user profile is extracted and, if valid, a purpose-built *FormFlow* is created for handling a guided conversation aimed to collect information about the user.

4.3.2 FormFlow

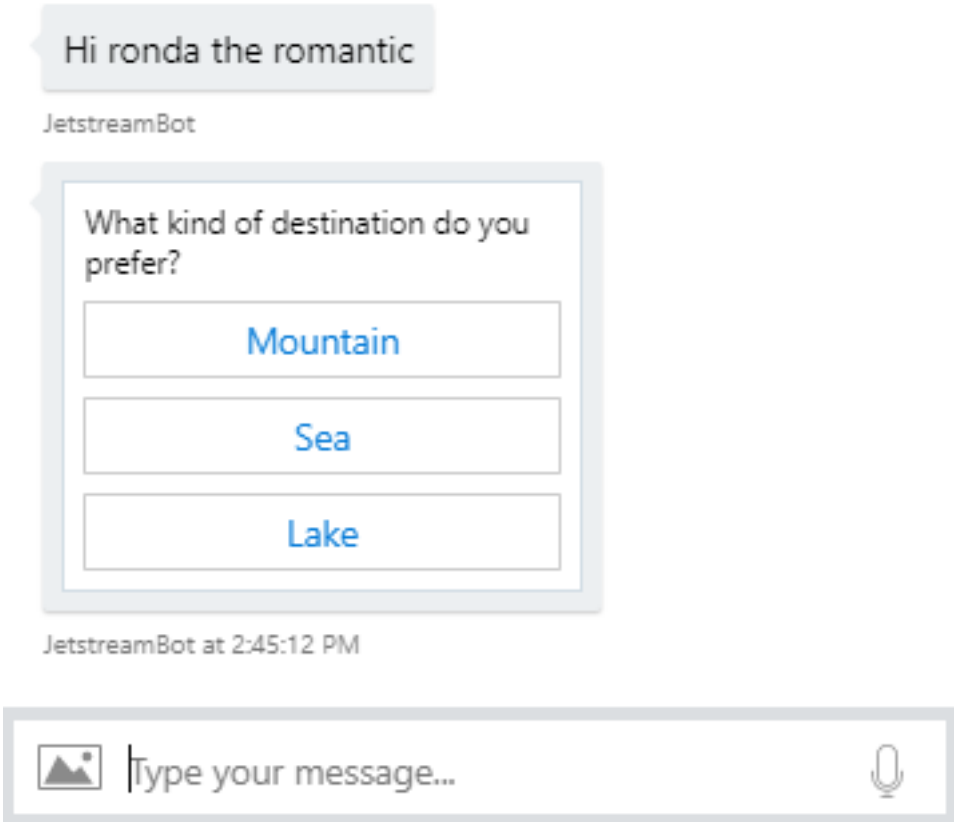
Chatbots are conversational agents, thus their main purpose is to establish and maintain conversations with one or more users. The Bot Builder SDK uses dialogs to model these conversations and manage their flows, invoking them when the message controller receives an activity message. However, if on one side dialogs offer a very flexible way to handle conversations, can become difficult using them for handling guided conversations in which bot behavior depends from user messages and so many possibilities have to be expected and handled, maintaining all possible information retrieved along it. To overcome this problem the Bot Builder SDK provides the *FormFlow*.

FormFlow automatically generates dialogs for supporting a guided conversation that follows a guideline defined by the developer. Indeed, he can specify all required information in a C# class using properties and enums, then supplying it to the FormFlow, it takes care of the dialogs required to retrieve these information. Thus i wrote a class with properties representing the information to retrieve from the user and decorated with attributes provided by the framework. Indeed, FormFlow in addition to letting to the developer the possibility to specify the aim of the conversation, allows also to customize it providing the way the requests have to be performed and how to handle unexpected responses, as in the code snippet shown below.

```
[Prompt("What kind of destination do you prefer?
    {||}")]
[Template(TemplateUsage.NotUnderstood,
    NotUnderstoodMsg)]
public DestinationTypeOptions DestinationType {
    get; set; }
```

Here is shown how the bot uses the *Prompt* attribute to customize the question supplied to the user for retrieving the destination type he is searching for, and the *Template* attribute to provides a standard way for reacting to unexpected responses. The properties must be integrals, floating points, string, DateTime or enumerations, like in this case, letting the developer to specify desired options.
For each property the FormFlow will expose the related question to the user, prompting also the options available, as in Figure 4.5, letting to him the possibility to write the response as text or selecting directly one of the proposal, then the choice is stored by the framework. Once defined the information that the chatbot has to retrieve, the FormFlow automatically manages the conversation flow formulating each request, and collecting each answer, however without additional guidelines, they are supplied one after the other in a static way, which is not very pleasant in terms of naturalness and fluidity of the conversation, because some questions could be related to other answers, so

Figure 4.5. Chatbot interaction



optionally omitted or proposed in a different fashion in order to naturalize the conversation. Likewise, the JetstreamBot bot according with the pattern card recognized and the received responses, dynamically manage the conversation flow creating step by step the next questions. This is useful to supply different questions to different users, for example if he has been classified as sea lover, it would not make much sense asking him if is searching for a mountain destination rather then for a sea one, moreover the question "Do you like skiing?" is meaningful only if him wants to go the mountain, while it will be inappropriate for the user previously mentioned.

In order to perform these customization, JetstreamBot uses the *FormBuilder*, that is a reach and versatile framework for managing FormFlow dialogs and exploiting their advanced features. Thanks to the FormBuilder the developer can use the Prompt and the Template attribute previously mentioned and dynamically managing conditional questions. User have also the possibility to go back on previous choice and change it, and the chatbot will automatically change also its internal information and will react as required to continue the conversation.

4.3.3 Destinations research

When the chatbot has collected all required information, its task is to retrieve those resorts that could interest the user. To do this it establish a set of values for the keys used to categorize the resorts based on the user profile received by Sitecore and the additional information obtained along the conversation, then it queries the external CosmosDB database for those destinations that match the values set. Wanting to leave the widest range of possible choices, without geographical limits, the user is not asked questions about the destination nation or continent, but rather the characteristics sought in this one, such as the presence of nightlife, the possibility of doing sports and so on. Thus, once queried the external database, the destinations found are grouped by continent, letting the user to choose what is his preference, then if more that one nation is available within it, also in this case the choice is let to the user, while if there is only one, it is immediately shown. Cosmos DB allows executing queries against documents in the collections using the preferred syntax, like SQL or LINQ, exactly as you would to interrogate the classic relational databases, letting to achieve entire documents as well as targeted fields of certain documents. In addition it allows to query the documents and retrieve wanted information based on a model composed as JSON object. In this way is possible to create your own class representing a JSON document ad fill it with data retrieved from the database.
An example of how this works is reported in the following code that is

a query executed for a targeted user profile, using the LINQ syntax.

```
query = from f in
    client.CreateDocumentQuery<JSONResort>(
        UriFactory.CreateDocumentCollectionUri(DatabaseName,
           CollectionName),
        new FeedOptions { MaxItemCount = 10 })
            where f.Cost >= Cost &&
                f.Restaurants >= Restaurants &&
                f.Beach_chk == Beach_chk &&
                f.Ski_chk == Ski_chk &&
                f.Lake_mountains_chk ==
                   Lake_mountains_chk &&
                f.Family <= Family &&
                f.Watersports >= Watersports &&
                f.Skiing_advanced >=
                   Skiing_advanced &&
                f.Skiing_intermediate >=
                   Skiing_intermediate &&
                f.Skiing_beginner >=
                   Skiing_beginner &&
                f.Snow_reliability >=
                   Snow_reliability &&
                f.HasSpa == Spa_chk
            orderby f.BusinessValue descending
            select new SimpleResort { Id = f.Id,
               CountryName = f.CountryName,
            Continent = f.Continent, ItemURL =
               f.ItemURL, ImageURL = f.ImageURL };
```

For each user profile JetstreamBot executes purpose-built queries based on different values, according to those aspects considered relevant for the pattern card associated to the user by Sitecore and for the choice done by the user himself during the conversation. In this ways different users will achieve different suggestions about their destinations.
The main purpose of the chatbot is suggesting a resort that could interest the user, so at the end of the dialogue it proposes to him a

destination. This phase is quite important as although the suggestion can be suitable to the user and perfect with respect to its characteristics, it could be discarded or could affect the navigation experience if not presented in the appropriate way. Thus rather then simply suggesting a destination name, letting to the user the task of navigate all the website searching for it, or redirecting him directly on the target page, that could be too invasive, i choose to use hero cards provided by the Bot Connector. This is a card that allows exchanging image and buttons during a conversation. Thanks to this functionality the bot delivers to the user a message in which the name of the suggested resort is accompanied by a large image of it achieved by its page within the travel website and a button that gives the opportunity of navigating to the resort's page.
Furthermore, if more than one resort is found for the nation chosen, an additional button allows to discard the presented destination and seeing another one. This possibility meets the needs of different people, with different tastes, so as to satisfy the user as much as possible, making him always find a suitable opportunity for him. However also this choice can quickly lead to a bad user experience, indeed if resorts are proposed without any logic, the user may get tired and lose confidence in the reliability of the proposals. In order to avoid this, a evaluation mechanism has been created for ranking destinations according to the interest shown by the users themselves. In fact is useless to propose those resorts that have never been successful, while if it has captured the attention of multiple users, then it should be placed among the suggestions.
Starting from the assumption that if a user stays for a long time on a website page, is because he is pretty interested to it, the idea was to give a point every time a resort page is considered interesting. In order to do this, a JavaScript function has been embedded into the website code, so that it is called when a user visits a resort page and stays on it at least for a predefined time. This function uses a purpose-built .NET web service that, obtained the pattern card associated to the current user, increase the resort's score inside the Cosmos DB database for the targeted category of users.

In this way when the chatbot queries the database for the resorts most suitable to the current user, it sorts them according to their score for the pattern card associated to him, so that the resorts with an higher value will be shown before the others.
This additional mechanism enforces the concept of pattern cards and of providing the right content to the user, that is the basic concept in order to increase the user engagement with respect to the brand.

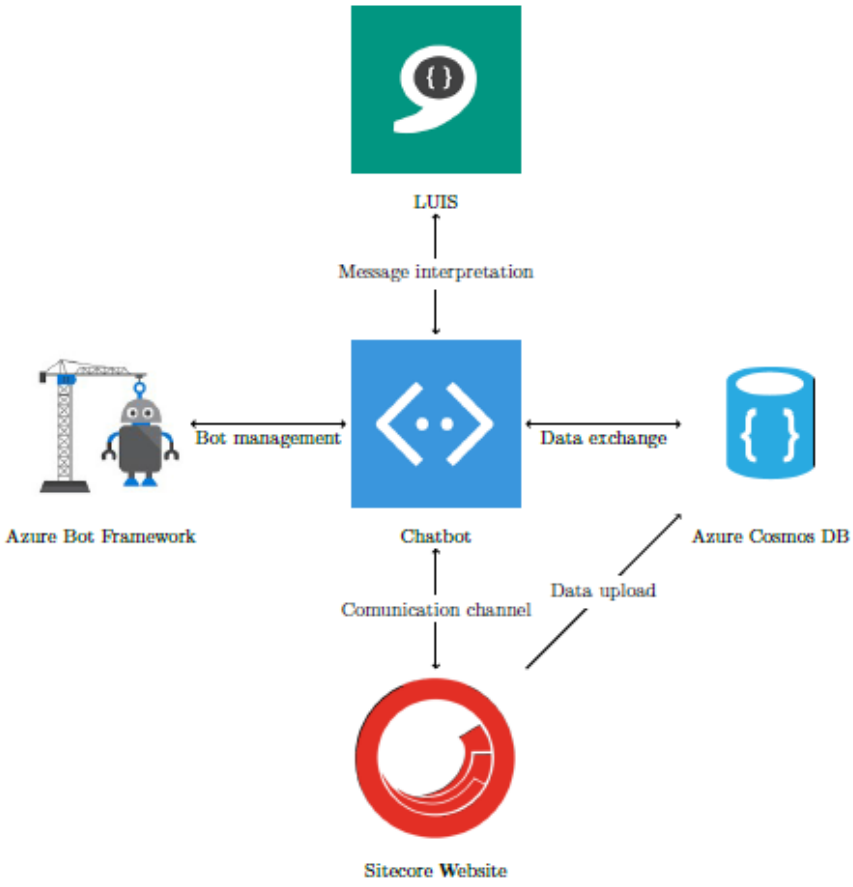
Chapter 5

Conclusions and further works

This thesis project concerns the use of the Microsoft Bot Framework for developing a chatbot to integrate in a Sitecore's website, able to exploit its personalization mechanism for delivering conversations tailored on the users and increasing their engagement with the brand. The framework above speeds up the chatbot development process, providing templates that can be used to create several type of bot, ranging from basic ones to those relying on artificial intelligence or backed by a knowledge base to use for responding to common questions. This allows developers to create with few clicks a bot equipped with all those basic functionality that enable to interact with it.
In addition to the templates, the frameworks provides also a powerful open source SDK that enables to compose several dialogs, create a guided conversation and utilize artificial intelligence as LUIS. Once created their chatbots, developers can modify and develop them downloading the source codes, rather than doing it directly on the portal, exploiting the online editor, furthermore the framework allows to create, delete and manage the channels on which the bots can reached, monitoring the use of each one based on the messages exchanged in a chosen range of time. Finally, it provides also a useful web chat connected to the bot in order to test it before being published on the

network.
While the Sitecore's website use its personalization engine to deliver personalized content to the user, the chatbot is able to learn from the site itself the characteristics of the user with which it is interacting, leading conversations that are built based on the its profile and that change depending on the answers provided.
The developed chatbot exploits the Language Understanding Intelligent Service (LUIS) provided by Microsoft to understand messages meaning and behavior accordingly establishing a conversation that ends with the suggestion to the user of one or more resorts that are most suitable to their characteristics and intentions. The destination selection process is backed by a Cosmos DB, a globally distributed, multi-model database service, in which are stored relevant data of resorts present in the travel site, constantly updated with users preferences, in order to deliver always the best results.
Hence the overall architecture can be seen as in Figure 5, with the chatbot at the center that is hosted into the Sitecore's website and communicates with Azure Cosmos DB to retrieve resorts information and with the LUIS service for the messages interpretation. Finally the Bot Framework, from which the bot can be managed. In the future some improvement can be applied to offer a user experience even better. Of sure some effort can be spent in order to expand the chatbot functionality, letting the possibility to interact with it for several additional purposes, such as for booking a flight for the chosen destination, reserve a car for moving on the place or connecting to others external services in the world of tourism. This can be achieved by an more intensive use of LUIS, so that to leave at the user a greater initiative along the conversation, relying on the artificial intelligence to understand user needs. Furthermore the chatbot could exploit an holiday calendar in order to provides suggestion based on some specific festivity and events available on the period chosen for traveling, to check which could be the more interesting place.
Another possible improvement is to exploit the user localization in order to make aware the bot about the distance that separate him from

Figure 5.1. Overall Architecture



the potential destinations, so to delivering more accurately suggestions. For example a user that wants to stay out only for a weekend, probably will prefer a resort not too far.
Finally, in addition to those information obtained from Sitecore website, the chatbot could also exploit data retrieved from user navigation on external sites and his researches on the network, in order to have a more complete picture about him.

Bibliography

[1] Suket Arora, Kamaljeet Batra, and Sarabjit Singh. *Dialogue System: A Brief Review*. 2013.

[2] Kat Austin. *The History and Future of Chatbots*. 2017. URL: http://inthechat.com/chatbots/the-history-and-future-of-chatbots/.

[3] Deane Barker. «Web Content Management: systems, features, and best practices». In: O'Reilly Media, Inc., 2015. Chap. 1, p. 10.

[4] Matt Carbone. *Remembering SmarterChild*. 2016. URL: http://blog.talla.com/remembering-smarterchild.

[5] Rollo Carpenter. *About the Jabberwacky AI*. 2018. URL: http://www.jabberwacky.com/j2about.

[6] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*. 2014.

[7] EQT. *Sitecore*. URL: https://www.eqtpartners.com/Investments/Current-Portfolio/sitecore/.

[8] Albert Gatt and Emiel Krahmer. *Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation*. 2017.

[9] Dr. Alfio Gliozzo et al. «Building Cognitive Applications with IBM Watson Services: Volume 1 Getting Started». In: RedBooks, 2017. Chap. 3, pp. 30–39.

[10] Charles Hayden. *How Eliza work*. URL: http://www.chayden.net/eliza/instructions.txt.

[11] Daniel Jurafsky and James H. Martin. «Speech and Language Processing». In: 2017. Chap. 28, pp. 418–440.

[12] Alireza Mansouri, Lilly Suriani Affendey, and Ali Mamat. «Named Entity Recognition Approaches». In: *International Journal of Computer Science and Network Security* (2008).

[13] Michael F. McTear. «Spoken dialogue technology: enabling the conversational user interface». In: *ACM Comput. Surv.* 34 (2002), pp. 90–169.

[14] Danny Miller. *SMARTERCHILD AND ELIZA*. 2007. URL: http://k2xl.com/wordpress/smarterchild-and-eliza/.

[15] Diana Perez-Marin and Ismael Pascual-Nieto. «Conversational Agents and Natural Language Interaction». In: Information Science Reference, 2011. Chap. 1, pp. 8–10.

[16] Lars Birkholm Petersen. *Best Practices for Developing Personas with the Sitecore Customer Engagement Platform*. 2012.

[17] Alan Ritter, Colin Cherry, and William B. Dolan. *Data-Driven Response Generation in Social Media*. 2011.

[18] Margaret Rouse. *web content management system (WCMS)*. URL: http://searchcontentmanagement.techtarget.com/definition/web-content-management-WCM.

[19] Sitecore. *Architecture overview*. URL: https://doc.sitecore.net/sitecore_experience_platform/81/setting_up_and_maintaining/xdb/platform/architecture_overview.

[20] Sitecore. *Processing overview*. URL: https://doc.sitecore.net/sitecore_experience_platform/81/setting_up_and_maintaining/xdb/platform/processing_overview.

[21] Sitecore. *Sitecore Launches Sitecore Experience Platform 8.1 to Deliver World-Class Context Marketing at Scale*. URL: https://www.sitecore.com/company/press-and-media/press-releases/2015/10/sitecore81.

[22] Robert Standefer and Kamran Iqbal. *Bot analytics*. 2017. URL: https://docs.microsoft.com/en-us/bot-framework/bot-service-manage-analytics.

[23] Robert Standefer et al. *Dialogs in the Bot Builder SDK for .NET*. 2017. URL: https://docs.microsoft.com/en-us/bot-framework/dotnet/bot-builder-dotnet-dialogs.

[24] Wallace and Richard S. «The Anatomy of A.L.I.C.E.» In: *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*. Ed. by Robert Epstein, Gary Roberts, and Grace Baber. Springer Netherlands, 2009, pp. 181–210.

[25] Joseph Weizembaum. «ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine». In: *Communications of the ACM* (1996).

[26] John West. «Professional Sitecore Development». In: John Wiley & Sons, Inc., 2012. Chap. 1, p. 2.

[27] Phil Wicklund. «Practical Sitecore 8 Configuration and Strategy: A User Guide for Sitecore's Content and Marketing Capabilities». In: Apress, 2015. Chap. 1, pp. 9–12.

[28] Natalie Wolchover. *How the Cleverbot Computer Chats Like a Human*. 2011. URL: https://www.livescience.com/15940-cleverbot-computer-chats-human.html.