CHAPTER - 1

INTRODUCTION

1. What is a Chat-bot?

A chatbot is an artificial intelligence (AI) program that simulates interactive human conversation by using key pre-calculated user phrases and auditory or text-based signals. Chatbots are frequently used for basic customer service and marketing systems that frequent social networking hubs and instant messaging (IM) clients. They are also often included in operating systems as intelligent virtual assistants.

A chatbot is also known as an artificial conversational entity (ACE), chat robot, talk bot, chatterbot or chatterbox.

Early classic chatbots include ELIZA (1966), a simulation of a psychotherapist, and PARRY (1972), based on paranoid schizophrenic behavior.

In 1950, Alan Turing proposed the Turing test intelligence criteria set, which depends on undetectable program simulation of human user behavior and activity. The Turing test generated high interest in the ELIZA program, which makes people believe they are chatting with human beings.

Modern chatbots are frequently used in situations in which simple interactions with only a limited range of responses are needed. This can include customer service and marketing applications, where the chatbots can provide answers to questions on topics such as products, services or company policies. If a customer's questions exceed the abilities of the chatbot, that customer is usually escalated to a human operator.

Chatbots are often used online and in messaging apps, but are also now included in many operating systems as intelligent virtual assistants, such as Siri for Apple products and Cortana for Windows. Dedicated chatbot appliances are also becoming increasingly common, such as Amazon's Alexa. These chatbots can perform a wide variety of functions based on user commands.

2. Why does one need a Chat-bot?

There are reasons for that like getting rid of routine tasks and simultaneous processing of multiple requests from users. Besides, a tremendous speed of processing users' requests with chatbots helps gaining customers' loyalty.

Consumers also benefit from chatbots and they are getting increasingly interested in this technology. A study presented at the 4th International Conference on Internet Science in November, 2017 identified reasons why people choose to interact with chatbots. According to this research, the main factors that motivate people to use chatbots are:

- **Productivity:** Chatbots provide the assistance or access to information quickly and efficiently.
- **Entertainment:** Chatbots amuse people by giving them funny tips, they also help killing time when users have nothing to do.
- **Social and relational factors:** Chatbots fuel conversions and enhance social experiences. Chatting with bots also helps to avoid lonliness, gives a chance to talk without being judged and improves conversational skills.

3. Types of Chatbot

Depending on how the specific bots were programmed, we can divide them into two large groups: working according to pre-prepared commands (simple chatbot) and trained (smart or advanced chatbot).

Simple chatbots work based on pre-written keywords that they understand. Each of these commands must be written by the developer separately using regular expressions or other forms of string analysis. If the user has asked a question without using a single keyword, the robot cannot understand it and, as a rule, responds with messages like “sorry, I did not understand”.

Smart chatbots rely on artificial intelligence when they communicate with users. Instead of pre-prepared answers, the robot responds with adequate suggestions on the topic. In addition, all the words said by the customers are recorded for later processing. However, the Forrester report “The State of Chatbots” points out that artificial intelligence is not a magic and is not yet ready to produce marvelous experiences for users on its own.

4. What Chat-bots can do?

Both startups and savvy companies are now incorporating interactive agents into their daily operations, communication with customers and sales processes. Chatbots can help to:

Improve customer service. It is the best option for those who don’t want their customers to:

Wait for operator’s answer — “Stay on the line, your call is very important to us” is always annoying, isn’t it?

Search for an answer in the FAQ — as a rule users don’t have time for scrolling dozens of pages with instructions.

Streamline the shopping process. It only takes to write what you want to the chatbot and the bot will send the information to the sales department. You don’t need to repeat several times “I need the same, but with metal buttons”. Besides, the chatbot remembers your preferences and uses this information when you return.

Personalize communication. A chatbot answers the specific questions of visitors instead of displaying a long list of information. The more attention a customer gets the greater his desire to buy something.

Improve a response rate. About 90% of questions sent from Facebook business pages remain unanswered. Chatbot responds to 100% of messages and converts more visitors into buyers.

Automate repetitive tasks. Most customers want to get answers on the same questions — When do you work? What is your location? Do you make deliveries? In order not to write the same answers every time, make a chatbot. It reduces your employees’ workload.

5. Training Objective

We, a team of 4 people had a main goal for creating a Chat-Bot for CRIS (Centre for Railways Information System) Employee website. This Chat-bot will help the employee to work efficiently on the employee website. The Chat-bot will help the user by providing them with links to the webpage of the site they want to access at one go, just by typing a message.

This will reduce the number of steps in which the employee reaches that page and perform it. It will help employees work efficiently and happily as now they also have a friend with them even while working. This Chat-bot will be like a friend who will have all answers to your queries.

CHAPTER – 2

TOOLS AND TECHNOLOGY USED

Technologies used:

Technologies used in the system are Google’s Dialogflow, . All the Technologies used in this program / software are described as follows:

1. Google’s Dialogflow

Dialogflow (formerly Api.ai, Speaktoit) is a Google-owned developer of human–computer interaction technologies based on natural language conversations. The company is best known for creating the Assistant (by Speaktoit), a virtual buddy for Android, iOS, and Windows Phone smartphones that performs tasks and answers users' question in a natural language. Speaktoit has also created a natural language processing engine that incorporates conversation context like dialogue history, location and user preferences.

In May 2012, Speaktoit received a venture round (funding terms undisclosed) from Intel Capital. In July 2014, Speaktoit closed their Series B funding led by Motorola Solutions Venture Capital with participation from new investor Plug and Play Ventures and existing backers Intel Capital and Alpine Technology Fund.

In September 2014, Speaktoit released api.ai (the voice-enabling engine that powers Assistant) to third-party developers, allowing the addition of voice interfaces to apps based on Android, iOS, HTML5, and Cordova. The SDK's contain voice recognition, natural language understanding, and text-to-speech. api.ai offers a web interface to build and test conversation scenarios. The platform is based on the natural language processing engine built by Speaktoit for its Assistant application. Api.ai allows Internet of Things developers to include natural language voice interfaces in their products. Assistant and Speaktoit's websites now redirect to api.ai's website, which redirects to the Dialogflow website.

Google bought the company in September 2016 and was initially known as API.AI; it provides tools to developers building apps ("Actions") for the Google Assistant virtual assistant. It was renamed on 10 October 2017 as Dialogflow.

The organization discontinued the Assistant app on December 15, 2016.

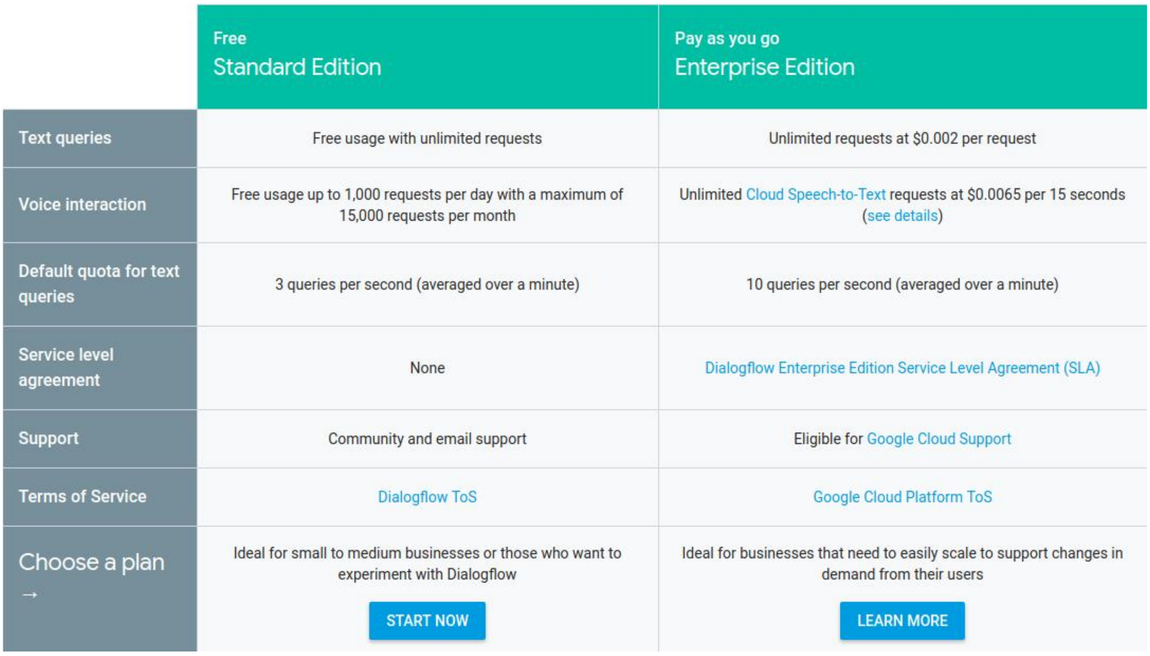
Voice and conversational interfaces created with Dialogflow works with a wide range of devices including phones, wearables, cars, speakers and other smart devices. It supports 14+ languages including Brazilian Portuguese, Chinese, English, Dutch, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish and Ukrainian. Dialogflow supports an array of services that are relevant to entertainment and hospitality industries. Dialogflow also includes an analytics tool that can measure the engagement or session metrics like usage patterns, latency issues, etc.

Why choose Dialogflow?

There are several reasons for choosing Dialogflow:

- **Price**

If you just want to learn building a Chatbot or you don’t have many users, a Standard Edition is totally free. As you can see below



	Free Standard Edition	Pay as you go Enterprise Edition
Text queries	Free usage with unlimited requests	Unlimited requests at $0.002 per request
Voice interaction	Free usage up to 1,000 requests per day with a maximum of 15,000 requests per month	Unlimited Cloud Speech-to-Text requests at $0.0065 per 15 seconds (see details)
Default quota for text queries	3 queries per second (averaged over a minute)	10 queries per second (averaged over a minute)
Service level agreement	None	Dialogflow Enterprise Edition Service Level Agreement (SLA)
Support	Community and email support	Eligible for Google Cloud Support
Terms of Service	Dialogflow ToS	Google Cloud Platform ToS
Choose a plan →	Ideal for small to medium businesses or those who want to experiment with Dialogflow START NOW	Ideal for businesses that need to easily scale to support changes in demand from their users LEARN MORE

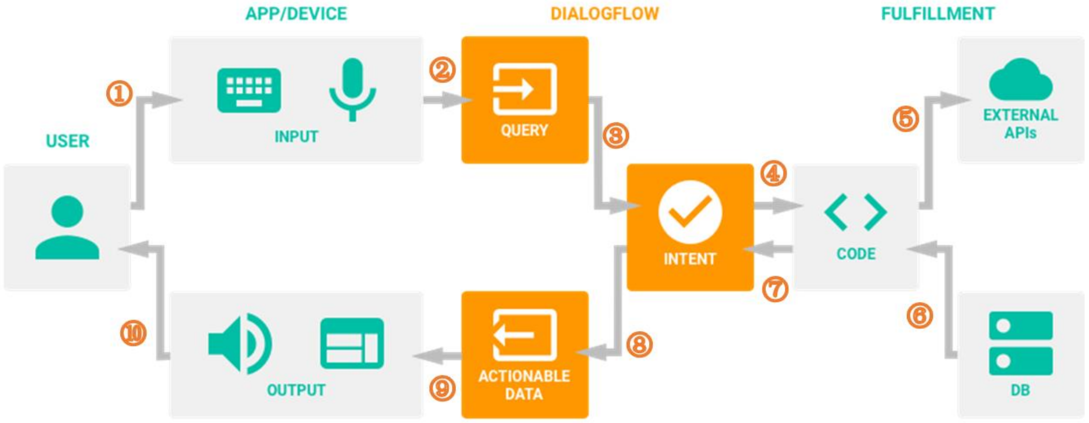
- **Multi-channel easy integration**

Dialogflow provides one-click integrations to most popular messaging Apps like Facebook Messenger, Slack, Twitter, Kik, Line, Skype, Telegram, Twilio and Viber. Even to some voice assistants like Google Assistant, Amazon Alexa and Microsoft Cortana.

- **Natural Language Processing(NLP)**

Compared to some platforms which works on predefined questions like Chatfuel, Dialogflow can offer better user experience with NLP. DialogFlow Agents are pretty good at NLP.

How do Chatbots work?



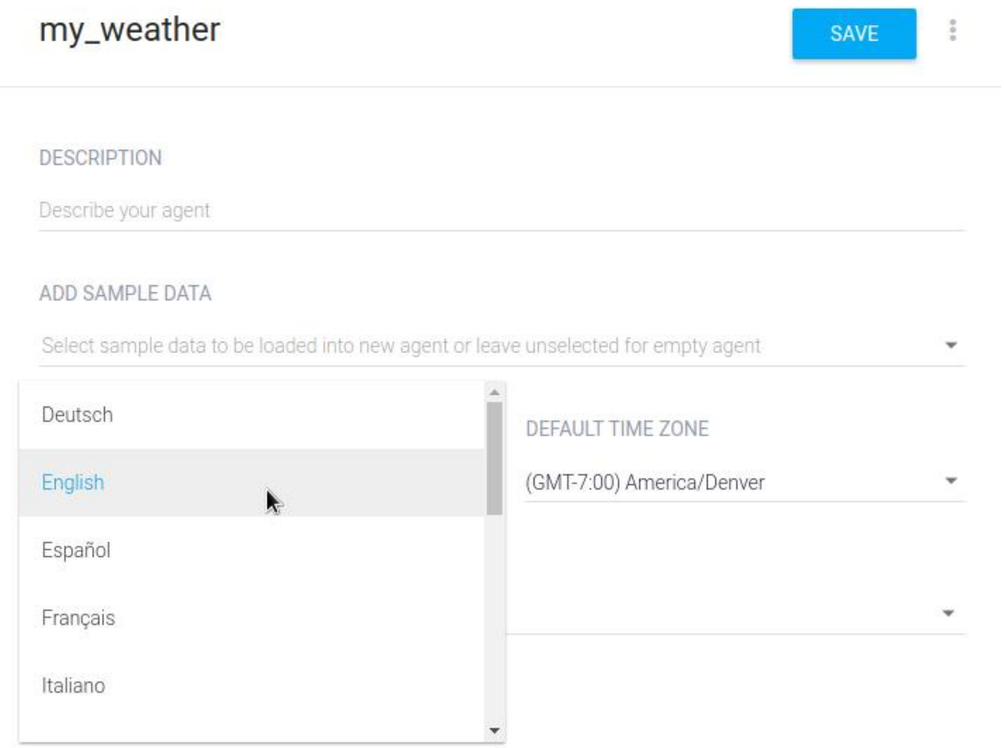
There are detailed steps:

1. A user sends a text/voice message to a device or an App
2. The App/Device transfers the message to Dialogflow
3. The message is categorized and matched to a corresponding intent (Intents are defined manually by developers in Dialogflow)
4. We define following actions for each intent in the fulfillment (Webhook).
5. When a certain intent is found by Dialogflow, the webhook will use external APIs to find a response in external data bases.
6. The external data bases send back required information to the webhook.
7. Webhook sends formatted response to the intent.
8. Intent generates actionable data acoording to different channels.
9. The actionable data go to output Apps/Devices.
10. The user gets a text/image/voice response.

How to build a chat-bot using Dialogflow?

Create an agent

If you don’t already have a Dialogflow account, sign up. If you have an account, login. Click on **Create Agent** in the left navigation and fill in the fields.



my_weather

DESCRIPTION

Describe your agent

ADD SAMPLE DATA

Select sample data to be loaded into new agent or leave unselected for empty agent

DEFAULT TIME ZONE

(GMT-7:00) America/Denver

Give a name and set a language to your agent. Then, click the Save button.

Create an intent

An intent maps what a user says with what your agent does. This first intent will cover when the user asks for the weather.

To create an intent:

Click on the plus icon add next to Intents. You will notice some default intents are already in your agent. Just leave them be for now.

Enter a name for your intent. This can be whatever you'd like, but it should be intuitive for what the intent is going to accomplish.

In the Training Phrases section, enter examples of what you might expect a user to ask for. Since you're creating a weather agent, you want to include questions about locations and different times. The more examples you provide, the more ways a user can ask a question and the agent will understand.Enter these examples:

What is the weather like

What is the weather supposed to be

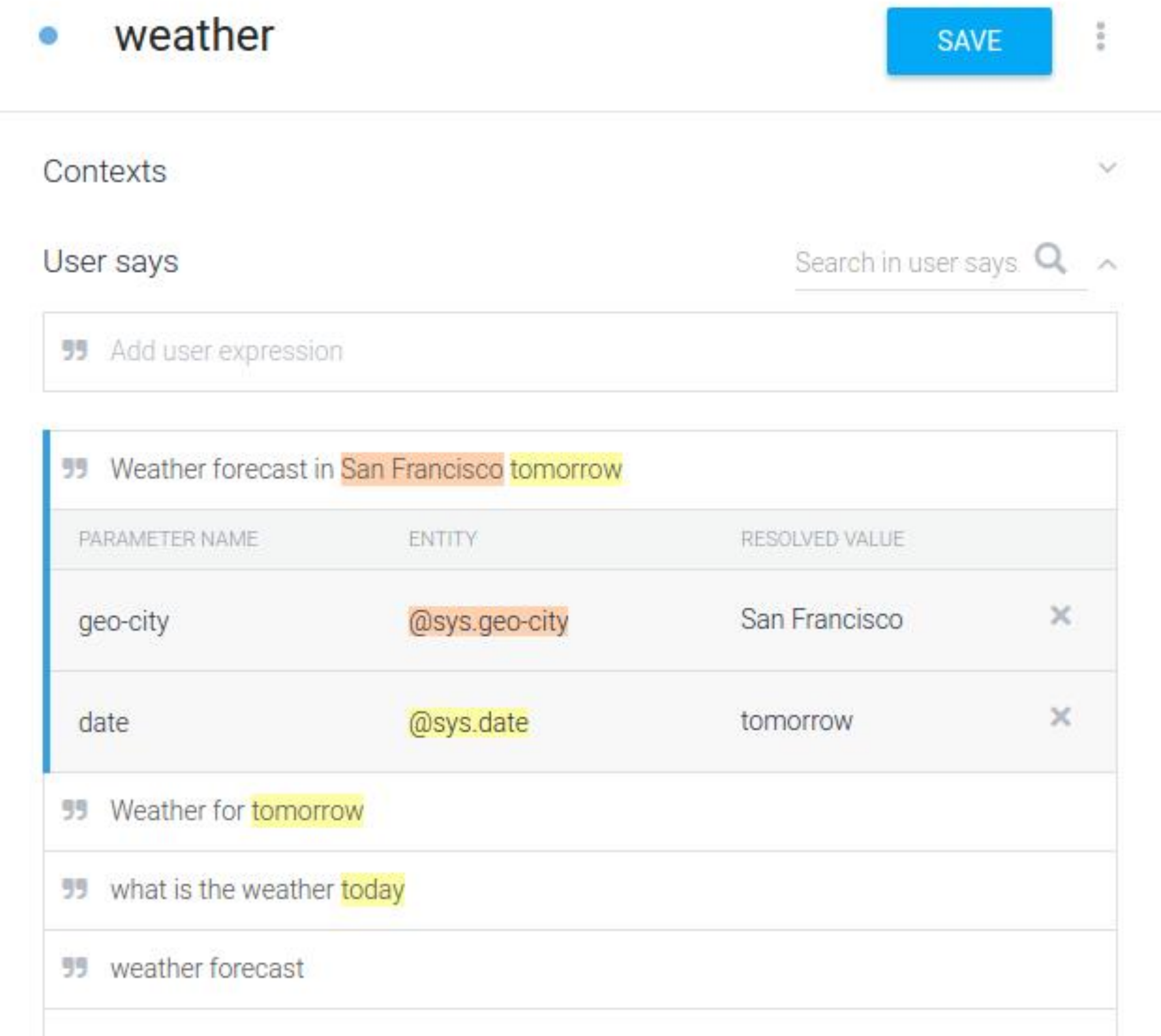
Weather forecast

What is the weather today

Weather for tomorrow

Weather forecast in San Francisco tomorrow

In the last three examples you’ll notice the words today and tomorrow are highlighted with one color, and San Francisco is highlighted with another. This means they were annotated as parameters that are assigned to existing date and city system entities. These date and city parameters allow Dialogflow to understand other dates and cities the user may say, and not just “today”, “tomorrow”, and “San Francisco”.



weather

Weather forecast in San Francisco tomorrow

PARAMETER NAME	ENTITY	RESOLVED VALUE
geo-city	@sys.geo-city	San Francisco
date	@sys.date	tomorrow

Weather for tomorrow

what is the weather today

weather forecast

Click **Save**

Add response

Now you'll add basic responses to the intent so the agent doesn't just sit there in awkward silence. As mentioned before, responses added to an intent don't use external information. So this will only address the information the agent gathered from the user's request.

If you've navigated away from the "weather" intent, return to it by clicking on Intents and then the "weather" intent.

In the same way you entered the Training Phrases, add the lines of text below in the Response section:

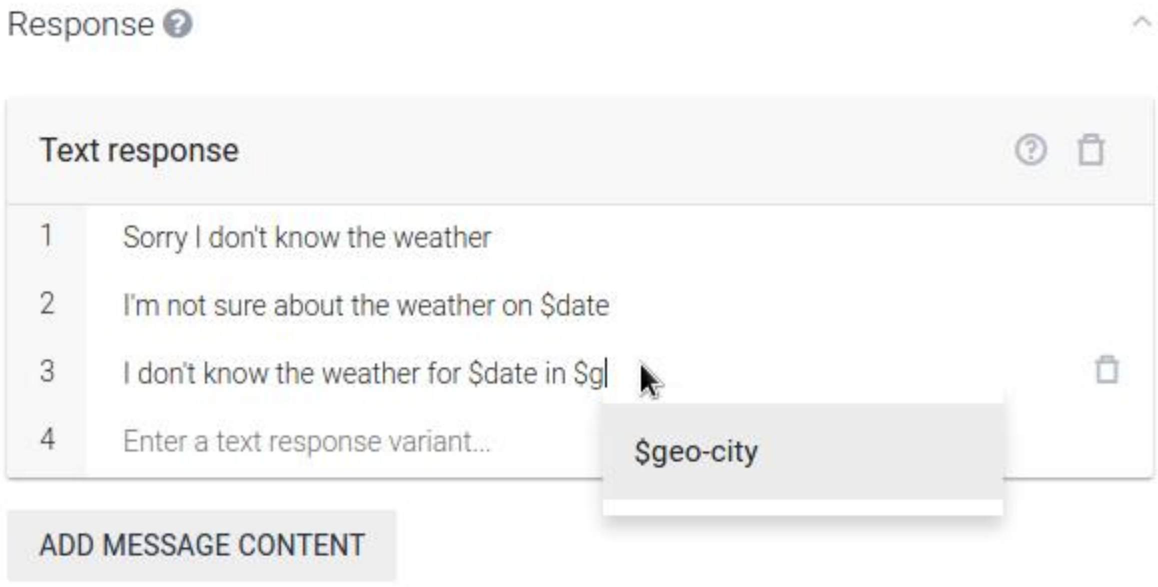
Sorry I don't know the weather

I'm not sure about the weather on $date

I don't know the weather for $date in $geo-city but I hope it's nice!

You can see the last two responses reference entities by their value placeholders. $date will insert the date from the request, and $geo-city will insert the city.

When the agent responds, it takes into account the parameter values gathered and will use a reply that includes those values it picked up. For example, if the request only includes a date, the agent will use the second response from the list.



Response

Text response

1 Sorry I don't know the weather

2 I'm not sure about the weather on $date

3 I don't know the weather for $date in $g

4 Enter a text response variant...

Click **Save**

Try it out

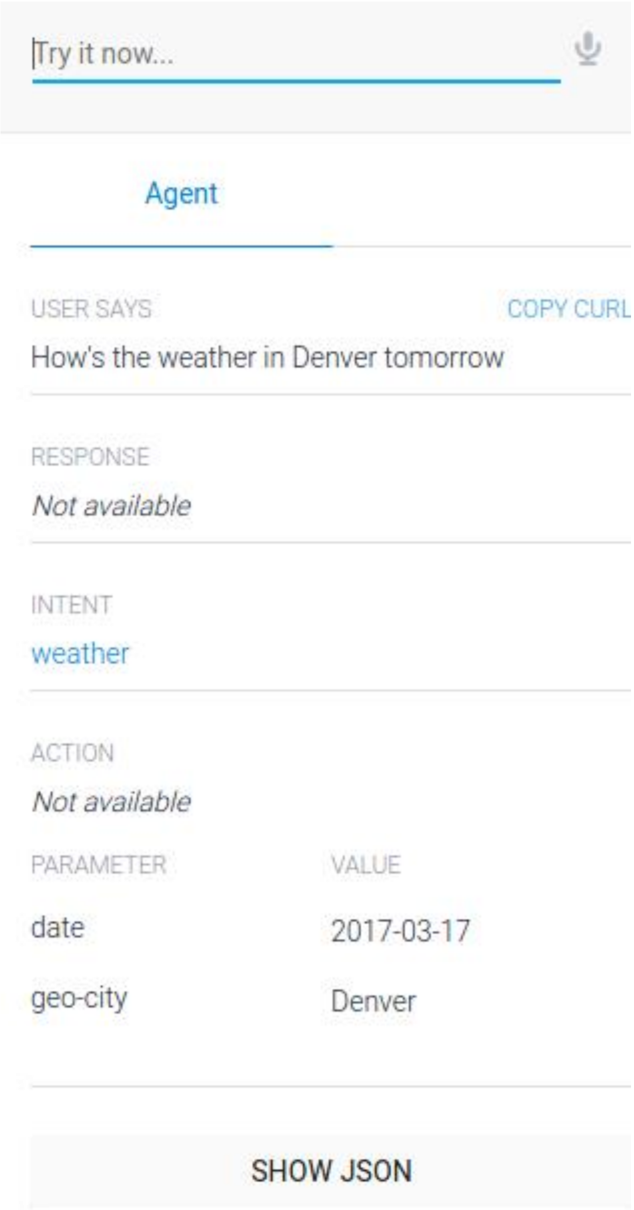
In the console on the right, type in a request. The request should be a little different than the examples you provided in the Training Phrases section. This can be something like “How’s the weather in Denver tomorrow”. After you type the request, hit “Enter/Return”.

Response – shows an appropriate response from the ones provided

The response chosen is based off of the values you provide in the query (e.g. By providing only the date, the agent should respond with the option that only includes the date)

Intent – weather again a successful trigger of the intent

Parameter – the values you provided in your query, should be reflected in the appropriate response



USER SAYS

How's the weather in Denver tomorrow

RESPONSE

Not available

INTENT

weather

ACTION

Not available

PARAMETER	VALUE
date	2017-03-17
geo-city	Denver

2. HTML

Hypertext Markup Language (**HTML**) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by *tags*, written using angle brackets. Tags such as <**img** /> and <**input** /> directly introduce content into the page. Other tags such as <**p**> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

HISTORY

Development

In 1980, physicist Tim Berners-Lee, a contractor at CERN, proposed and prototyped ENQUIRE, a system for CERN researchers to use and share documents. In 1989, Berners-Lee wrote a memo proposing an Internet-based hypertext system. Berners-Lee specified HTML and wrote the browser and server software in late 1990. That year, Berners-Lee and CERN data systems engineer Robert Cailliau collaborated on a joint request for funding, but the project was not formally adopted by CERN. In his personal notes from 1990 he listed "some of the many areas in which hypertext is used" and put an encyclopedia first.

The first publicly available description of HTML was a document called "HTML Tags", first mentioned on the Internet by Tim Berners-Lee in late 1991. It describes 18 elements comprising the initial, relatively simple design of HTML. Except for the hyperlink tag, these were strongly influenced by SGMLguid, an in-house Standard Generalized Markup Language (SGML)-based documentation format at CERN. Eleven of these elements still exist in HTML 4.

HTML is a markup language that web browsers use to interpret and compose text, images, and other material into visual or audible web pages. Default characteristics for every item of HTML markup are defined in the browser, and these characteristics can be altered or enhanced by the web page designer's additional use of CSS. Many of the text elements are found in the 1988 ISO technical report TR 9537 *Techniques for using SGML*, which in turn covers the features of early text formatting languages such as that used by the RUNOFF command developed in the early 1960s for the CTSS (Compatible Time-Sharing System) operating system: these formatting commands were derived from the commands used by typesetters to manually format documents. However, the SGML concept of generalized markup is based on elements (nested annotated ranges with attributes) rather than merely print effects, with also the separation of structure and markup; HTML has been progressively moved in this direction with CSS.

Berners-Lee considered HTML to be an application of SGML. It was formally defined as such by the Internet Engineering Task Force (IETF) with the mid-1993 publication of the first proposal for an HTML specification, the "Hypertext Markup Language (HTML)" Internet Draft by Berners-Lee and Dan Connolly, which included an SGML Document type definition to define the grammar. The draft expired after six months, but was notable for its acknowledgment of the NCSA Mosaic browser's custom tag for embedding in-line images, reflecting the IETF's philosophy of basing standards on successful prototypes. Similarly, Dave Raggett's competing Internet-Draft, "HTML+ (Hypertext Markup Format)", from late 1993, suggested standardizing already-implemented features like tables and fill-out forms.

After the HTML and HTML+ drafts expired in early 1994, the IETF created an HTML Working Group, which in 1995 completed "HTML 2.0", the first HTML specification intended to be treated as a standard against which future implementations should be based.

Further development under the auspices of the IETF was stalled by competing interests. Since 1996, the HTML specifications have been maintained, with input from commercial software vendors, by the World Wide Web Consortium (W3C). However, in 2000, HTML also became an international standard (ISO/IEC 15445:2000). HTML 4.01 was published in late 1999, with further errata published through 2001. In 2004, development began on HTML5 in the Web Hypertext Application Technology Working Group (WHATWG), which became a joint deliverable with the W3C in 2008, and completed and standardized on 28 October 2014.

HTML versions timeline

HTML 2

November 24, 1995

HTML 2.0 was published as RFC 1866. Supplemental RFCs added capabilities:

- November 25, 1995: RFC 1867 (form-based file upload)
- May 1996: RFC 1942 (tables)
- August 1996: RFC 1980 (client-side image maps)
- January 1997: RFC 2070 (internationalization)

HTML 3

January 14, 1997

HTML 3.2 was published as a W3C Recommendation. It was the first version developed and standardized exclusively by the W3C, as the IETF had closed its HTML Working Group on September 12, 1996.

Initially code-named "Wilbur", HTML 3.2 dropped math formulas entirely, reconciled overlap among various proprietary extensions and adopted most of Netscape's visual markup tags. Netscape's blink element and Microsoft's marquee element were omitted due to a mutual agreement between the two companies. A markup for mathematical formulas similar to that in HTML was not standardized until 14 months later in MathML.

HTML 4

December 18, 1997

HTML 4.0 was published as a W3C Recommendation. It offers three variations:

- Strict, in which deprecated elements are forbidden
- Transitional, in which deprecated elements are allowed
- Frameset, in which mostly only frame related elements are allowed.

Initially code-named "Cougar", HTML 4.0 adopted many browser-specific element types and attributes, but at the same time sought to phase out Netscape's visual markup features by marking them as deprecated in favor of style sheets. HTML 4 is an SGML application conforming to ISO 8879 – SGML.

April 24, 1998

HTML 4.0 was reissued with minor edits without incrementing the version number.

December 24, 1999

HTML 4.01 was published as a W3C Recommendation. It offers the same three variations as HTML 4.0 and its last errata were published on May 12, 2001.

May 2000

ISO/IEC 15445:2000 ("ISO HTML", based on HTML 4.01 Strict) was published as an ISO/IEC international standard. In the ISO this standard falls in the domain of the ISO/IEC JTC1/SC34 (ISO/IEC Joint Technical Committee 1, Subcommittee 34 – Document description and processing languages).

After HTML 4.01, there was no new version of HTML for many years as development of the parallel, XML-based language XHTML occupied the W3C's HTML Working Group through the early and mid-2000s.

HTML 5

October 28, 2014

HTML5 was published as a W3C Recommendation.

November 1, 2016

HTML 5.1 was published as a W3C Recommendation.

December 14, 2017

HTML 5.2 was published as a W3C Recommendation.

HTML draft version timeline

October 1991

HTML Tags, an informal CERN document listing 18 HTML tags, was first mentioned in public.

June 1992

First informal draft of the HTML DTD, with seven subsequent revisions (July 15, August 6, August 18, November 17, November 19, November 20, November 22)

November 1992

HTML DTD 1.1 (the first with a version number, based on RCS revisions, which start with 1.1 rather than 1.0), an informal draft

June 1993

Hypertext Markup Language was published by the IETF IIIR Working Group as an Internet Draft (a rough proposal for a standard). It was replaced by a second version one month later, followed by six further drafts published by IETF itself that finally led to HTML 2.0 in RFC 1866.

November 1993

HTML+ was published by the IETF as an Internet Draft and was a competing proposal to the Hypertext Markup Language draft. It expired in May 1994.

April 1995 (authored March 1995)

HTML 3.0 was proposed as a standard to the IETF, but the proposal expired five months later (28 September 1995) without further action. It included many of the capabilities that were in Raggett's HTML+ proposal, such as support for tables, text flow around figures and the display of complex mathematical formulas.

W3C began development of its own Arena browser as a test bed for HTML 3 and Cascading Style Sheets, but HTML 3.0 did not succeed for several reasons. The draft was considered very large at 150 pages and the pace of browser development, as well as the number of interested parties, had outstripped the resources of the IETF. Browser vendors, including Microsoft and Netscape at the time, chose to implement different subsets of HTML 3's draft features as well as to introduce their own extensions to it. (see Browser wars). These included extensions to control stylistic aspects of documents, contrary to the "belief [of the academic engineering community] that such things as text color, background texture, font size and font face were definitely outside the scope of a language when their only intent was to specify how a document would be organized." Dave Raggett, who has been a W3C Fellow for many years, has commented for example: "To a certain extent, Microsoft built its business on the Web by extending HTML features."

January 2008

HTML5 was published as a Working Draft by the W3C.

Although its syntax closely resembles that of SGML, HTML5 has abandoned any attempt to be an SGML application and has explicitly defined its own "html" serialization, in addition to an alternative XML-based XHTML5 serialization.

2011 HTML5 – Last Call

On 14 February 2011, the W3C extended the charter of its HTML Working Group with clear milestones for HTML5. In May 2011, the working group advanced HTML5 to "Last Call", an invitation to communities inside and outside W3C to confirm the technical soundness of the specification. The W3C developed a comprehensive test suite to achieve broad interoperability for the full specification by 2014, which was the target date for recommendation. In January 2011, the WHATWG renamed its "HTML5" living standard to "HTML". The W3C nevertheless continues its project to release HTML5.

2012 HTML5 – Candidate Recommendation

In July 2012, WHATWG and W3C decided on a degree of separation. W3C will continue the HTML5 specification work, focusing on a single definitive standard, which is considered as a "snapshot" by WHATWG. The WHATWG organization will continue its work with HTML5 as a "Living Standard". The concept of a living standard is that it is never complete and is always being updated and improved. New features can be added but functionality will not be removed.

In December 2012, W3C designated HTML5 as a Candidate Recommendation. The criterion for advancement to W3C Recommendation is "two 100% complete and fully interoperable implementations".

2014 HTML5 – Proposed Recommendation and Recommendation

In September 2014, W3C moved HTML5 to Proposed Recommendation.

On 28 October 2014, HTML5 was released as a stable W3C Recommendation, meaning the specification process is complete.

Markup

HTML markup consists of several key components, including those called *tags* (and their *attributes*), character-based *data types*, *character references* and *entity references*. HTML tags most commonly come in pairs like `<h1>` and `</h1>`, although some represent *empty elements* and so are unpaired, for example `<img>`. The first tag in such a pair is the *start tag*, and the second is the *end tag* (they are also called *opening tags* and *closing tags*).

Another important component is the HTML *document type declaration*, which triggers standards mode rendering.

The following is an example of the classic "Hello, World!" program:

```
<!DOCTYPE html>
<html>
 <head>
  <title>This is a title</title>
 </head>
 <body>
  <p>Hello world!</p>
 </body>
</html>
```

The text between `<html>` and `</html>` describes the web page, and the text between `<body>` and `</body>` is the visible page content. The markup text `<title>This is a title</title>` defines the browser page title.

The Document Type Declaration `<!DOCTYPE html>` is for HTML5. If a declaration is not included, various browsers will revert to "quirks mode" for rendering.

Elements

HTML documents imply a structure of nested HTML elements. These are indicated in the document by HTML *tags*, enclosed in angle brackets thus: `<p>`.

In the simple, general case, the extent of an element is indicated by a pair of tags: a "start tag" `<p>` and "end tag" `</p>`. The text content of the element, if any, is placed between these tags.

Tags may also enclose further tag markup between the start and end, including a mixture of tags and text. This indicates further (nested) elements, as children of the parent element.

The start tag may also include *attributes* within the tag. These indicate other information, such as identifiers for sections within the document, identifiers used to bind style information to the presentation of the document, and for some tags such as the `<img>` used to embed images, the reference to the image resource.

Some elements, such as the line break `<br>`, do not permit *any* embedded content, either text or further tags. These require only a single empty tag (akin to a start tag) and do not use an end tag.

Many tags, particularly the closing end tag for the very commonly used paragraph element `<p>`, are optional. An HTML browser or other agent can infer the closure for the end of an element from the context and the structural rules defined by the HTML standard. These rules are complex and not widely understood by most HTML coders.

The general form of an HTML element is therefore: `<tag attribute1="value1" attribute2="value2">"content"</tag>`. Some HTML elements are defined as *empty elements* and take the form `<tag attribute1="value1" attribute2="value2">`. Empty elements may enclose no content, for instance, the `<br>` tag or the inline `<img>` tag. The name of an HTML element is the name used in the tags. Note that the end tag's name is preceded by a slash character, `/`, and that in empty elements the end tag is neither required nor allowed. If attributes are not mentioned, default values are used in each case.

Element examples

Header of the HTML document: `<head>...</head>`. The title is included in the head, for example:

```
<head>
  <title>The Title</title>
</head>
```

Headings: HTML headings are defined with the `<h1>` to `<h6>` tags with H1 being the highest (or most important) level and H6 the least:

```
<h1>Heading level 1</h1>
<h2>Heading level 2</h2>
<h3>Heading level 3</h3>
<h4>Heading level 4</h4>
<h5>Heading level 5</h5>
<h6>Heading level 6</h6>
```

Paragraphs:

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

Line breaks: `<br>`. The difference between `<br>` and `<p>` is that `br` breaks a line without altering the semantic structure of the page, whereas `p` sections the page into paragraphs. Note also that `br` is an *empty element* in that, although it may have attributes, it can take no content and it may not have an end tag.

```
<p>This <br> is a paragraph <br> with <br> line breaks</p>
```

This is a link in HTML. To create a link the `<a>` tag is used. The `href` attribute holds the URL address of the link.

```
<a href="https://www.wikipedia.org/">A link to Wikipedia!</a>
```

Inputs:

There are many possible ways a user can give input/s like:

```
<input type="text" /> <!-- This is for text input -->
<input type="file" /> <!-- This is for uploading files -->
<input type="checkbox" /> <!-- This is for checkboxes -->
```

Comments:

```
<!-- This is a comment -->
```

Comments can help in the understanding of the markup and do not display in the webpage.

There are several types of markup elements used in HTML:

Structural markup indicates the purpose of text

For example, `<h2>Golf</h2>` establishes "Golf" as a second-level heading. Structural markup does not denote any specific rendering, but most web browsers have default styles for element formatting. Content may be further styled using Cascading Style Sheets (CSS).

Presentational markup indicates the appearance of the text, regardless of its purpose

For example, `<b>boldface</b>` indicates that visual output devices should render "boldface" in bold text, but gives little indication what devices that are unable to do this (such as aural devices that read the text aloud) should do. In the case of both `<b>bold</b>` and `<i>italic</i>`, there are other elements that may have equivalent visual renderings but that are more semantic in nature, such as `<strong>strong text</strong>` and `<em>emphasized text</em>` respectively. It is easier to see how an aural user agent should interpret the latter two elements. However, they are not equivalent to their presentational counterparts: it would be undesirable for a screen-reader to emphasize the name of a book, for instance, but on a screen such a name would be italicized. Most presentational markup elements have become deprecated under the HTML 4.0 specification in favor of using CSS for styling.

Hypertext markup makes parts of a document into links to other documents

An anchor element creates a hyperlink in the document and its `href` attribute sets the link's target URL. For example, the HTML markup `<a href="https://www.google.com/">Wikipedia</a>`, will render the word "Wikipedia" as a hyperlink. To render an image as a hyperlink, an `img` element is inserted as content into the `a` element. Like `br`, `img` is an empty element with attributes but no content or closing tag. `<a href="https://example.org"><img src="image.gif" alt="descriptive text" width="50" height="50" border="0"></a>`.

Attributes

Most of the attributes of an element are name-value pairs, separated by `=` and written within the start tag of an element after the element's name. The value may be enclosed in single or double quotes, although values consisting of certain characters can be left unquoted in HTML (but not XHTML). Leaving attribute values unquoted is considered unsafe. In contrast with name-value pair attributes, there are some attributes that affect the element simply by their presence in the start tag of the element, like the `ismap` attribute for the `img` element.

There are several common attributes that may appear in many elements :

- The `id` attribute provides a document-wide unique identifier for an element. This is used to identify the element so that stylesheets can alter its presentational properties, and scripts may alter, animate or delete its contents or presentation. Appended to the URL of the page, it provides a globally unique identifier for the element, typically a sub-section of the page. For example, the ID "Attributes" in `https://en.wikipedia.org/wiki/HTML#Attributes`.
- The `class` attribute provides a way of classifying similar elements. This can be used for semantic or presentation purposes. For example, an HTML document might semantically use the designation `<class="notation">` to indicate that all elements with this class value are subordinate to the main text of the document.

In presentation, such elements might be gathered together and presented as footnotes on a page instead of appearing in the place where they occur in the HTML source. Class attributes are used semantically in microformats. Multiple class values may be specified; for example `<class="notation important">` puts the element into both the `notation` and the `important` classes.

- An author may use the `style` attribute to assign presentational properties to a particular element. It is considered better practice to use an element's `id` or `class` attributes to select the element from within a stylesheet, though sometimes this can be too cumbersome for a simple, specific, or ad hoc styling.
- The `title` attribute is used to attach subtextual explanation to an element. In most browsers this attribute is displayed as a tooltip.
- The `lang` attribute identifies the natural language of the element's contents, which may be different from that of the rest of the document. For example, in an English-language document:

```
• <p>Oh well, <span lang="fr">c'est la vie</span>, as they say in France.</p>
```

The abbreviation element, `abbr`, can be used to demonstrate some of these attributes:

```
<abbr id="anId" class="jargon" style="color:purple;" title="Hypertext Markup
Language">HTML</abbr>
```

This example displays as HTML; in most browsers, pointing the cursor at the abbreviation should display the title text "Hypertext Markup Language."

Most elements take the language-related attribute `dir` to specify text direction, such as with "rtl" for right-to-left text in, for example, Arabic, Persian or Hebrew.

Character and entity references

As of version 4.0, HTML defines a set of 252 character entity references and a set of 1,114,050 numeric character references, both of which allow individual characters to be written via simple markup, rather than literally. A literal character and its markup counterpart are considered equivalent and are rendered identically.

The ability to "escape" characters in this way allows for the characters < and & (when written as &lt; and &amp;, respectively) to be interpreted as character data, rather than markup. For example, a literal < normally indicates the start of a tag, and & normally indicates the start of a character entity reference or numeric character reference; writing it as &amp; or &#x26; or &#38; allows & to be included in the content of an element or in the value of an attribute. The double-quote character ("), when not used to quote an attribute value, must also be escaped as &quot; or &#x22; or &#34; when it appears within the attribute value itself. Equivalently, the single-quote character ('), when not used to quote an attribute value, must also be escaped as &#x27; or &#39; (or as &apos; in HTML5 or XHTML documents) when it appears within the attribute value itself. If document authors overlook the need to escape such characters, some browsers can be very forgiving and try to use context to guess their intent. The result is still invalid markup, which makes the document less accessible to other browsers and to other user agents that may try to parse the document for search and indexing purposes for example.

Escaping also allows for characters that are not easily typed, or that are not available in the document's character encoding, to be represented within element and attribute content. For example, the acute-accented e (é), a character typically found only on Western European and South American keyboards, can be written in any HTML document as the entity reference &eacute; or as the numeric references &#xE9; or &#233;, using characters that are available on all keyboards and are supported in all character encodings. Unicode character encodings such as UTF-8 are compatible with all modern browsers and allow direct access to almost all the characters of the world's writing systems.

Data types

HTML defines several data types for element content, such as script data and stylesheet data, and a plethora of types for attribute values, including IDs, names, URIs, numbers, units of length, languages, media descriptors, colors, character encodings, dates and times, and so on. All of these data types are specializations of character data.

Document type declaration

HTML documents are required to start with a Document Type Declaration (informally, a "doctype"). In browsers, the doctype helps to define the rendering mode—particularly whether to use quirks mode.

The original purpose of the doctype was to enable parsing and validation of HTML documents by SGML tools based on the Document Type Definition (DTD). The DTD to which the DOCTYPE refers contains a machine-readable grammar specifying the permitted and prohibited content for a document conforming to such a DTD. Browsers, on the other hand, do not implement HTML as an application of SGML and by consequence do not read the DTD.

HTML5 does not define a DTD; therefore, in HTML5 the doctype declaration is simpler and shorter:[77]

```
<!DOCTYPE html>
```

An example of an HTML 4 doctype

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"https://www.w3.org/TR/html4/strict.dtd">
```

This declaration references the DTD for the "strict" version of HTML 4.01. SGML-based validators read the DTD in order to properly parse the document and to perform validation. In modern browsers, a valid doctype activates standards mode as opposed to quirks mode.

In addition, HTML 4.01 provides Transitional and Frameset DTDs, as explained below. Transitional type is the most inclusive, incorporating current tags as well as older or "deprecated" tags, with the Strict DTD excluding deprecated tags. Frameset has all tags necessary to make frames on a page along with the tags included in transitional type.

Semantic HTML

Semantic HTML is a way of writing HTML that emphasizes the meaning of the encoded information over its presentation (look). HTML has included semantic markup from its inception, but has also included presentational markup, such as `<font>`, `<i>` and `<center>` tags. There are also the semantically neutral span and div tags. Since the late 1990s, when Cascading Style Sheets were beginning to work in most browsers, web authors have been encouraged to avoid the use of presentational HTML markup with a view to the separation of presentation and content.

In a 2001 discussion of the Semantic Web, Tim Berners-Lee and others gave examples of ways in which intelligent software "agents" may one day automatically crawl the web and find, filter and correlate previously unrelated, published facts for the benefit of human users. Such agents are not commonplace even now, but some of the ideas of Web 2.0, mashups and price comparison websites may be coming close. The main difference between these web application hybrids and Berners-Lee's semantic agents lies in the fact that the current aggregation and hybridization of information is usually designed in by web developers, who already know the web locations and the API semantics of the specific data they wish to mash, compare and combine.

An important type of web agent that does crawl and read web pages automatically, without prior knowledge of what it might find, is the web crawler or search-engine spider. These software agents are dependent on the semantic clarity of web pages they find as they use various techniques and algorithms to read and index millions of web pages a day and provide web users with search facilities without which the World Wide Web's usefulness would be greatly reduced.

In order for search-engine spiders to be able to rate the significance of pieces of text they find in HTML documents, and also for those creating mashups and other hybrids as well as for more automated agents as they are developed, the semantic structures that exist in HTML need to be widely and uniformly applied to bring out the meaning of published text.

Presentational markup tags are deprecated in current HTML and XHTML recommendations. The majority of presentational features from previous versions of HTML are no longer allowed as they lead to poorer accessibility, higher cost of site maintenance, and larger document sizes.

Good semantic HTML also improves the accessibility of web documents (see also Web Content Accessibility Guidelines). For example, when a screen reader or audio browser can correctly ascertain the structure of a document, it will not waste the visually impaired user's time by reading out repeated or irrelevant information when it has been marked up correctly.

HTTP

The World Wide Web is composed primarily of HTML documents transmitted from web servers to web browsers using the Hypertext Transfer Protocol (HTTP). However, HTTP is used to serve images, sound, and other content, in addition to HTML. To allow the web browser to know how to handle each document it receives, other information is transmitted along with the document. This meta data usually includes the MIME type (e.g., text/html or application/xhtml+xml) and the character encoding (see Character encoding in HTML).

In modern browsers, the MIME type that is sent with the HTML document may affect how the document is initially interpreted. A document sent with the XHTML MIME type is expected to be well-formed XML; syntax errors may cause the browser to fail to render it. The same document sent with the HTML MIME type might be displayed successfully, since some browsers are more lenient with HTML.

The W3C recommendations state that XHTML 1.0 documents that follow guidelines set forth in the recommendation's Appendix C may be labeled with either MIME Type. XHTML 1.1 also states that XHTML 1.1 documents should[84] be labeled with either MIME type.

HTML e-mail

Most graphical email clients allow the use of a subset of HTML (often ill-defined) to provide formatting and semantic markup not available with plain text. This may include typographic information like coloured headings, emphasized and quoted text, inline images and diagrams. Many such clients include both a GUI editor for composing HTML e-mail messages and a rendering engine for displaying them. Use of HTML in e-mail is criticized by some because of compatibility issues, because it can help disguise phishing attacks, because of accessibility issues for blind or visually impaired people, because it can confuse spam filters and because the message size is larger than plain text.

Naming conventions

The most common filename extension for files containing HTML is .html. A common abbreviation of this is .htm, which originated because some early operating systems and file systems, such as DOS and the limitations imposed by FAT data structure, limited file extensions to three letters.

HTML Application

An HTML Application (HTA; file extension ".hta") is a Microsoft Windows application that uses HTML and Dynamic HTML in a browser to provide the application's graphical interface. A regular HTML file is confined to the security model of the web browser's security, communicating only to web servers and manipulating only web page objects and site cookies. An HTA runs as a fully trusted application and therefore has more privileges, like creation/editing/removal of files and Windows Registry entries. Because they operate outside the browser's security model, HTAs cannot be executed via HTTP, but must be downloaded (just like an EXE file) and executed from local file system.

HTML 4 Variations

Since its inception, HTML and its associated protocols gained acceptance relatively quickly. However, no clear standards existed in the early years of the language. Though its creators originally conceived of HTML as a semantic language devoid of presentation details, practical uses pushed many presentational elements and attributes into the language, driven largely by the various browser vendors. The latest standards surrounding HTML reflect efforts to overcome the sometimes chaotic development of the language and to create a rational foundation for building both meaningful and well-presented documents. To return HTML to its role as a semantic language, the W3C has developed style languages such as CSS and XSL to shoulder the burden of presentation. In conjunction, the HTML specification has slowly reined in the presentational elements.

There are two axes differentiating various variations of HTML as currently specified: SGML-based HTML versus XML-based HTML (referred to as XHTML) on one axis, and strict versus transitional (loose) versus frameset on the other axis.

SGML-based versus XML-based HTML

One difference in the latest HTML specifications lies in the distinction between the SGML-based specification and the XML-based specification. The XML-based specification is usually called XHTML to distinguish it clearly from the more traditional definition. However, the root element name continues to be "html" even in the XHTML-specified HTML. The W3C intended XHTML 1.0 to be identical to HTML 4.01 except where limitations of XML over the more complex SGML require workarounds. Because XHTML and HTML are closely related, they are sometimes documented in parallel. In such circumstances, some authors conflate the two names as (X)HTML or X(HTML).

Like HTML 4.01, XHTML 1.0 has three sub-specifications: strict, transitional and frameset.

Aside from the different opening declarations for a document, the differences between an HTML 4.01 and XHTML 1.0 document—in each of the corresponding DTDs—are largely syntactic. The underlying syntax of HTML allows many shortcuts that XHTML does not, such as elements with optional opening or closing tags, and even empty elements which must not have an end tag. By contrast, XHTML requires all elements to have an opening tag and a closing tag. XHTML, however, also introduces a new shortcut: an XHTML tag may be opened and closed within the same tag, by including a slash before the end of the tag like this: `<br/>`. The introduction of this shorthand, which is not used in the SGML declaration for HTML 4.01, may confuse earlier software unfamiliar with this new convention. A fix for this is to include a space before closing the tag, as such: `<br />`.[89]

To understand the subtle differences between HTML and XHTML, consider the transformation of a valid and well-formed XHTML 1.0 document that adheres to Appendix C (see below) into a valid HTML 4.01 document. To make this translation requires the following steps:

1. **The language for an element should be specified with a `lang` attribute rather than the XHTML `xml:lang` attribute.** XHTML uses XML's built in language-defining functionality attribute.
2. **Remove the XML namespace (`xmlns=URI`).** HTML has no facilities for namespaces.
3. **Change the document type declaration** from XHTML 1.0 to HTML 4.01. (see DTD section for further explanation).
4. If present, **remove the XML declaration.** (Typically this is: `<?xml version="1.0" encoding="utf-8"?>`).
5. **Ensure that the document's MIME type is set to `text/html`.** For both HTML and XHTML, this comes from the HTTP `Content-Type` header sent by the server.
6. **Change the XML empty-element syntax to an HTML style empty element** (`<br />` to `<br>`).

Those are the main changes necessary to translate a document from XHTML 1.0 to HTML 4.01. To translate from HTML to XHTML would also require the addition of any omitted opening or closing tags. Whether coding in HTML or XHTML it may just be best to always include the optional tags within an HTML document rather than remembering which tags can be omitted.

A well-formed XHTML document adheres to all the syntax requirements of XML. A valid document adheres to the content specification for XHTML, which describes the document structure.

The W3C recommends several conventions to ensure an easy migration between HTML and XHTML (see HTML Compatibility Guidelines). The following steps can be applied to XHTML 1.0 documents only:

- Include both `xml:lang` and `lang` attributes on any elements assigning language.
- Use the empty-element syntax only for elements specified as empty in HTML.
- Include an extra space in empty-element tags: for example `<br />` instead of `<br>`.
- Include explicit close tags for elements that permit content but are left empty (for example, `<div></div>`, not `<div />`).
- Omit the XML declaration.

By carefully following the W3C's compatibility guidelines, a user agent should be able to interpret the document equally as HTML or XHTML. For documents that are XHTML 1.0 and have been made compatible in this way, the W3C permits them to be served either as HTML (with a `text/html` MIME type), or as XHTML (with an `application/xhtml+xml` or `application/xml` MIME type). When delivered as XHTML, browsers should use an XML parser, which adheres strictly to the XML specifications for parsing the document's contents.

Transitional versus strict

HTML 4 defined three different versions of the language: Strict, Transitional (once called Loose) and Frameset. The Strict version is intended for new documents and is considered best practice, while the Transitional and Frameset versions were developed to make it easier to transition documents that conformed to older HTML specification or didn't conform to any specification to a version of HTML 4. The Transitional and Frameset versions allow for presentational markup, which is omitted in the Strict version. Instead, cascading style sheets are encouraged to improve the presentation of HTML documents. Because XHTML 1 only defines an XML syntax for the language defined by HTML 4, the same differences apply to XHTML 1 as well.

The Transitional version allows the following parts of the vocabulary, which are not included in the Strict version:

- **A looser content model**
  - Inline elements and plain text are allowed directly in: body, blockquote, form, noscript and noframes
- **Presentation related elements**
  - underline (u)(Deprecated. can confuse a visitor with a hyperlink.)
  - strike-through (s)
  - center (Deprecated. use CSS instead.)
  - font (Deprecated. use CSS instead.)
  - basefont (Deprecated. use CSS instead.)
- **Presentation related attributes**
  - background (Deprecated. use CSS instead.) and bgcolor (Deprecated. use CSS instead.) attributes for body (required element according to the W3C.) element.
  - align (Deprecated. use CSS instead.) attribute on div, form, paragraph (p) and heading (h1...h6) elements
  - align (Deprecated. use CSS instead.), noshade (Deprecated. use CSS instead.), size (Deprecated. use CSS instead.) and width (Deprecated. use CSS instead.) attributes on hr element
  - align (Deprecated. use CSS instead.), border, vspace and hspace attributes on img and object (caution: the object element is only supported in Internet Explorer (from the major browsers)) elements
  - align (Deprecated. use CSS instead.) attribute on legend and caption elements
  - align (Deprecated. use CSS instead.) and bgcolor (Deprecated. use CSS instead.) on table element
  - nowrap (Obsolete), bgcolor (Deprecated. use CSS instead.), width, height on td and th elements
  - bgcolor (Deprecated. use CSS instead.) attribute on tr element

    - clear (Obsolete) attribute on br element
    - compact attribute on dl, dir and menu elements
    - type (Deprecated. use CSS instead.), compact (Deprecated. use CSS instead.) and start (Deprecated. use CSS instead.) attributes on ol and ul elements
    - type and value attributes on li element
    - width attribute on pre element
- **Additional elements in Transitional specification**
    - menu (Deprecated. use CSS instead.) list (no substitute, though unordered list is recommended)
    - dir (Deprecated. use CSS instead.) list (no substitute, though unordered list is recommended)
    - isindex (Deprecated.) (element requires server-side support and is typically added to documents server-side, form and input elements can be used as a substitute)
    - applet (Deprecated. use the object element instead.)
- **The language (Obsolete) attribute on script element** (redundant with the type attribute).
- **Frame related entities**
    - iframe
    - noframes
    - target (Deprecated in the map, link and form elements.) attribute on a, client-side image-map (map), link, form and base elements

The Frameset version includes everything in the Transitional version, as well as the frameset element (used instead of body) and the frame element.

Frameset versus transitional

In addition to the above transitional differences, the frameset specifications (whether XHTML 1.0 or HTML 4.01) specify a different content model, with frameset replacing body, that contains either frame elements, or optionally noframes with a body.

Summary of specification versions

As this list demonstrates, the loose versions of the specification are maintained for legacy support. However, contrary to popular misconceptions, the move to XHTML does not imply a removal of this legacy support. Rather the X in XML stands for extensible and the W3C is modularizing the entire specification and opening it up to independent extensions. The primary achievement in the move from XHTML 1.0 to XHTML 1.1 is the modularization of the entire specification. The strict version of HTML is deployed in XHTML 1.1 through a set of modular extensions to the base XHTML 1.1 specification. Likewise, someone looking for the loose (transitional) or frameset specifications will find similar extended XHTML 1.1 support (much of it is contained in the legacy or frame modules). The modularization also allows for separate features to develop on their own timetable. So for example, XHTML 1.1 will allow quicker migration to emerging XML standards such as MathML (a presentational and semantic math language based on XML) and XForms—a new highly advanced web-form technology to replace the existing HTML forms.

In summary, the HTML 4 specification primarily reined in all the various HTML implementations into a single clearly written specification based on SGML. XHTML 1.0, ported this specification, as is, to the new XML defined specification. Next, XHTML 1.1 takes advantage of the extensible nature of XML and modularizes the whole specification. XHTML 2.0 was intended to be the first step in adding new features to the specification in a standards-body-based approach.

3. CSS

Cascading Style Sheets (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name *cascading* comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

In addition to HTML, other markup languages support the use of CSS including XHTML, plain XML, SVG, and XUL.

CSS has a simple syntax and uses a number of English keywords to specify the names of various style properties.

A style sheet consists of a list of *rules*. Each rule or rule-set consists of one or more *selectors*, and a *declaration block*.

Selector

In CSS, *selectors* declare which part of the markup a style applies to by matching tags and attributes in the markup itself.

Selectors may apply to the following:

- all elements of a specific type, e.g. the second-level headers h2
- elements specified by attribute, in particular:
    - *id*: an identifier unique within the document
    - *class*: an identifier that can annotate multiple elements in a document
- elements depending on how they are placed relative to others in the document tree.

Classes and IDs are case-sensitive, start with letters, and can include alphanumeric characters, hyphens and underscores. A class may apply to any number of instances of any elements. An ID may only be applied to a single element.

Pseudo-classes are used in CSS selectors to permit formatting based on information that is not contained in the document tree. One example of a widely used pseudo-class is `:hover`, which identifies content only when the user "points to" the visible element, usually by holding the mouse cursor over it. It is appended to a selector as in `a:hover` or `#elementid:hover`. A pseudo-class classifies document elements, such as `:link` or `:visited`, whereas a *pseudo-element* makes a selection that may consist of partial elements, such as `::first-line` or `::first-letter`.[6]

Selectors may be combined in many ways to achieve great specificity and flexibility. Multiple selectors may be joined in a spaced list to specify elements by location, element type, id, class, or any combination thereof. The order of the selectors is important. For example, `div .myClass {color: red;}` applies to all elements of class myClass that are inside div elements, whereas `.myClass div {color: red;}` applies to all div elements that are in elements of class myClass.

The following table provides a summary of selector syntax indicating usage and the version of CSS that introduced it.

Pattern	Matches	First defined in CSS level
`E`	an element of type E	1
`E:link`	an E element is the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited)	1

`E:active`	an E element during certain user actions	1
`E::first-line`	the first formatted line of an E element	1
`E::first-letter`	the first formatted letter of an E element	1
`.c`	all elements with class="c"	1
`#myid`	the element with id="myid"	1
`E.warning`	an E element whose class is "warning" (the document language specifies how class is determined)	1
`E#myid`	an E element with ID equal to "myid"	1
`E F`	an F element descendant of an E element	1
`*`	any element	2
`E[foo]`	an E element with a "foo" attribute	2
`E[foo="bar"]`	an E element whose "foo" attribute value is exactly equal to "bar"	2
`E[foo~="bar"]`	an E element whose "foo" attribute value is a list of whitespace-separated values, one of which is exactly equal to "bar"	2

`E[foo|="en"]`	an E element whose "foo" attribute has a hyphen-separated list of values beginning (from the left) with "en"	2
`E:first-child`	an E element, first child of its parent	2
`E:lang(fr)`	an element of type E in language "fr" (the document language specifies how language is determined)	2
`E::before`	generated content before an E element's content	2
`E::after`	generated content after an E element's content	2
`E > F`	an F element child of an E element	2
`E + F`	an F element immediately preceded by an E element	2
`E[foo^="bar"]`	an E element whose "foo" attribute value begins exactly with the string "bar"	3
`E[foo$="bar"]`	an E element whose "foo" attribute value ends exactly with the string "bar"	3
`E[foo*="bar"]`	an E element whose "foo" attribute value contains the substring "bar"	3
`E:root`	an E element, root of the document	3
`E:nth-child(n)`	an E element, the n-th child of its parent	3

`E:nth-last-child(n)`	an E element, the n-th child of its parent, counting from the last one	3
`E:nth-of-type(n)`	an E element, the n-th sibling of its type	3
`E:nth-last-of-type(n)`	an E element, the n-th sibling of its type, counting from the last one	3
`E:last-child`	an E element, last child of its parent	3
`E:first-of-type`	an E element, first sibling of its type	3
`E:last-of-type`	an E element, last sibling of its type	3
`E:only-child`	an E element, only child of its parent	3
`E:only-of-type`	an E element, only sibling of its type	3
`E:empty`	an E element that has no children (including text nodes)	3
`E:target`	an E element being the target of the referring URI	3
`E:enabled`	a user interface element E that is enabled	3
`E:disabled`	a user interface element E that is disabled	3
`E:checked`	a user interface element E that is checked (for instance a radio-button or checkbox)	3

`E:not(s)`	an E element that does not match simple selector s	3
`E ~ F`	an F element preceded by an E element	3

Declaration block

A declaration block consists of a list of *declarations* in braces. Each declaration itself consists of a *property*, a colon (`:`), and a *value*. If there are multiple declarations in a block, a semi-colon (`;`) must be inserted to separate each declaration.

Properties are specified in the CSS standard. Each property has a set of possible values. Some properties can affect any type of element, and others apply only to particular groups of elements.

Values may be keywords, such as "center" or "inherit", or numerical values, such as 200px (200 pixels), 50vw (50 percent of the viewport width) or 80% (80 percent of the parent element's width). Color values can be specified with keywords (e.g. "red"), hexadecimal values (e.g. #FF0000, also abbreviated as #F00), RGB values on a 0 to 255 scale (e.g. `rgb(255, 0, 0)`), RGBA values that specify both color and alpha transparency (e.g. `rgba(255, 0, 0, 0.8)`), or HSL or HSLA values (e.g. `hsl(000, 100%, 50%)`, `hsla(000, 100%, 50%, 80%)`).

Length units

Non-zero numeric values representing linear measures must include a length unit, which is either an alphabetic code or abbreviation, as in `200px` or `50vw`; or a percentage sign, as in `80%`. Some units – `cm` (centimetre); `in` (inch); `mm` (millimetre); `pc` (pica); and `pt` (point) – are *absolute*, which means that the rendered dimension does not depend upon the structure of the page; others – `em` (em); `ex` (ex) and `px` (pixel) – are *relative*, which means that factors such as the font size of a parent element can affect the rendered measurement. These eight units were a feature of CSS 1 and retained in all subsequent revisions. The proposed CSS Values and Units Module Level 3 will, if adopted as a

W3C Recommendation, provide seven further length units: `ch`; `Q`; `rem`; `vh`; `vmax`; `vmin`; and `vw`.

Use

Before CSS, nearly all presentational attributes of HTML documents were contained within the HTML markup. All font colors, background styles, element alignments, borders and sizes had to be explicitly described, often repeatedly, within the HTML. CSS lets authors move much of that information to another file, the style sheet, resulting in considerably simpler HTML.

For example, headings (`h1` elements), sub-headings (`h2`), sub-sub-headings (`h3`), etc., are defined structurally using HTML. In print and on the screen, choice of font, size, color and emphasis for these elements is *presentational*.

Before CSS, document authors who wanted to assign such typographic characteristics to, say, all `h2` headings had to repeat HTML presentational markup for each occurrence of that heading type. This made documents more complex, larger, and more error-prone and difficult to maintain. CSS allows the separation of presentation from structure. CSS can define color, font, text alignment, size, borders, spacing, layout and many other typographic characteristics, and can do so independently for on-screen and printed views. CSS also defines non-visual styles, such as reading speed and emphasis for aural text readers. The W3C has now deprecated the use of all presentational HTML markup.

For example, under pre-CSS HTML, a heading element defined with red text would be written as:

```
<h1><font color="red"> Chapter 1. </font></h1>
```

Using CSS, the same element can be coded using style properties instead of HTML presentational attributes:

```
<h1 style="color: red;"> Chapter 1. </h1>
```

The advantages of this may not be immediately clear (since the second form is actually more verbose), but the power of CSS becomes more apparent when the style properties are placed in an internal style element or, even better, an external CSS file. For example, suppose the document contains the style element:

```
<style>
h1 {
   color: red;
}
</style>
```

All `h1` elements in the document will then automatically become red without requiring any explicit code. If the author later wanted to make `h1` elements blue instead, this could be done by changing the style element to:

```
<style>
h1 {
   color: blue;
}
</style>
```

rather than by laboriously going through the document and changing the color for each individual `h1` element.

The styles can also be placed in an external CSS file, as described below, and loaded using syntax similar to:

```
<link href="path/to/file.css" rel="stylesheet" type="text/css">
```

This further decouples the styling from the HTML document, and makes it possible to restyle multiple documents by simply editing a shared external CSS file.

4. React

React (also known as **React.js** or **ReactJS**) is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies.

React can be used as a base in the development of single-page or mobile applications, as it is optimal for fetching rapidly changing data that needs to be recorded. However, fetching data is only the beginning of what happens on a web page, which is why complex React applications usually require the use of additional libraries for state management, routing, and interaction with an API: Next.js and Gatsby.js are examples of such libraries.

Basic Usage

The following is a rudimentary example of React usage in HTML with JSX and JavaScript.

```
<div id="myReactApp"></div>

<script type="text/babel">
  class Greeter extends React.Component {
    render() {
      return <h1>{this.props.greeting}</h1>
    }
  }

  ReactDOM.render(<Greeter greeting="Hello World!" />,
document.getElementById('myReactApp'));
</script>
```

The `Greeter` class is a React component that accepts a property `greeting`.

The `ReactDOM.render` method creates an instance of the `Greeter` component, sets the `greeting` property to `'Hello World'` and inserts the rendered component as a child element to the DOM element with id `myReactApp`.

When displayed in a web browser the result will be

```
<div id="myReactApp">
  <h1>Hello World!</h1>
</div>
```

Components

React code is made of entities called components. Components can be rendered to a particular element in the DOM using the React DOM library. When rendering a component, one can pass in values that are known as "props":

```
ReactDOM.render(<Greeter greeting="Hello World!" />,
document.getElementById('myReactApp'));
```

The two primary ways of declaring components in React is via functional components and class-based components.

Functional components

Functional components are declared with a function that then returns some JSX.

```
function Greeting(props) {
  return <div>Hello, {props.name}!</div>;
}
```

Class-based components

Class-based components are declared using ES6 classes. They are also known as "stateful" components, because their state can hold values throughout the component and can be passed to child components through props:

```
class ParentComponent extends React.Component {
  state = { color: 'green' };
  render() {
    return (
      <ChildComponent color={this.state.color} />
    );
  }
}
```

Virtual DOM

Another notable feature is the use of a virtual Document Object Model, or virtual DOM. React creates an in-memory data-structure cache, computes the resulting differences, and then updates the browser's displayed DOM efficiently. This allows the programmer to write code as if the entire page is rendered on each change, while the React libraries only render subcomponents that actually change.

Lifecycle methods

Lifecycle methods are hooks that allow execution of code at set points during a component's lifetime.

- `shouldComponentUpdate` allows the developer to prevent unnecessary re-rendering of a component by returning false if a render is not required.
- `componentDidMount` is called once the component has "mounted" (the component has been created in the user interface, often by associating it with a DOM node). This is commonly used to trigger data loading from a remote source via an API.
- `componentWillUnmount` is called immediately before the component is torn down or "unmounted". This is commonly used to clear resource demanding dependencies to the component that will not simply be removed with the unmounting of the component (e.g., removing any `setInterval()` instances that are related to the component, or an "eventListener" set on the "document" because of the presence of the component)
- `render` is the most important lifecycle method and the only required one in any component. It is usually called every time the component's state is updated, which should be reflected in the user interface.

JSX

JSX, or JavaScript XML , is an extension to the JavaScript language syntax. Similar in appearance to HTML, JSX provides a way to structure component rendering using syntax familiar to many developers. React components are typically written using JSX, although they do not have to be (components may also be written in pure JavaScript). JSX is similar to another extension syntax created by Facebook for PHP called XHP.

An example of JSX code:

```
class App extends React.Component {
  render() {
    return (
      <div>
        <p>Header</p>
        <p>Content</p>
        <p>Footer</p>
      </div>
    );
  }
}
```

Nested elements

Multiple elements on the same level need to be wrapped in a single container element such as the `<div>` element shown above, or returned as an array.

Attributes

JSX provides a range of element attributes designed to mirror those provided by HTML. Custom attributes can also be passed to the component. All attributes will be received by the component as props.

JavaScript expressions

JavaScript expressions (but not statements) can be used inside JSX with curly brackets `{}`:

```
<h1>{10+1}</h1>
```

The example above will render

```
<h1>11</h1>
```

Conditional statements

If–else statements cannot be used inside JSX but conditional expressions can be used instead. The example below will render `{ i === 1 ? 'true' : 'false' }` as the string `'true'` because `i` is equal to 1.

```
class App extends React.Component {
  render() {
    const i = 1;
    return (
      <div>
        <h1>{ i === 1 ? 'true' : 'false' }</h1>
      </div>
    );
  }
}
```

The above will render:

```
<div>
  <h1>true</h1>
</div>
```

Functions and JSX can be used in conditionals:

```
class App extends React.Component {
  render() {
    const sections = [1, 2, 3];
    return (
      <div>
        {sections.length > 0 && sections.map(n => (
            /* 'key' is used by react to keep track of list items and their changes */
            /* Each 'key' must be unique */
            <div key={"section-" + n}>Section {n}</div>
        ))}
      </div>
    );
  }
}
```

The above will render:

```
<div>
  <div>Section 1</div>
  <div>Section 2</div>
  <div>Section 3</div>
</div>
```

Code written in JSX requires conversion with a tool such as Babel before it can be understood by web browsers. This processing is generally performed during a software build process before the application is deployed.

React Hooks

Hooks are functions that let developers "hook into" React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes.

React provides a few built-in Hooks like `useState`. You can also create your own Hooks to reuse stateful behavior between different components.

Use of the Flux architecture

To support React's concept of unidirectional data flow (which might be contrasted with AngularJS's bidirectional flow), the Flux architecture represents an alternative to the popular model-view-controller architecture. Flux features *actions* which are sent through a central *dispatcher* to a *store*, and changes to the store are propagated back to the view. When used with React, this propagation is accomplished through component properties.

Flux can be considered a variant of the observer pattern.

A React component under the Flux architecture should not directly modify any props passed to it, but should be passed callback functions that create *actions* which are sent by the dispatcher to modify the store. The action is an object whose responsibility is to describe what has taken place: for example, an action describing one user "following" another might contain a user id, a target user id, and the type `USER_FOLLOWED_ANOTHER_USER`. The stores, which can be thought of as models, can alter themselves in response to actions received from the dispatcher.

This pattern is sometimes expressed as "properties flow down, actions flow up". Many implementations of Flux have been created since its inception, perhaps the most well-known being Redux, which features a single store, often called a single source of truth.

History

React was created by Jordan Walke, a software engineer at Facebook, who released an early prototype of React called "FaxJS". He was influenced by XHP, an HTML component framework for PHP. It was first deployed on Facebook's News Feed in 2011 and later on Instagram in 2012. It was open-sourced at JSConf US in May 2013.

React Native, which enables native Android, iOS, and UWP development with React, was announced at Facebook's React Conf in February 2015 and open-sourced in March 2015.

On April 18, 2017, Facebook announced React Fiber, a new core algorithm of React framework library for building user interfaces. React Fiber was to become the foundation of any future improvements and feature development of the React framework.

On September 26, 2017, React 16.0 was released to the public.

On February 16, 2019, React 16.8 was released to the public. The release introduced React Hooks.

CHAPTER – 3

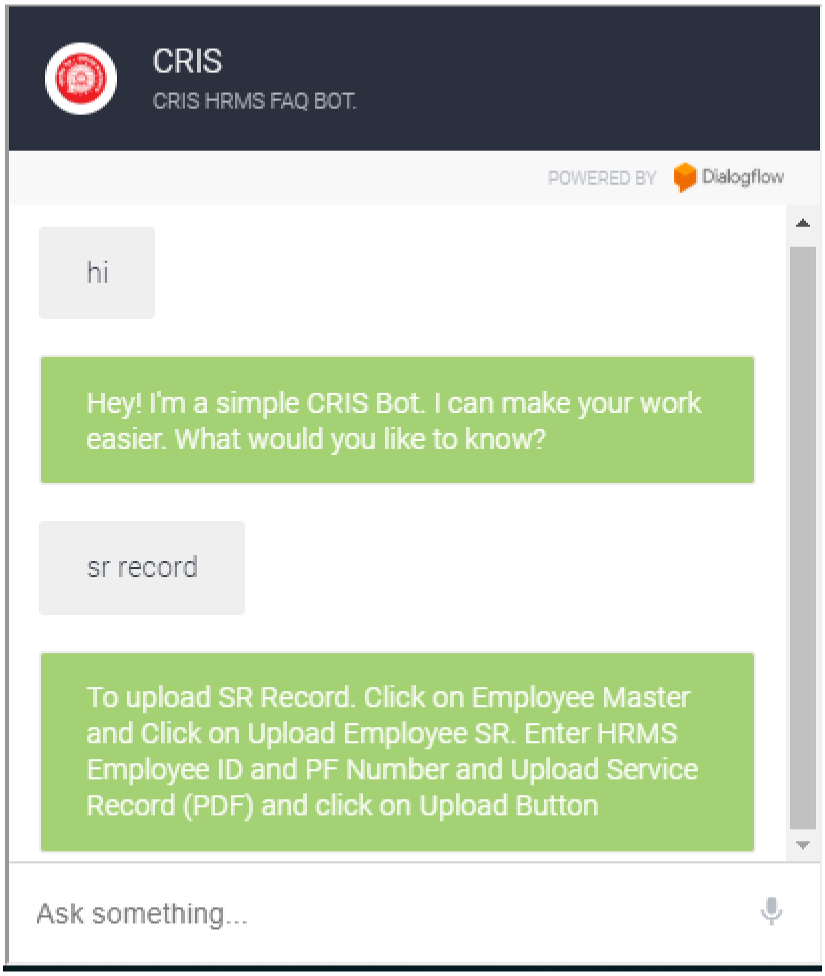
TECHNICAL CONTENTS

1. Architecture

• Front End System

The front end system of the CRIS (Centre for Railways Information System) Chat-bot will be having HTML and CSS or React. The will be having a message box where they can type their message and also will have a list of all previous messages and their replies.

When you open the website you will see a portion of website covered by a Chat-bot. The top part of the Chat-bot window shows the logo and the name of the Chat-bot. We can type anything in that box with which the dialogflow will search the message with the intents and will try to match it with any one the intents. If the message doesn't match any of the intents, it will show a message like "Please contact HRMS department for queries."



hi

Hey! I'm a simple CRIS Bot. I can make your work easier. What would you like to know?

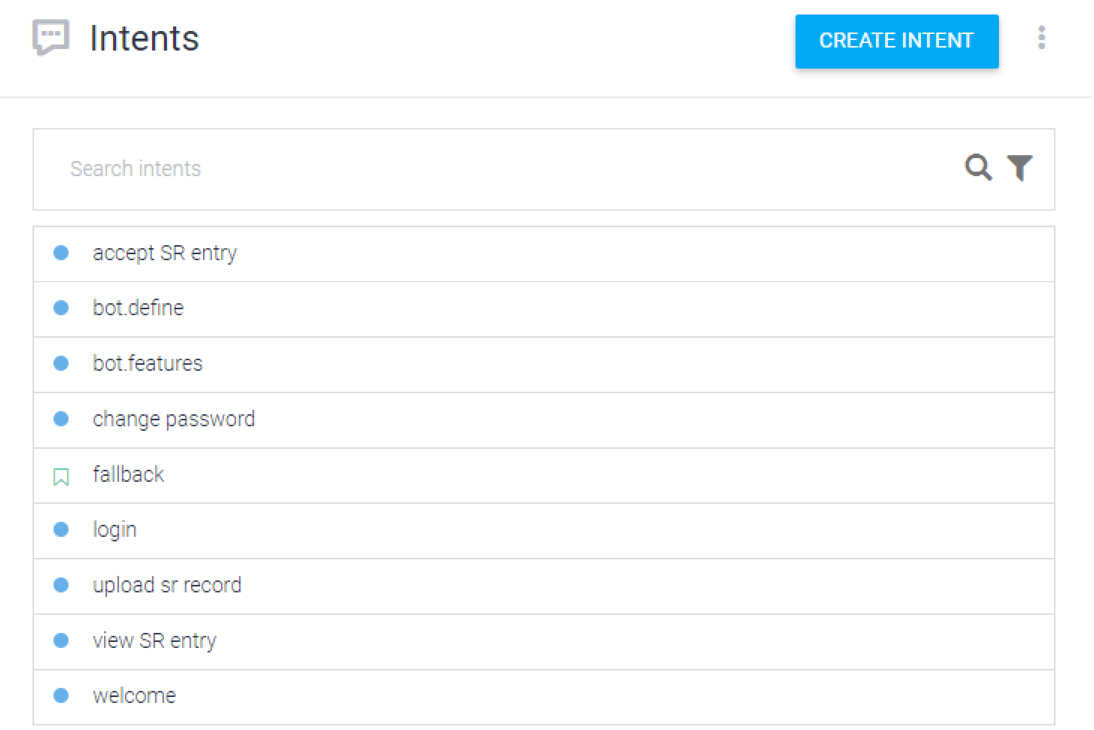
sr record

To upload SR Record. Click on Employee Master and Click on Upload Employee SR. Enter HRMS Employee ID and PF Number and Upload Service Record (PDF) and click on Upload Button

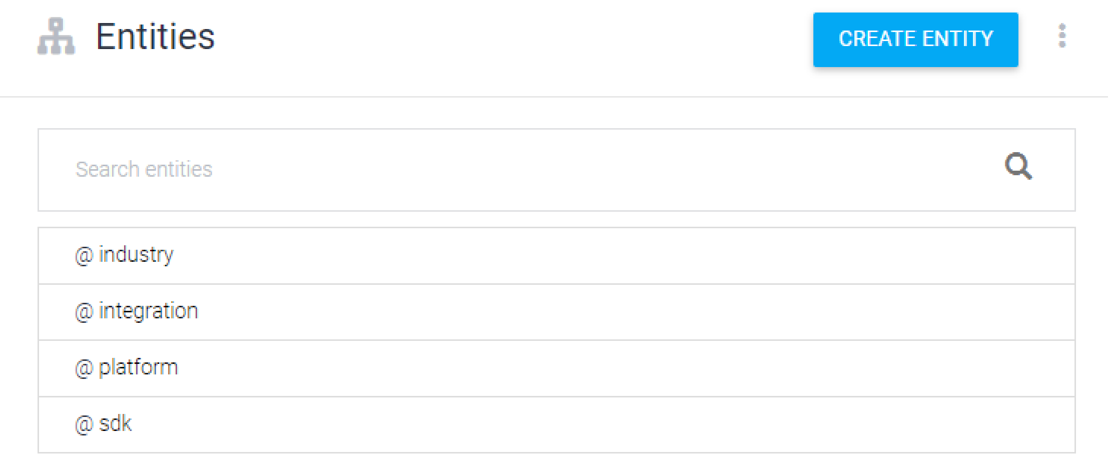
- **Back End System**

The backend system of our program will be totally handled by the dialogfllow. It will create a table for himself containing all the intents and entities thoroughly explained.

The backend system will be connected to the frontend by Dialogflow API or by Webhoook.

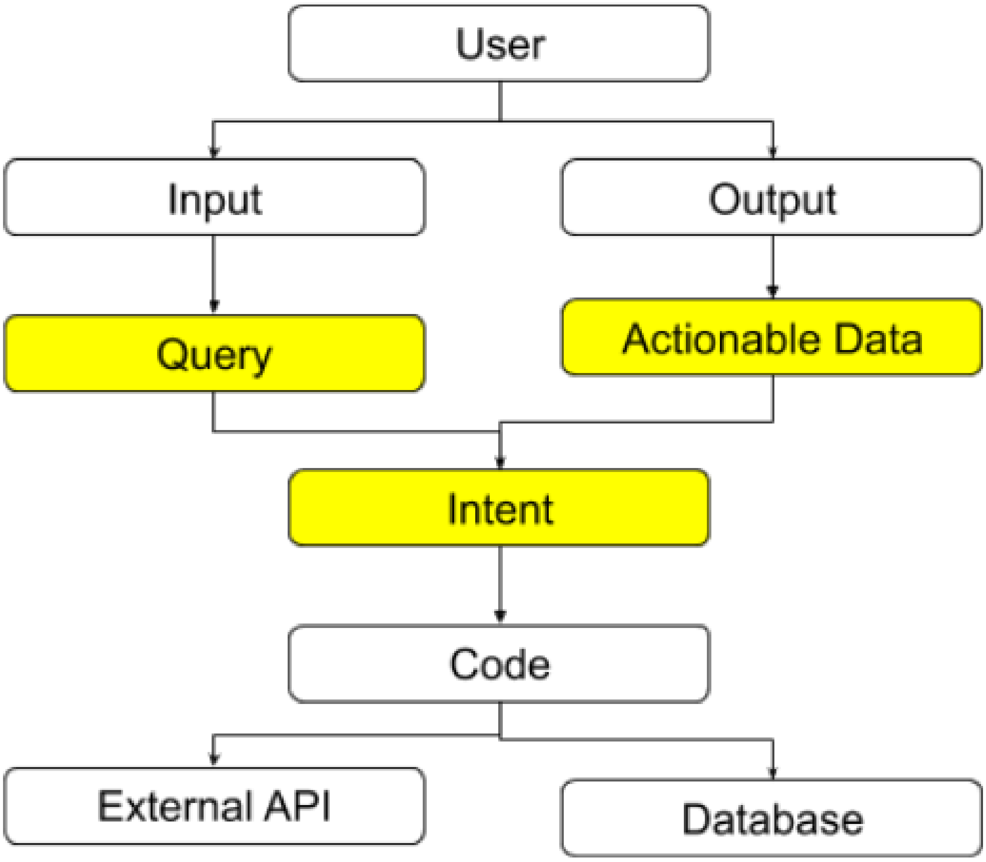


- accept SR entry
- bot.define
- bot.features
- change password
- fallback
- login
- upload sr record
- view SR entry
- welcome



- @ industry
- @ integration
- @ platform
- @ sdk

- **Connection between Front-end and Back-end**



ADVANTAGES OF CHATBOTS

Humans can serve a limited number of clients at the same time. This restriction does not exist for chatbots, and they can manage all necessary queries simultaneously.

Their main advantages are:

- **Reduced costs:** Chatbots eliminate the need for labor during online interaction with customers. This is obviously a great advantage for companies that receive multiple queries at once. In addition to saving costs with them, companies can align the chatbot with their objectives, and use them as a means to enhance customer conversion.
- **24/7 Availability:** Unlike humans, once we install a chatbot, it can handle queries at any time of day. Thus, the customer does not have to wait for a commercial of the company to help him. This also allows companies to monitor customer "traffic" during non-working hours and contact them later.
- **Learning and updating:** AI-based chatbots are able to learn from interactions and update independently. This is one of the main advantages. When you hire a new employee, you have to train them continuously. However, chatbots "form" themselves (with certain limitations, of course).
- **Management of multiple clients:** Humans can serve a limited number of customers at the same time. This restriction does not exist for chatbots, and they can manage all the necessary queries simultaneously. This is one of the main advantages of using chatbot, as no customer is left unattended and you are solving different problems at the same time. There are chatbots companies already working on developing voice chatbot services.

DISADVANTAGES OF CHATBOTS

- **Complex interface:** It is often considered that chatbots are complicated and need a lot of time to understand what you want in customer. Sometimes, it can also annoy the client about their slowness, or their difficulty in filtering responses.
- **They don't get you right:** Fixed chatbots can get stuck easily. If a query doesn't relate to something you've previously "taught" it, you won't understand it. This can lead to a frustrated customer and the loss of the sale. Other times they do understand you, but they need double (or triple) as many messages as one person, which spoils the user experience.

- **Time-consuming:** Chatbots are installed with the aim of speeding up responses and improving customer interaction. However, due to the limited availability of data and the time needed for self-updating, this process can be slow and costly. Therefore, there are times when instead of serving several customers at once, chatbots may become confused and not serve the customer well.
- **Installation cost:** Chatbots are useful programs that help you save a lot of labor by ensuring availability at all times and serving several customers at once. But unlike humans, each chatbot needs to be programmed differently for each business, which increases the initial installation cost. Considering the last-minute changes that can always occur, this is a risky investment, as updating the program will generate additional costs.
- **Null decision making:** Chatbots can attack the nerves of more than one because they are not able to make decisions.

  This can lead to problems. For example: Microsoft launched a chatbot for Twitter. In less than 24 hours, the content it received from users turned it into a racist and misogynistic account.

  It is very important that your chatbot is well optimized, so that it does not end up being a disaster like this.

- **Bad memory:** The chatbots are not able to memorize a conversation already had, which forces the user to write the same thing over and over again. This can be cumbersome for the client and annoying for the effort required. Therefore, it is important to be careful when designing chatbots and make sure that the program is able to understand users' queries and respond accordingly.

  We have already seen the main advantages and disadvantages of chatbots. They are very advanced, yes. But, as you can see, there is still a lot to do. The truth is that the potential of these tools is incredible.

  We encourage you, if you think it might be useful for your business, to look for an option to start testing. There are some that are not too expensive and with user interfaces that are easy to use. If you want to move forward, there is a world of options and tools to create a chatbot in the marketplace.

CHAPTER – 4

Chatbot development Platforms

The process of building, testing and deploying chatbots can be done on cloud-based chatbot development platforms offered by cloud Platform as a Service (PaaS) providers such as Oracle Cloud Platform SnatchBot and IBM Watson. These cloud platforms provide Natural Language Processing, Artificial Intelligence and Mobile Backend as a Service for chatbot development.

API

Some Companies like Microsoft Azure and AARC are currently providing their Bot Engines through which chatbot Platforms or Software can be developed.

Malicious Use

Malicious chatbots are frequently used to fill chat rooms with spam and advertisements, by mimicking human behavior and conversations or to entice people into revealing personal information, such as bank account numbers. They are commonly found on Yahoo! Messenger, Windows Live Messenger, AOL Instant Messenger and other instant messaging protocols. There has also been a published report of a chatbot used in a fake personal ad on a dating service's website.

Tay, an AI chatbot that learns from previous interaction, caused major controversy due to it being targeted by internet trolls on Twitter. The bot was exploited, and after 16 hours began to send extremely offensive Tweets to users. This suggests that although the bot learnt effectively from experience, adequate protection was not put in place to prevent misuse.

If a text-sending algorithm can pass itself off as a human instead of a chatbot, its message would be more credible. Therefore, human-seeming chatbots with well-crafted online identities could start scattering fake news that seem plausible, for instance making false claims during a presidential election. With enough chatbots, it might be even possible to achieve artificial social proof.

Limitaions of Chatbot

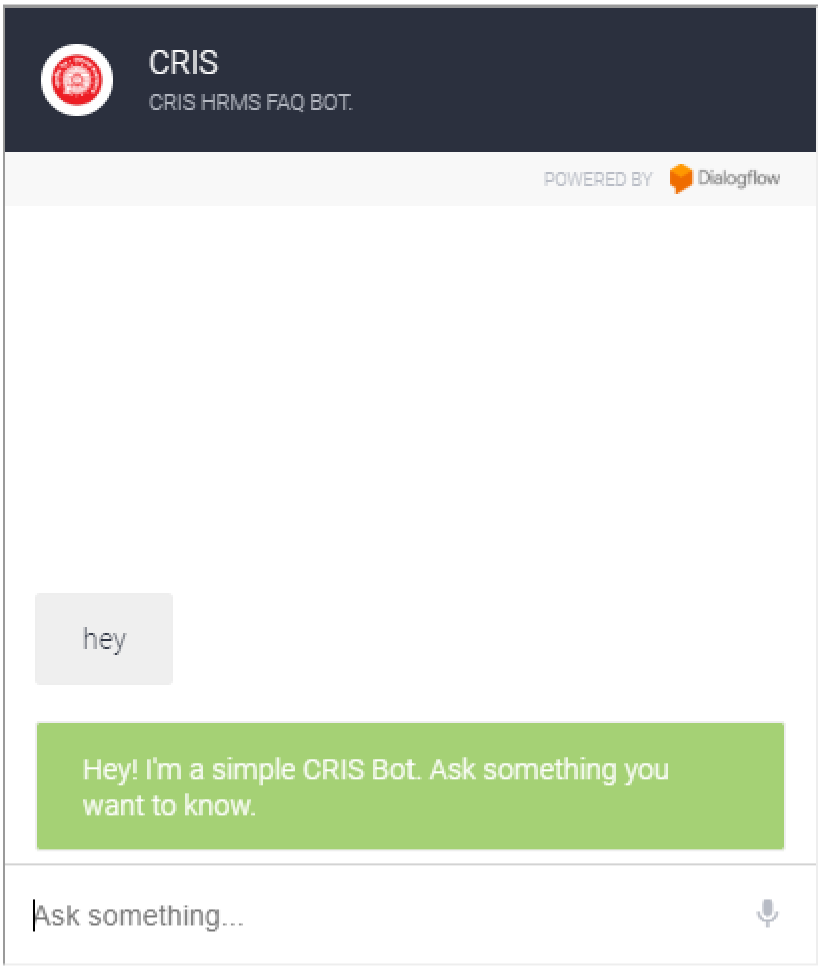
The creation and implementation of chatbots is still a developing area, heavily related to artificial intelligence and machine learning, so the provided solutions, while possessing obvious advantages, have some important limitations in terms of functionalities and use cases. However this is changing over time.

The most common ones are listed below:

- As the database, used for output generation, is fixed and limited, chatbots can fail while dealing with an unsaved query.
- A chatbot's efficiency highly depends on language processing and is limited because of irregularities, such as accents and mistakes that can create an important barrier for international and multi-cultural organisations
- Chatbots are unable to deal with multiple questions at the same time and so conversation opportunities are limited.
- As it happens usually with technology-led changes in existing services, some consumers, more often than not from the old generation, are uncomfortable with chatbots due to their limited understanding, making it obvious that their requests are being dealt machines.

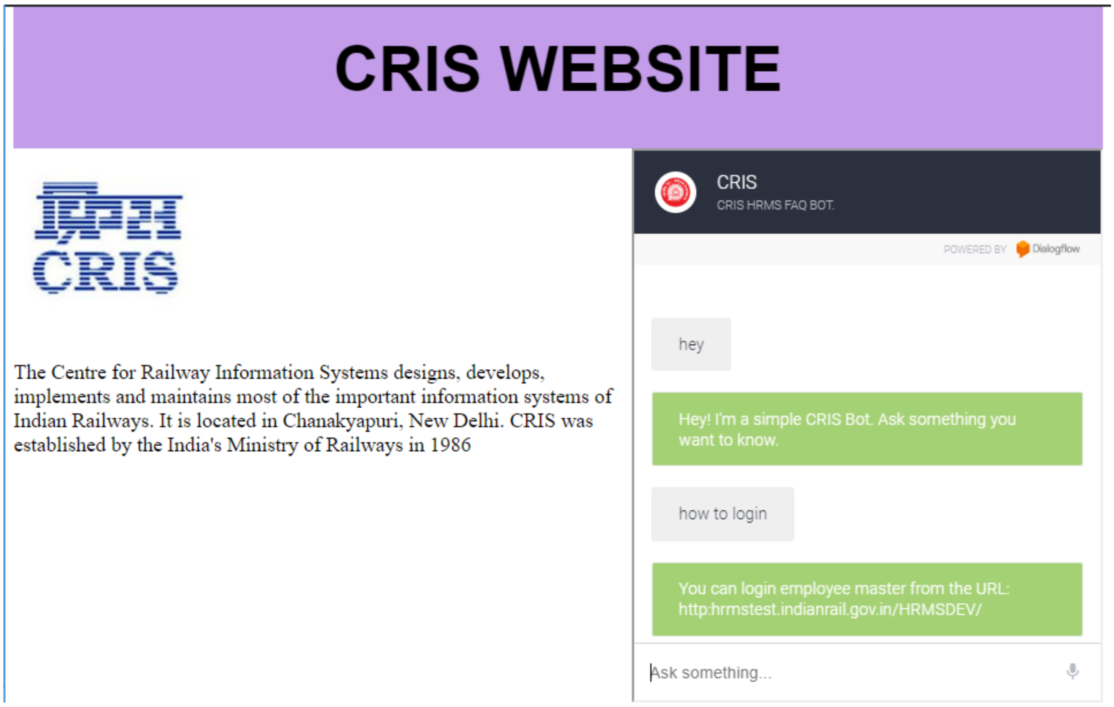
CHAPTER – 5

Screenshots



hey

Hey! I'm a simple CRIS Bot. Ask something you want to know.



CRIS WEBSITE

The Centre for Railway Information Systems designs, develops, implements and maintains most of the important information systems of Indian Railways. It is located in Chanakyapuri, New Delhi. CRIS was established by the India's Ministry of Railways in 1986

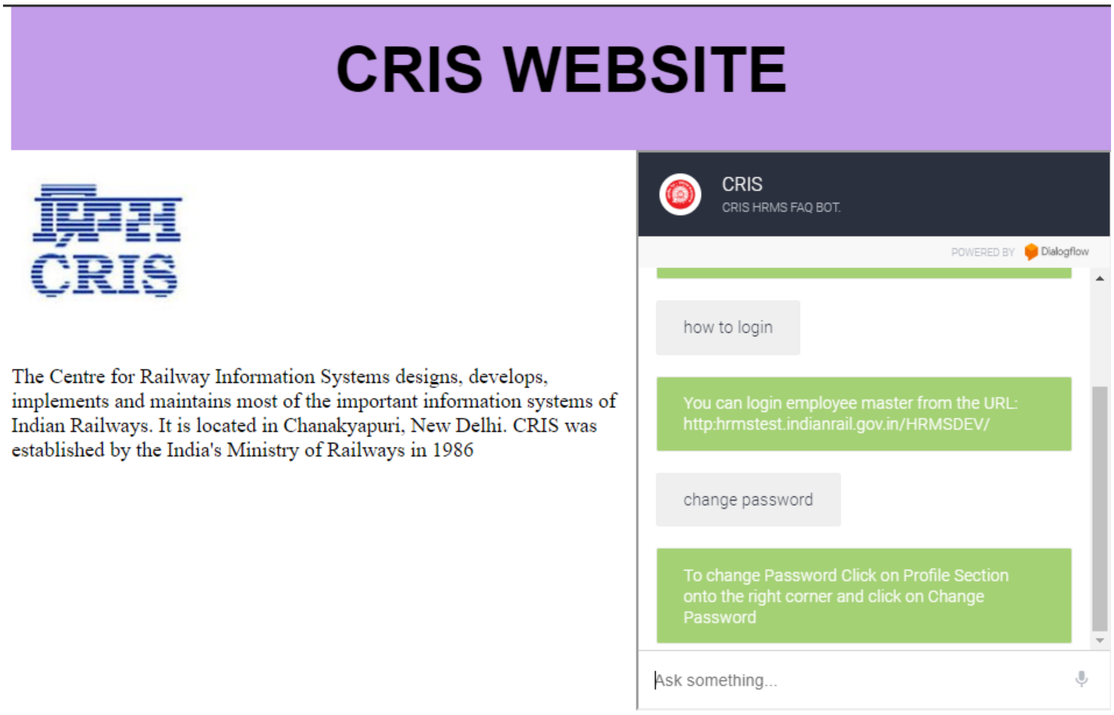
CRIS
CRIS HRMS FAQ BOT.

hey

Hey! I'm a simple CRIS Bot. Ask something you want to know.

how to login

You can login employee master from the URL: http:hrmstest.indianrail.gov.in/HRMSDEV/



CRIS WEBSITE

The Centre for Railway Information Systems designs, develops, implements and maintains most of the important information systems of Indian Railways. It is located in Chanakyapuri, New Delhi. CRIS was established by the India's Ministry of Railways in 1986

CRIS
CRIS HRMS FAQ BOT.

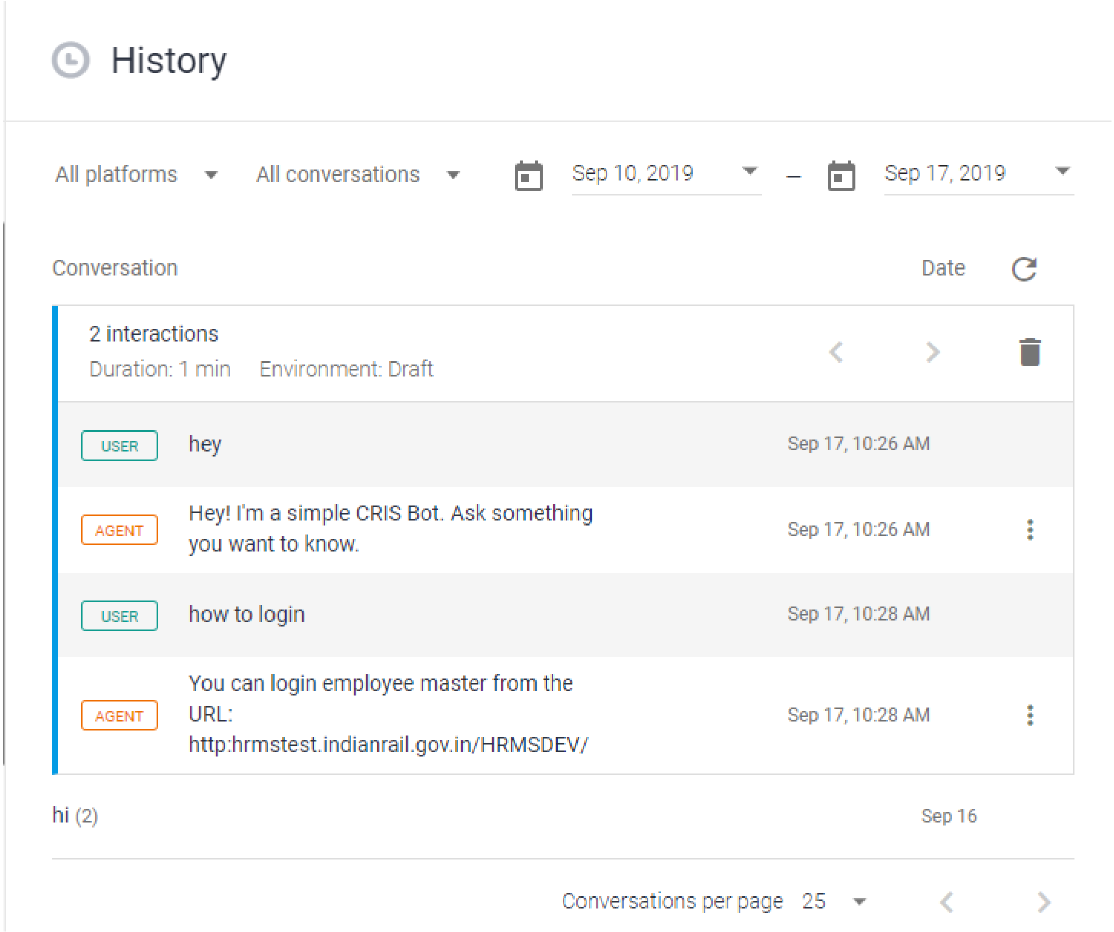
how to login

You can login employee master from the URL: http:hrmstest.indianrail.gov.in/HRMSDEV/

change password

To change Password Click on Profile Section onto the right corner and click on Change Password

This is how the conversations are stored in dialogflow's database. The following image shows it.



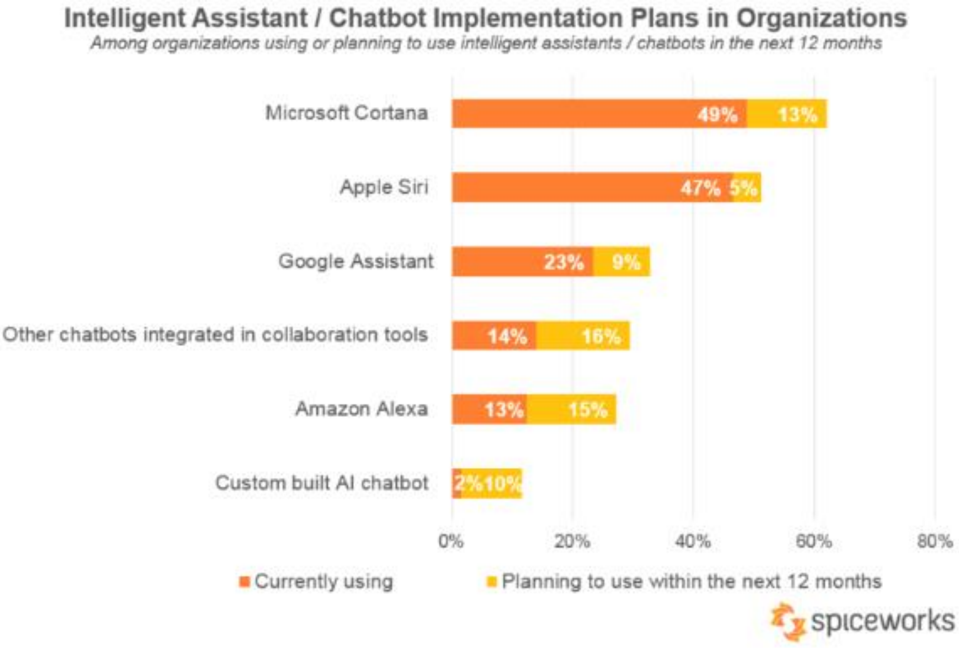
History

All platforms ▾ All conversations ▾ Sep 10, 2019 ▾ – Sep 17, 2019 ▾

Conversation	Date
2 interactions Duration: 1 min Environment: Draft	
USER hey	Sep 17, 10:26 AM
AGENT Hey! I'm a simple CRIS Bot. Ask something you want to know.	Sep 17, 10:26 AM
USER how to login	Sep 17, 10:28 AM
AGENT You can login employee master from the URL: http:hrmstest.indianrail.gov.in/HRMSDEV/	Sep 17, 10:28 AM
hi (2)	Sep 16

Conversations per page 25 ▾

CONCLUSION



Chatbots are the new Apps! As we have discussed in the above chapter, this project brings the power to efficiency in both business and service department. Chatbots give a human like touch to some aspects and make it an enjoying conversation. And they are focused entirely on providing information and completing tasks for the humans they interact with. The above mentioned functionality in all the deliverables is implemented by Dialogflow, HTML and React. By implementing the above mentioned deliverables I was able to add a basic chatbot functionality., I have implemented a simple CRIS chatbot that gives employee and short links to user sites information whenever a user ask and tells that I was also able to converse with the bot. I intend to enhance the system developed so far. Next step towards building chatbots involve helping people to facilitate their work and interact with computers using natural language or using set of rules. Future Dialogflow chatbot, backed by machine-learning technology, will be able to remember past conversations and learn from them to answer new ones. The challenge would be conversing with multiple bot users and multiple user

REFERENCES/ BIBLIOGRAPHY

- Dialogflow documentation
- W3schools.com
- Udemy.com
- Wikipedia.com
- React documentation
- Npm library documentation