TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Gestión de juegos conversacionales

Una aplicación en la nube

Autor
Guillermo Muriel Sánchez lafuente

Directores
Juan Julián Merelo Guervós

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, septiembre de 2018

Gestión de juegos conversacionales

Una aplicación en la nube

Autor
Guillermo Muriel Sánchez lafuente

Directores
Juan Julián Merelo Guervós

Departamento de Arquitectura y Tecnología de Computadores
Granada, septiembre de 2018

Gestión de juegos conversacionales

Guillermo Muriel Sánchez lafuente

Palabras clave: juegos conversacionales, aplicación, computación en la nube, API REST, SCRUM, DevOps, arquitectura cliente-servidor.

Resumen

Los juegos conversacionales necesitan una gestión del servidor que sea capaz de gestionar eficientemente diferentes clientes, navegadores y clientes conversacionales. Además, es necesario presentarlos de forma atractiva para que los usuarios se enganchen en este tipo de juegos.

En este proyecto se programará una aplicación cliente-servidor, con especial énfasis en el servidor, pero con diferentes tipos de front-end; a la vez, el servidor se podrá usar como servicio web y será adaptable a diferentes tipos de juegos.

Management of conversational games: An application in the cloud

Guillermo Muriel Sánchez lafuente

Keywords: conversational game, application, cloud computing, API REST, SCRUM, DevOps, client–server model.

Abstract

All conversational games need a good management of the server that stores them. This server must efficiently manage different accesses of different clients, browsers and conversational clients. In addition, the information must be presented in an attractive way with the aim that users adore playing these types of games.

In this project a client-server application will be programmed, with special emphasis on the server, but it will be able to addapt with different types of front-end; at the same time, the server can be used as a web service and will be adaptable to different types of games.

Yo, **Guillermo Muriel Sánchez lafuente**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 47255733W, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Guillermo Muriel Sánchez lafuente

Granada a 7 de septiembre de 2018.

D. **Juan Julián Merelo Guervós**, Profesor del Área de Departamento de Arquitectura y Tecnología de Computadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Gestión de juegos conversacionales, Una aplicación en la nube***, ha sido realizado bajo su supervisión por **Guillermo Muriel Sánchez lafuente**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 7 de septiembre de 2018.

El director:

Juan Julián Merelo Guervós

Agradecimientos

Poner aquí agradecimientos...

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger

that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces

specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms

that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work

in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a

different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to

a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License.

Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means

to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you

convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `https://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `https://www.gnu.org/licenses/why-not-lgpl.html`.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El ser humano es considerado un ser curioso, inquieto y siempre en busca de la aventura. No hay más que pararse a observar a un niño con un año de edad y comprobarlo.

Nuestra especie necesita retos a los que enfrentarse para crecer y mejorar. En definitiva, abrir la mente para evolucionar.

El mundo en el que vivimos nos plantea diariamente una serie de obstáculos que tenemos que superar día a día. Este trabajo, se realiza la mayor parte de las veces de una manera mecánica. Es decir, durante esa actividad, el cerebro no está usándose a pleno rendimiento. No estamos motivados.

Sin embargo, con la aparición de los juegos, al ser humano se le ofrece una actividad distinta, se le permite crujir su rutina.

Volviendo al origen de nuestra especie, nos adentramos en nuestra principal característica evolutiva, la resolución de problemas, tan fundamental para el desarrollo de nuestro intelecto.

He aquí la idea principal del proyecto, que no es otra que la de generar esa posibilidad que ayude al ser humano a su desarrollo. Todo esto apoyado mediante la creación de un juego conversacional. Gracias al cual se planteen y resuelvan acertijos de los más ingeniosos.

Acertijos propuestos por individuos con alma de detective y resueltos por los mismos.

1.1. Motivación y objetivo

El poder colaborar en el desarrollo intelectual y en el entretenimiento de las personas es motivación más que suficiente para embarcarse en un proyecto de tal magnitud y dificultad.

Actualmente, cuando se decide crear un juego conversacional, se tienen al alcance de la mano, a través de internet, infinitas posibilidades para su creación.

El principal objetivo de este juego es la comunidad se pueda beneficiar del conocimiento incluido en el mimo. Para ello, la tarea fundamental será liberarlo.

Este juego será desarrollado de una manera actual, teniendo muy en cuenta filosofías de desarrollo de software ágiles y con eje fundamental el software libre.

Con el objetivo de que la aplicación sea abierta, este proyecto estará liberado bajo la licencia *GNU General Public License v3.0* [1] y alojado en un repositorio público en GitHub.

Este juego deberá ser accesible desde cualquier plataforma, por lo que para ello será fundamental su desarrollo mediante un servicio en la nube, es decir, el paradigma del '*Cloud Computing*' o 'Computación en la nube' [2] estará muy presente en este proyecto.

1.2. Descripción del proyecto

El proyecto consistirá en el desarrollo de una aplicación cliente-servidor desplegada en la nube, adaptable a cualquier tipo de *front-end* [3] o interfaz.

Este proyecto tendrá especial énfasis en el servidor o *back-end* [3], pero sin olvidar la parte de interfaz, ya que se busca que los usuarios se enganchen a este juego.

Para la parte de servidor, se desarrollará una API REST[4] [5][6], que nos permitirá la comunicación desde la interfaz hasta los datos alojados en la nube, añadiendo la capa de abstracción necesaria para que la aplicación sea adaptable, en un futuro, a diferentes tipos de *front-end* o interfaces.

Esta aplicación se desarrollará mediante la filosofía o práctica DevOps [7] [8] [9] [10]. En este proyecto será fundamental el desarrollo ágil de software y el monitoreo en todas las fases de su desarrollo, desde la integración hasta el despliegue e implementación.

Este proyecto se llamará **Project X**, y estará alojado en la plataforma GitHub en un repositorio del mismo nombre que el proyecto [14]. Esta plataforma facilita enormemente el desarrollo ágil y la filosofía DevOps nombrada anteriormente, haciendo que sea fácil el seguimiento de todas y cada una de las modificaciones realizadas en el proyecto y su continua integración y despliegue.

1.3. Licencia

Este proyecto y su memoria están liberados bajo la licencia GNU[1].

[1] https://es.wikipedia.org/wiki/GNU_General_Public_License

1.4. Estructura de la memoria

- **Capítulo 1. Introducción**: el presente capítulo, es una breve introducción al trabajo realizado.
- **Capítulo 2. Gestión y planificación del proyecto**: cómo se ha realizado la gestión del proyecto, metodología, planificación, riesgos y costes.
- **Capítulo 3. Especificación de requisitos**: descripción de los requisitos e historias de usuario del proyecto.
- **Capítulo 4. Análisis**: análisis de los requerimientos del sistema, tecnología y herramientas de desarrollo.
- **Capítulo 5. Diseño**: diseño de la aplicación, arquitectura y funcionamiento general.
- **Capítulo 6. Implementación**: implementación de los aspectos descritos en el capítulo de diseño anterior.
- **Capítulo 7. Pruebas**: verificación del software y validación del proyecto mediante tests unitarios.
- **Capítulo 8. Conclusiones y posibles ampliaciones**: conclusiones finales del proyecto, posibles ampliaciones y propuesta de mejoras.

Capítulo 2

Gestión y Planificación del proyecto

En este capítulo se tratarán todas las cuestiones relacionadas con la gestión del proyecto: la metodología de desarrollo escogida, la gestión del alcance, tiempo, riesgos y costes.

Lo que se busca con ello es determinar claramente los objetivos a cumplir durante la realización, así como asegurar el buen avance del proyecto repartiendo de forma adecuada los recursos, previniendo posibles problemas y estableciendo un marco de trabajo para estructurar, planificar y controlar el desarrollo del software.

2.1. Gestión del alcance

En esta sección se determinará el alcance general del proyecto, así como los criterios que serán necesarios para la aceptación de éste, y las restricciones a las que está sometido. Por último, se detallarán los entregables que se generarán al finalizar.

2.1.1. Descripción del alcance del proyecto

Para la creación de acertijos por parte de los usuarios y la posibilidad de que la comunidad de usuarios de la aplicación disponga de la posibilidad de poder resolver éstos e interactuar, es necesario tener en cuenta una serie de factores:

- **Por parte de los usuarios:**
  - **Datos personales del usuario:** será necesario almacenar datos personales tales como nombre y apellidos, contraseña y *nick* o nombre de usuario.
  - **Acertijos escritos:** es muy importante asociar cada usuario con el acertijo que él mismo ha escrito.

- **Propuestas de solución a acertijos:** también muy importante el registro en el sistema de las posibles propuestas de un usuario para la solución de los acertijos.
- **Valoración de las propuestas de solución a los acertijos del usuario:** un usuario podrá acceder a las soluciones propuestas para sus acertijos y puntuar según lo cerca que se encuentren de la solución final.

▪ **Por parte de los acertijos:**

- **Datos específicos de cada acertijo:** será necesario almacenar el título, el acertijo, la solución al mismo, 3 pistas para facilitar el comienzo a los otros usuarios la búsqueda de la solución y el estado del porcentaje de resolución de la misma.
- **Acertijos escritos:** es muy importante asociar cada acertijo con el usuario que lo ha escrito.
- **Soluciones propuestas de cada acertijo:** también muy importante el registro en el sistema de las propuestas para la solución de cada acertijo.

La aplicación contendrá dos funcionalidades principales, destacadas y que sustentarán el objetivo principal del proyecto: escritura de acertijos y resolución de los mismos.

Para la funcionalidad de escritura, el sistema será el encargado de almacenar los acertijos propuestos y mostrarlos al mundo. En la segunda funcionalidad nombrada, será el propio sistema el encargado de gestionar la interacción entre usuarios con el objetivo de resolver los enigmas.

2.1.2. Criterios de aceptación

El proyecto será aceptado si la aplicación será capaz de permitir a un usuario la escritura de un acertijo completo. Además, se podrá permitir al usuario proponer soluciones a distintos acertijos propuestos por otros usuarios.

Por último, un usuario será capaz de poder evaluar las propuestas de solución de cada uno de sus acertijos y, éstos, automáticamente actualizarán su estado de resolución basándose en las puntuaciones de las mismas.

2.1.3. Restricciones del proyecto

El proyecto debe suponer un total de 12 créditos, es decir, unas 300 horas de trabajo, y debe ser entregado no más tarde del 07/09/2018.

2.1.4. Entregables del proyecto

Como resultado del proyecto se considerarán los siguientes entregables:

- Código fuente de todo el software desarrollado[14].
- Memoria del trabajo de fin de grado[15].

2.2. Metodología de desarrollo

Antes de iniciar cualquier proyecto de desarrollo de software, es muy importante determinar y definir qué metodología se seguirá. El objetivo es establecer un plan de acción, reduciendo así el riesgo de que se sufran retrasos en el desarrollo, sobre costes, o un mal planteamiento que suponga el fracaso del proyecto en su totalidad. Probablemente en la práctica existan tantas metodologías de trabajo como proyectos de desarrollo, pero en todos los casos es necesario tener en cuenta las características y particularidades del proyecto a la hora de elegir y adaptar una metodología concreta al caso específico.

En el caso concreto de este proyecto, nos encontramos con varios factores importantes que resultan determinantes para la elección de una metodología concreta:

- Al tratarse de una aplicación que necesita de interfaz, es necesario dedicar tiempo al desarrollo de la misma de una manera independiente a las demás partes.
- En esta aplicación será necesaria la creación de una base de datos donde almacenar todos los acertijos, usuarios y propuestas de solución.
- Al tener que desarrollar una API REST que gestione las peticiones desde nuestra interfaz y nuestra base de datos, será necesario dedicar tiempo para el desarrollo de la misma, no de manera independiente, pues se necesita previamente una conexión de ésta con la base de datos.

Por ello, como metodología de desarrollo para mi proyecto lo mejor es crear una mezcla entre la metodología/*framework* SCRUM y la filosofía/práctica DevOps.

El lado SCRUM me aporta la creación de un marco de desarrollo de software con el que obtengo un mayor control y flexibilidad a la hora de enfrentar la incertidumbre, los cambios y los riesgos. Este marco permite realizar modificaciones a la planificación y requisitos de forma controlada y rápida, evitando así que el proyecto fracase por no realizarlos a tiempo. Además, permite que el contacto con el cliente sea mayor y continuo, evitando así que exista demasiada incertidumbre.

Con la cultura DevOps se me proporciona la idea del desarrollo basado en pruebas [16][17]. Con esto obtengo que para cada entregable existe la garantía de que es un producto de calidad, pues ha pasado previamente todos los tests que describían su funcionalidad.

2.2.1. SCRUM

Las metodologías ágiles[54] surgen como respuesta a los problemas característicos de las metodologías tradicionales. Tanto SCRUM como otras metodologías ágiles (Iconix, Cristal Methods), se basan en los siguientes principios:

- Es más importante que se desarrolle un producto de software de calidad y que funcione que escribir una documentación exhaustiva.
- Las interacciones entre individuos son más importantes que las herramientas y procesos utilizados.
- La comunicación con el cliente es vital para el éxito del proyecto.
- Aún cuando establecer un plan es necesario, es más importante el contar con una buena capacidad de respuesta ante los cambios.

Características de SCRUM

Esta metodología se centra en el desarrollo incremental iterativo. A cada una de estas iteraciones las llamaremos ***Sprint***, y normalmente serán de una duración corta entre 3 y 4 semanas. Cada *sprint* termina con una pieza clave de software completa y funcional.

Se prioriza el trabajo que resulta más valioso para el desarrollo del proyecto, con el objetivo de maximizar la utilidad de lo que se desarrolla. Los requisitos y la prioridad de éstos se revisan regularmente, normalmente al inicio de cada *sprint*. El objetivo es construir de forma incremental un producto que se ajuste realmente a las necesidades del cliente, asegurando así su satisfacción.

Para mantener el ritmo de trabajo, sometido a constantes cambios, es necesario que el equipo se centre en construir **software de calidad**, pero para que esto pueda ser posible son necesarios dos aspectos: la gestión del proyecto debe asegurar una buena definición de las características deseables y evitar cualquier obstáculo que pueda afectar el trabajo del equipo; además el cliente debe entusiasmarse realmente por el proyecto desarrollado, de forma que exista un compromiso real que se mantenga tras la finalización de cada *sprint* con el objetivo de mantener la comunicación constantemente y conocer así sus necesidades reales.

Herramientas y roles

Sprint define todas las funcionalidades requeridas del producto a desarrollar en el *Product Backlog*, determinando su prioridad, valor y riesgo, además de una estimación (de forma muy general) de la cantidad de trabajo que supone desarrollarla. Esta lista evoluciona a lo largo de todo el desarrollo.

Además esta metodología define tres roles: el *SCRUM master*, el *Product owner* y el equipo de desarrollo.

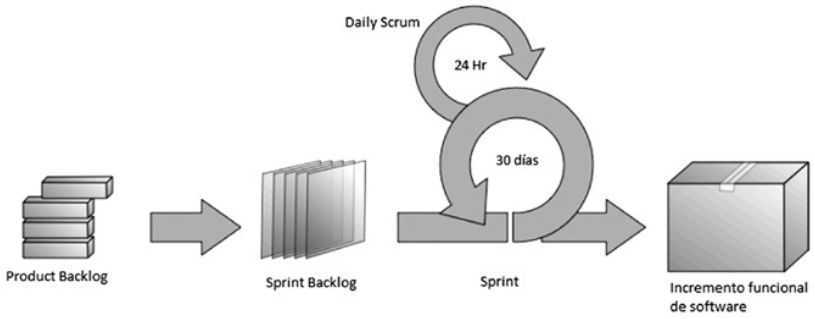


Figura 2.1: Ciclo de vida de un *sprint*

- ***SCRUM Master***: es el líder del equipo. Se encarga de asegurarse de que se está siguiendo la metodología, se cumplen sus valores y prácticas.
- ***Product Owner***: es el encargado de gestionar el desarrollo y el intermediario entre el cliente y el equipo. Se encarga de organizar y priorizar todas las funcionalidades requeridas.
- **Equipo de desarrollo**: es la pieza clave del proyecto. Puede estar a su vez subdividido en equipos de desarrollo, *testing* y despliegue.

Ciclo de trabajo

SCRUM define un evento principal llamado Sprint, el cual corresponde con una ventana de trabajo que, tras su finalización, produce una versión funcional del producto. Dentro de cada *sprint* tienen lugar una serie de eventos. La figura 2.1 muestra el ciclo de vida de un *sprint*.

1. **Planificación de la iteración (*Sprint planning*)**: se define el plan de trabajo de este *sprint* especifico determinando qué se entregara y cómo se logrará. Es decir, se determinan las funcionalidades a implementar y la estimación de cantidad de trabajo. Con todo esto, y partiendo de la información en el *Product backlog*, se establece el *Sprint backlog*.
2. **Reunión diaria (*Daily meeting*)**: cada día se realiza una reunión para explicar los objetivos que se han alcanzado desde la ultima reunión y determinar qué se realizará el día siguiente. Esto permite identificar problemas rápidamente y evitar que se retrase el ritmo de trabajo.
3. **Revisión de la iteración (*Sprint review*)**: se realiza tras la finalización del *sprint* completo. Se revisa todo lo que se hizo y lo que no, se comentan los problemas que tuvieron lugar y cómo se resolvieron, y se revisan los resultados obtenidos.
4. **Retrospectiva de la iteración (*Sprint retrospective*)**: esta reunión tiene el objetivo de analizar los puntos a mejorar y los puntos fuertes del equipo con el fin de aumentar la productividad.

5. **Replanificación**: conjuntamente con el cliente se revisa la planificación teniendo en cuenta los posibles cambios que hayan podido surgir durante el *sprint*.

2.2.2. Aplicación de SCRUM a este proyecto

Dado que la mayoría de las practicas de SCRUM están centradas en mejorar la productividad de un equipo, se han realizado una serie de adaptaciones teniendo en cuenta las características y circunstancias del proyecto. Se ha prescindido de la reunión diaria, pues el equipo de desarrollo está formado por una única persona. En este caso los *sprints* tuvieron una duración de inicial de 3 semanas, por lo que en lugar del *Daily meeting* se ha optado por realizar una autorreflexión semanal en la que se anotaba todo lo realizado hasta el momento y todas las actividades pendientes de abordar. Además, la reunión al final de cada *sprint* se realizó con el tutor. Sin embargo, se mantuvo comunicación con éste constantemente a lo largo de todo el desarrollo.

2.2.3. DevOps

Las antiguas metodologías de desarrollo de software separaban en dos sectores bien diferenciados aquellos que escribían el software (Dpto. Desarrollo) de aquellos que desplegaban en la nube (Dpto. Sistemas) [11].

DevOps surge con la idea de romper esta división y unir en un solo equipo a aquellos que desarrollan software y aquellos que lo despliegan. Esta unión proporciona transparencia a la hora de consultar el estado del producto y las distintas etapas que ha superado o superará.

La filosofía DevOps se basa en los mismos principios que las metodologías ágiles. Sin embargo, con DevOps se quiere fomentar una cultura de equipo, en la que todas las personas que formen el equipo sepan qué está haciendo cada una.

Características de DevOps

Esta filosofía busca la calidad de un producto software, creado por un equipo que se adapte rápido a los cambios e inconvenientes que aparezcan a lo largo de su desarrollo. Estos cambios deben ser identificados con rapidez para su posterior solución.

DevOps engloba todas las buenas prácticas a tener en cuenta a la hora de la entrega, el desarrollo y la administración de aplicaciones a lo largo del ciclo de vida de desarrollo de software: [13]

- Desarrollo y revisión de código con la ayuda de un controlador de versiones.
- Uso de herramientas de integración continua

- Continuo testeo del producto para obtener un software de calidad
- Uso de un gestor de repositorio optimizar la descarga y el almacenamiento de archivos binarios utilizados y producidos en el desarrollo de software.
- Uso de un gestor que automatice el proceso de despliegue
- Uso de un gestor de aprovisionamiento que nos cargue las configuraciones que necesarias de nuestro proyecto
- Monitorización del rendimiento de la aplicación

2.2.4. Aplicación de DevOps a este proyecto

Para este proyecto se usará Git [1] como herramienta para el control de versiones y GitHub [2] como servicio público donde almacenar el repositorio de la aplicación.

Es fundamental en este proyecto el desarrollo basado en pruebas y la integración continua en el repositorio. El código debe de estar testeado previamente antes de añadirlo oficialmente al repositorio del mismo si se quiere obtener un software de calidad.

En este proyecto el equipo está formado por una persona, por lo que el mismo equipo será transparente y formado por el sector de desarrollo y el sector de despliegue.

2.2.5. Combinación de SCRUM y DevOps

Como se ha explicado anteriormente, SCRUM nos define un marco para el desarrollo de la aplicación. Un marco donde se definen *sprints* para la consecución de objetivos.

SCRUM se usará en este proyecto como una estructura que ayude a organizar y dividir los pasos para la creación de la aplicación, cada división la interpretaremos como un *sprint*.

Dentro de cada *sprint*, DevOps ayudará a la obtención de esos objetivos gracias a su filosofía basada en el uso de buenas prácticas relacionadas con el desarrollo basado en pruebas, herramientas de automatización para el despliegue y la revisión del código gracias a un controlador de versiones.

2.3. Gestión de la configuración

En esta sección se define la gestión de la configuración bajo la cual desarrollaremos todas las actividades que impliquen la creación, modificación o eliminación de alguno de los elementos de trabajo que conforman este proyecto,

[1] https://git-scm.com/
[2] https://github.com/

asegurando así que todo lo que concierne al proyecto sea válido durante la vida del mismo.

Teniendo en cuenta que los elementos de trabajo de este proyecto son los ficheros de código fuente y todos los archivos que conforman la documentación, abordaremos esta sección especificando, de forma independiente, cómo se ha tratado cada tipo de elemento de trabajo.

2.3.1. Gestión del código

Para la gestión del código se creará un repositorio local con un sistema de control de versiones, que a su vez se sincronizará con un repositorio remoto en GitHub, y por lo tanto almacenado en la nube. GitHub es un servicio web de control de versiones y desarrollo de software basado en Git[3], publicado bajo una Licencia de código abierto. Esta copia remota previene el riesgo de pérdida del código fuente.

2.3.2. Gestión de la documentación

Para almacenar la documentación de este proyecto se utilizará un repositorio en GitHub llamado **Memoria_TFG_GuillermoMuriel** [4].

Esta plataforma permite la creación de un repositorio cuyo lenguaje principal será LaTeX[5], y le añade una licencia de código abierto.

En cuanto a la edición y gestión de la memoria del proyecto, se utilizará la plataforma online Overleaf v2[6]. Dispone de control de versiones y edición en concurrencia. De la misma forma, al estar almacenada en la nube, se previene el riesgo de pérdida.

A la hora de realizar los diagramas de secuencia correspondientes, se ha utilizado la plataforma Plantuml[7].

2.4. Gestión del tiempo

En este apartado se trata la planificación temporal del proyecto y las fases de la misma. Si bien se ha establecido una planificación inicial, las tareas a realizar se especificarán de forma concreta a medida que se avance en el desarrollo del proyecto en función de los resultados obtenidos tras la finalización de cada *sprint*.

[3] https://git-scm.com/
[4] https://github.com/guillesiesta/Memoria_TFG_GuillermoMuriel
[5] https://www.latex-project.org/
[6] https://v2.overleaf.com/
[7] http://plantuml.com/

2.4.1. Planificación de los *sprints*

Teniendo en cuenta la metodología de desarrollo elegida, se ha optado por describir las tareas abordadas en cada *sprint* y su correspondiente gráfico *Burn-Down*. Los diagramas *Burn-down* son gráficos de dos ejes que se utilizan para representar el avance ideal del proyecto frente al avance real. En el eje X se representa el tiempo y en el eje Y la cantidad de trabajo. Es necesario actualizarlo diariamente. Antes, procederemos a introducir una visión general de los requisitos temporales escogidos sobre la organización del tiempo:

Disponemos de 300 horas para la realización del proyecto.

Para la realización de este proyecto se van a programar 5 *sprints* de 60 horas de duración cada uno. Por semana se trabajará de lunes a viernes unas 4 horas al día, en total 20 horas a la semana.

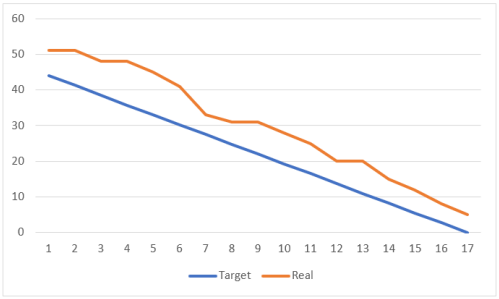
En total, se espera que cada *sprint* dure 3 semanas .

Por lo que el proyecto, salvo imprevisto, debería estar finalizado en torno a las 15 semanas.

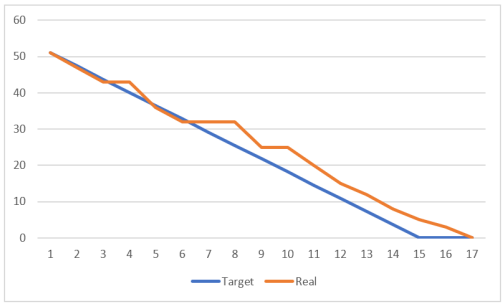
Aproximadamente debería concluir en 4 meses.

- ***Sprint* 1**. En esta primera fase determinará la organización temporal y se estudiará la viabilidad del proyecto. Se extraerán los requisitos y a partir de estos se hará un análisis de las tecnologías existentes para poder desarrollar nuestro proyecto y desplegarlo en la nube y, se escogerán cuales serán las herramientas que más se adecúen a las necesidades del proyecto.
- ***Sprint* 2**. En esta segunda fase se comenzará a trabajar en el almacenamiento de datos, se extraerá toda la documentación necesaria y se procederá a la construcción de la misma. Nuestra base de datos debe estar organizada, estructurada y desarrollada para que en el siguiente *sprint* ya pueda ser utilizada.
- ***Sprint* 3**. En este *sprint* se procederá a construir nuestro intermediario entre la interfaz y la base de datos. Lo llamaremos controlador, es decir, nuestra API REST. Además, se implementaran tests que este controlador deberá de superar, así se confirma su correcto funcionamiento antes del montaje final. En esta fase conectaremos nuestro controlador a nuestra base de datos.
- ***Sprint* 4**. Esta fase está desarrollada única y enteramente al desarrollo de nuestra interfaz. Se escribirán los tests que deben de ser superados para que nuestra interfaz sea apta para su despliegue.
- ***Sprint* 5**. En la fase final se procederá a la unión de nuestra interfaz y nuestro controlador ya unido con la base de datos. A partir de aquí, con nuestro sistema funcionando y testeado, procederemos a desplegarlo usando las tecnologías escogidas previamente.

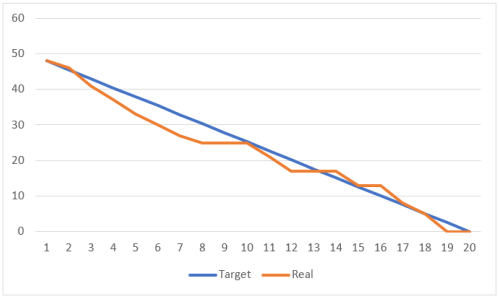
  Una vez la aplicación esté desplegada, se procederá a la redacción de la memoria.



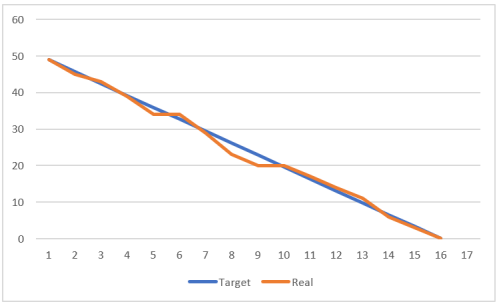
(a) *Sprint* 1



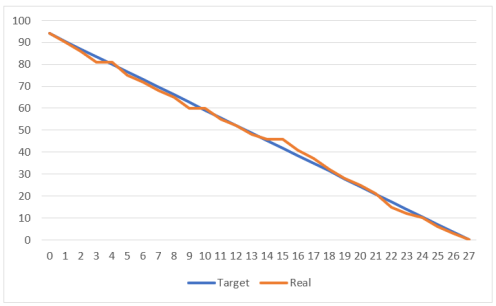
(b) *Sprint* 2



(c) *Sprint* 3



(d) *Sprint* 4



(e) *Sprint* 5

Figura 2.2: Gráficos *burn-down* de los distintos *sprints* del proyecto

2.5. Gestión de riesgos

Un riesgo de un proyecto es un evento o condición que podría tener un efecto positivo o negativo sobre alguno de los recursos u objetivos del proyecto, como podría ser tiempo, coste, alcance o calidad. La gestión de riesgos es un proceso continuo que debe considerar tanto fuentes internas como externas de dichos riesgos. Por norma general, la materialización de un riesgo implica consecuencias negativas, por lo que la gestión de riesgos tiene un papel muy importante dentro de la gestión del proyecto.

En esta sección se identificarán los posibles riesgos del proyecto, estimando la probabilidad de aparición y su impacto potencial. Posteriormente se definirán las estrategias y pasos a seguir para prevenirlos o minimizar su impacto en caso de que se materialicen.

2.5.1. Identificación de riesgos

- **R001** - No identificar correctamente los requisitos del proyecto.
- **R002** - Dificultad en las tareas de formación.
- **R003** - Aparición de nuevos requisitos.
- **R004** - No conseguir ajustar la planificación a las horas requeridas.
- **R005** - Cambios en la planificación o retrasos en la misma.
- **R006** - Errores eligiendo las tecnologías para el desarrollo del proyecto.
- **R007** - Disponibilidad del experto insuficiente.
- **R008** - La estructura de los datos de partida cambia con el tiempo.
- **R009** - Nulo o deficiente proceso de validación y verificación .
- **R010** - Detección de fallos importantes en el software tras la validación.
- **R011** - Detección de fallos importantes en la aplicación tras la validación.
- **R012** - La interfaz no cumple con los requisitos.
- **R013** - La API no cumple con los requisitos.
- **R014** - La base de datos escogida no cumple con los requisitos.
- **R015** - Pérdida de parte del proyecto (documentación o código).
- **R016** - Computador de desarrollo estropeado.
- **R017** - Se sobrepasan los límites de los servicios gratuitos de las plataformas escogidas.
- **R018** - La plataforma GitHub no está disponible.

- **R019** - Las tecnologías usadas cambian de versión durante el proceso de desarrollo.
- **R020** - Problemas de transporte de información entre las tecnologías implementadas.

2.5.2. Riesgos materializados

El proyecto se realizo de una forma óptima, aún cuando se materializó algún riesgo, por ejemplo, el caso del riesgo **R020**, al que fue fácil dar respuesta aunque su impacto en el proyecto resultó ser de gran gravedad: en algunos navegadores es necesario implementar una cabecera para habilitar y dar permiso a la aplicación para el acceso a recursos cuando accedemos a alguna API, alojada en un servidor de un dominio distinto (intercambio de datos de origen cruzado), mediante algún tipo de petición a través del protocolo HTTP[18].

Este problema se solucionó añadiendo en el controlador o API una librería que añadía una cabecera segura en las peticiones HTTP que se realizaban[19].

Avanzando acontecimientos, esta librería pertenece a Python, que es el lenguaje principal en el que hemos escrito nuestra API. Más concretamente, es una extensión de nuestro *framework* Flask, llamada Flask-CORS[20].

2.6. Gestión de costes

En esta sección se hace una estimación de los costes del proyecto teniendo en cuenta los recursos humanos, los materiales y otros costes indirectos.

2.6.1. Costes de recursos humanos

En primer lugar tendremos en cuenta los costes asociados con el equipo de desarrollo. Dado que este equipo está formado únicamente por una persona, se considerará el rol de Analista Programador.

Según la guía Hays del mercado laboral 2018 [21], el salario de un analista programador en la zona de Andalucía rondaría los 22.000 €brutos anuales.

Para el cálculo total del coste de este recurso se ha hecho uso de la calculadora de contratos de la Universidad de Santiago de Compostela[8]. Así pues, teniendo en cuenta el salario mensual, el tiempo que el trabajador estará en el proyecto y el número de horas diarias dedicadas, se calcula el coste total.

- Número de pagas: 14
- Bruto mensual: 1.570,00€
- Periodo de contratación: 01-05-2018 al 01-09-2018

[8] `http://imaisd.usc.es/ferramentas/calculadora/calculadoracontratos.asp`

- Días semanales: 5
- Categoría: Grado - Ingeniero programador en formación
- Coste total 10.400,59€, de los cuales 838,68 son la aportación a la seguridad social.

2.6.2. Costes de material

Se tendrá en cuenta el ordenador portátil con el que se ha desarrollado el proyecto. Suponiendo que la vida útil de un ordenador son 4 años y el portátil utilizado tiene un coste de 550,00€, con un valor residual de 50,00€y un coeficiente de amortización lineal del 33,3 % , el coste de amortización viene dado por:

(Precio de compra - Valor residual) x Coeficiente de amortización

$$(550,00€ - 50,00€) \cdot 0,333 = 166,50€$$

Así pues, ponderando este coste por el número de horas del proyecto con respecto al número de horas anuales de uso del portátil (20 horas semanales por 52 semanas), el coste de material atribuido al proyecto es:

$$166,50€ x (412.5/1040) = 65,95\ €$$

2.6.3. Costes indirectos

Son constes indirectos aquellos que no se pueden incluir de forma directa en el proyecto como: facturas de luz, calefacción, internet y demás. Para estimar este coste se tendremos en cuenta las facturas de la localización dónde se ha desarrollado el proyecto. Este total asciende a unos: 100€al mes. En total unos 400€los 4 meses.

2.6.4. Costes de servicios *cloud*

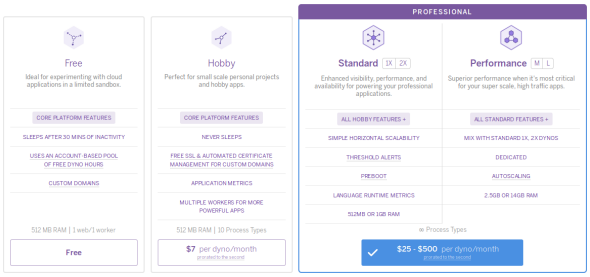
Para la realización de este proyecto se han usado 3 tecnologías distintas de almacenamiento en la nube. Al ser un proyecto que está en una fase inicial de desarrollo, se decidió por contratar los planes gratuitos.

Inicialmente no habrá excesiva demanda en la aplicación. Sin embargo, estos servicios al funcionar bajo demanda, al incrementar los usuarios, peticiones y accesos a la aplicación es importante tener en cuenta los posibles cambios de planes para ajustarnos a la misma.

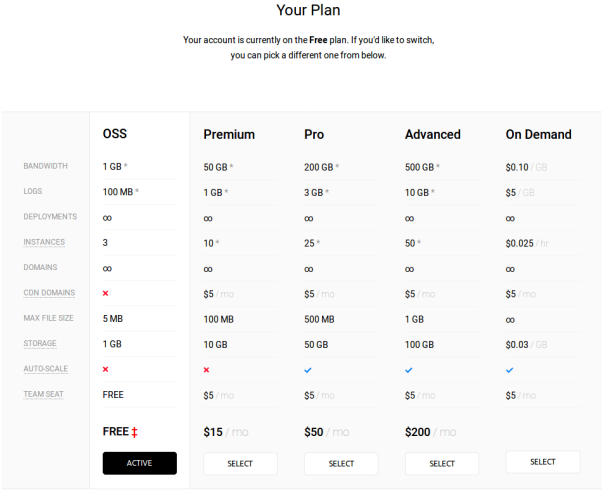
Estos cambios podemos observarlos en la tabla 2.3

2.6.5. Coste total del proyecto

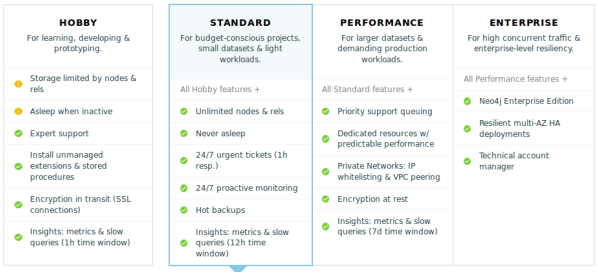
En la tabla 2.1 se hace un resumen del coste total del proyecto.



(a) Planes Heroku Cloud Platform



(b) Planes Zeit Platform



(c) Planes GrapheneDB Cloud

Figura 2.3: Planes *cloud* de los distintos servicios del proyecto

Recurso	**Coste total**
Recursos humanos	10.400,59€
Recursos materiales	65,95€
Costes indirectos	400€
Total	**10866,54 €**

Tabla 2.1: Tabla de costes del proyecto

Capítulo 3

Especificación de Requisitos

Los requisitos se pueden definir como aquellas características esperables y/o deseables que debe cumplir un sistema para satisfacer unas necesidades especificas. Esta tarea es imprescindible para asegurar el éxito del proyecto. En esta sección describiremos los requisitos del proyecto desde una perspectiva menos tradicional que encaja mejor con la metodología de desarrollo elegida: historias de usuario. Las historias de usuario describen de forma breve aquello que un determinado usuario (en este caso desarrollador) espera al final del desarrollo. Es necesario que cumplan una serie de características:

- Deben ser independientes.
- Ser negociables. No actúan como un contrato, sino como una serie de pautas de acción.
- Deben aportar valor al proyecto.
- Se debe poder realizar una estimación sobre ellas (temporal o de carga de trabajo).
- Deben tener un tamaño manejable.
- Deben ser verificables, pues es necesario saber si una historia se ha cumplido.

Cada historia se ha expresado siguiendo el siguiente patrón:

Como `<usuario>`, quiero / espero `<objetivo>`

Además cada una de ellas se acompaña de una estimación temporal, unos criterios de aceptación, un nivel de importancia y un identificador.

Se distinguirán entre tres tipos de niveles de importancia, según su prioridad.

- **Alta** (Cumplimiento obligatorio): la historia es imprescindible para el desarrollo y éxito del proyecto. Es necesario que esté en la versión final para aceptar el proyecto.

- **Media** (Cumplimiento deseable): es deseable la presencia de esta historia en la versión final del proyecto, pero no supone un aumento significativo del valor del producto.

- **Baja** (Cumplimiento opcional): su presencia no es crucial para el éxito del proyecto, pero aporta cierto valor funcional.

Aún cuando el tiempo total de todas las historias de usuario sea mayor al tiempo disponible, no supone un inconveniente, pues las historias se desarrollarán más de una a la vez ya que se solapan y es casi imposible determinar tareas aisladas para el cumplimiento de cada una.

A continuación se especificarán todas las historias de usuario del proyecto teniendo en cuenta el documento guía 3.1 para la especificación de historias de usuario.

Código - Nombre	
Descripción	Breve descripción de la historia de usuario.
Estimación	Estimación temporal de la historia.
Prioridad	Prioridad según la importancia que tiene esta historia para el desarrollo satisfactorio del proyecto.
Aceptación	Criterios a cumplir para considerar que esta historia de usuario se ha cubierto satisfactoriamente.
Notas	Observaciones varias.

Tabla 3.1: Documento guía. Especificación de las historias de usuario.

3.1. Historias de usuario

HU-00 -Proponer una solución a un acertijo	
Descripción	Quiero que el usuario sea capaz de proponer soluciones a los acertijos del sistema.
Estimación	3 semanas
Prioridad	Alta
Aceptación	En el caso de que el usuario introduzca correctamente una solución a un acertijo, ésta se almacenará en la plataforma y se enlazará con el acertijo.
Notas	

HU-01 - Login o acceso a la aplicación	
Descripción	Quiero que el usuario acceda a la plataforma a través de usuario y contraseña.
Estimación	3 semanas
Prioridad	Alta
Aceptación	En el caso de que el usuario introduzca correctamente su usuario y contraseña, se accederá a la plataforma.
Notas	

HU-02 - Escribir un acertijo	
Descripción	Quiero que el usuario sea capaz de escribir un acertijo y subirlo a la plataforma.
Estimación	3 semanas
Prioridad	Alta
Aceptación	Cuando el usuario introduzca su acertijo, éste deberá ser accesible en la plataforma.
Notas	

HU-03 - Consulta de propios acertijos	
Descripción	Quiero que el usuario sea capaz de consultar sus acertijos y el estado de estos.
Estimación	3 semanas
Prioridad	Media
Aceptación	Cuando el usuario introduzca su acertijo, éste deberá ser accesible en la plataforma en un apartado en que solamente se muestren los acertijos escritos del usuario.
Notas	

HU-04 - Puntuación de las respuestas de un acertijo	
Descripción	Quiero que el usuario sea capaz de puntuar las respuestas propuestas por los demás usuarios sobre sus acertijos.
Estimación	3 semanas
Prioridad	Alta
Aceptación	Cuando el usuario acceda a las soluciones del acertijo y modifique la puntuación de la solución, ésta deberá ser actualizada al momento.
Notas	

HU-05 - Actualización del porcentaje del estado de un acertijo	
Descripción	Quiero que sistema actualice el estado de un acertijo basándose en las puntuaciones obtenidas de las respuestas propuestas al mismo.
Estimación	3 semanas
Prioridad	Alta
Aceptación	Cuando el usuario acceda a un acertijo, éste mostrará el estado de resolución en el que se encuentra. Cuando el usuario cambien la puntuación de las soluciones, el estado debe actualizarse a la puntuación más alta.
Notas	

HU-06 - Consulta de todas las soluciones propuestas a un acertijo	
Descripción	Quiero que un usuario al acceder a un acertijo a resolver, tenga la posibilidad de consultar todas las propuestas como solución escritas para el mismo.
Estimación	3 semanas
Prioridad	Alta
Aceptación	Al acceder a un acertijo, si éste tiene soluciones, deberán mostrarse en el caso de que el usuario lo solicite.
Notas	

HU-07 -Modificación de los datos de un usuario	
Descripción	Quiero que un usuario pueda modificar sus datos.
Estimación	3 semanas
Prioridad	Baja
Aceptación	Al acceder a sus datos personales y modificarlos, éstos se actualizarán automáticamente en el sistema .
Notas	Esta funcionalidad se contemplará única y exclusivamente cuando se disponga de tiempo de más en el *sprint* correspondiente.

HU-08 - Adaptabilidad a distintos *front-ends*	
Descripción	Quiero que mi aplicación sea adaptable a distintos *front-end*.
Estimación	3 semanas
Prioridad	Alta
Aceptación	La API desarrollada debe de ser accesible desde cualquier lugar y en cualquier momento. Esto es, debe permitir el acceso a peticiones HTTP.
Notas	

HU-10 - Interfaz sencilla e intuitiva	
Descripción	Quiero que mi aplicación sea fácil de usar y atractiva.
Estimación	3 semanas
Prioridad	Alta
Aceptación	La interfaz no contendrá excesiva carga de información, y será intuitiva.
Notas	

HU-11 - Accesibilidad y gestión de los datos	
Descripción	Quiero que la base de datos de mi aplicación sea sencilla de administrar y que acepte peticiones HTTP de mi API.
Estimación	3 semanas
Prioridad	Alta
Aceptación	La base de datos no dispondrá de excesiva complejidad y será accesible, alojándose ésta en la nube.
Notas	

HU-12 - Tecnologías usadas libres	
Descripción	Quiero desarrollar mi aplicación con software libre.
Estimación	3 semanas
Prioridad	Media
Aceptación	La mayor parte de la aplicación deberá estar desarrollada con software libre.
Notas	

HU-12+1 -Modificación de los datos de un usuario	
Descripción	Quiero que un usuario se pueda dar de alta en la plataforma.
Estimación	3 semanas
Prioridad	Baja
Aceptación	Cuando un usuario no esté dado de alta sea capaz de registrarse en el sistema.
Notas	Esta funcionalidad se contemplará única y exclusivamente cuando se disponga de tiempo de más en el *sprint* correspondiente.

Capítulo 4

Análisis

En este capítulo se aborda la fase de análisis de los distintos requerimientos del sistema para cumplir con los objetivos del proyecto. Tanto los acertijos, las soluciones propuestas a ellos y la puntuación de cada solución son datos que son introducidos por los usuarios. El sistema tiene que ser capaz de poder permitir esa interacción de una manera sencilla e intuitiva.

En primer lugar se abordarán las cuestiones relativas al funcionamiento de la aplicación y, a partir de ahí, se definirá la arquitectura mediante la cual la aplicación podrá desarrollarse.

Una vez definida la arquitectura, se hará una búsqueda exhaustiva de las distintas tecnologías existentes que ayuden a cumplir los requisitos específicos de cada módulo que la forma, así pues se justificará la elección de la tecnología más adecuada para el desarrollo de la aplicación.

Por último, abordaremos el despliegue de la aplicación. Una vez más, se procederá a hacer una búsqueda exhaustiva de las distintas tecnologías que permitirán el despliegue en la nube de cada uno de nuestros módulos diferenciados. Una vez estén definidas, se procederá a la elección de aquella alternativa que nos solucione y complete nuestros requisitos.

4.1. Requisitos de la aplicación

La aplicación se presentará al usuario como una web actual en la que habrá que introducir usuario y contraseña para acceder.

Desde el punto de vista del usuario, una vez dentro de la aplicación se presentarán todos los acertijos del sistema, pues el objetivo principal es que el usuario comience a interactuar. Por ello, es fundamental que en cualquier momento el usuario pueda:

- Proponer un acertijo para compartir con los demás usuarios.
- Acceder a sus acertijos para consultar su estado.

- Puntuar las soluciones propuestas a sus acertijos.
- Proponer soluciones a cualquier acertijo del sistema.

Desde el punto de vista del sistema, para conseguir materializar las funciones básicas a llevar a cabo por un usuario y, aún más importante, la gestión de las mismas por un servicio que sea adaptable a cualquier tipo de interfaz, será necesario el desarrollo de una API REST, que actúe como intermediario entre la interfaz y la base de datos.

Aún dentro del sistema, es necesario el almacenamiento de toda esa información que será usada por el usuario continuamente en una base de datos intuitiva, actual y sencilla de manejar.

4.2. Arquitectura Modelo-Vista-Controlador

Una vez definido el funcionamiento general de la aplicación, se pueden distinguir 3 bloques o módulos bien diferenciados:

- Interfaz o *front-end*
- API REST
- Base de datos

La arquitectura *MVC* (Modelo Vista Controlador) es la más adecuada para el desarrollo de este proyecto, ya que nos permite hacer una distinción de tres bloques independientes que colaborarán entre sí para formar nuestra aplicación[24][25]. Esta arquitectura distingue los bloques[23]:

- **Vista**. En este bloque es donde se representan los datos con los que el sistema operará. En nuestro caso, es nuestra interfaz.
- **Controlador**. Este bloque será el responsable de responder a los distintos eventos en los que se solicite algún tipo de consulta al modelo, o los que se necesite un transporte de información desde el modelo a la interfaz o viceversa. En nuestro caso, será nuestra API.
- **Modelo**. En este bloque se definirá el tipo de datos y su almacenamiento lógico. En nuestro caso, será nuestra base de datos.

Cada bloque de nuestra arquitectura se abordará de manera independiente, teniendo muy en cuenta, que éstos, una vez definidos y desarrollados, tendrán que estar conectados para así permitir el flujo de información correspondiente a los eventos por parte de los requisitos del usuario.

En la imagen 4.1 se pueden contemplar los tres bloques que forman la arquitectura *MVC*.

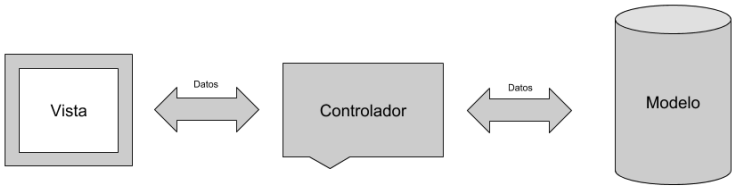


Figura 4.1: Arquitectura Modelo-Vista-Controlador

4.3. Tecnologías a emplear

Comenzaremos definiendo las posibles tecnologías que encontramos para desarrollar cada uno de los bloques de nuestra aplicación. Para escoger las tecnologías a emplear, nos basaremos en los requerimientos de la aplicación y nuestra propia intuición como desarrolladores.

4.3.1. Tecnologías de desarrollo

Inicialmente se hará una búsqueda de las distintas tecnologías para el desarrollo del bloque, se compararán y se decidirá cual de ellas es la escogida y el por qué.

Modelo - Base de Datos

En el mercado podemos encontrar dos tipos distintos de bases de datos: relacionales y no relacionales[26].

Para el desarrollo de nuestro proyecto, estamos buscando una base de datos que sea intuitiva y sencilla de administrar. Es fundamental encontrar una base de datos que administre eficientemente unos datos de una manera distinta a la visión de "tabla"que nos ofrecen las bases de datos relacionales. Es por ello que se ha escogido una base de datos **no relacional** para el almacenamiento de los datos de la aplicación.

Dentro de las bases de datos no relacionales encontramos distintos tipos:

- Bases de datos documentales
- Bases de datos en grafo
- Bases de datos clave/valor
- Bases de datos multivalor
- Bases de datos orientadas a objetos
- Bases de datos tabular

- Bases de datos de *arrays*

Ante esta abrumadora lista de alternativas, el tutor recomendó que se usaran **bases de datos orientadas en grafos**. Este tipo no relacional de bases de datos está en auge y se planea que para un futuro no muy lejano sea el tipo de base de datos más usado[28][29].

Dentro de las bases de datos basadas en grafos se pueden encontrar dos de las más solicitadas:

- Neo4j[1]
- Titan[2]

Para nuestra base de datos buscamos la sencillez, la simplicidad y el uso de un único proceso que actúe como servidor e interfaz gráfica de una base de datos intuitiva. La facilidad de instalación y manejo de **Neo4j** la han convertido en la candidata ideal para nuestra aplicación. Además es importante tener en cuenta la facilidad a la hora de conectar con cualquier API y escalar conforme el tamaño de la base de datos aumente, la posibilidad de hacer copias de seguridad en local y, sobre todo, la aparición de Cypher como lenguaje de consulta[30].

Gracias al inmejorable*online training* y la perfecta descripción de la documentación oficial de Neo4j[3] el aprendizaje de este concepto nuevo de bases de datos y su lenguaje de consulta ha sido un proceso sencillo de llevar a cabo.

El uso de Cypher[4] como lenguaje de consulta ha facilitado notablemente la elección de esta base de datos, pues gracias a éste solamente tenemos que tener en cuenta nodos y relaciones que hay entre ellos. Con una serie de órdenes simples podemos gestionar excesivamente rápida e intuitivamente toda la información que necesitemos para el funcionamiento de nuestra aplicación.

Controlador - API REST

Para que nuestra API sea sencilla de implementar, usando un lenguaje simple, de alto nivel y con un inicio sencillo a la hora del aprendizaje, libre y con una documentación extensa y infinitud de librerías podemos escoger, podemos distinguir varios lenguajes actuales y de los más utilizados para la creación de servicios REST[31][32][33]:

- Javascript - Nodejs
- Python
- Java

[1] https://neo4j.com/
[2] http://titan.thinkaurelius.com/
[3] https://neo4j.com/graphacademy/online-training/
[4] https://neo4j.com/developer/cypher-query-language/

- PHP
- Ruby

Ante estas alternativas, todas son válidas y reúnen la documentación necesaria para desarrollar el proyecto. Es por ello, por lo que para tomar esta decisión se tendrá en cuenta la formación práctica durante el Grado en Ingeniería Informática de la UGR. En esta línea, se ha seleccionado **Python**, pues es un lenguaje en el que, desde el punto de vista personal, se tiene menos soltura en comparación con otros, con lo que optar por desarrollar el proyecto en este lenguaje supone una gran oportunidad para desarrollar nuevas aptitudes y habilidades.

Además, Python es un lenguaje que se está convirtiendo en uno de los más utilizados en la plataforma líder en software libre GitHub[5], con lo que además se abordaría uno de los requisitos a tener en cuenta en el desarrollo del proyecto: la utilización de tecnologías actuales.

Una vez decidido el lenguaje, existen dos alternativas bien diferenciadas a la hora de crear un servicio web en Python:

- Flask[6]
- Django[7]

Una vez estudiadas las características principales de cada alternativa[34] , y puesto que nuestra aplicación necesitará de un gestor ligero que contenga solamente métodos que se llamarán desde nuestra interfaz e interactúen con la base de datos, descartamos la opción de usar Django, pues éste es demasiado pesado y tiene añadida la gran dificultad de comprender la estructura con la que funciona. Éste es justo lo contrario que Flask, una herramienta simple de entender y que con poco tiempo de estudio se es apto para desarrollar una API que cumpla los requisitos de nuestra aplicación.

Nuestra API, por lo tanto estará escrita en **Python** y para el desarrollo del servicio se usará el *framework* **Flask**.

Vista - Interfaz

Existen infinidad de *frameworks* y lenguajes a la hora de desarrollar una interfaz web. Con la idea de querer implementar una interfaz que sea interactiva y ligera, se busca un lenguaje que permita el desarrollo de una web básica, con un ligero aspecto de red social, ya que se busca en todo momento, que el usuario se enganche a la aplicación. Para ello es fundamental el uso de una interfaz fácil de utilizar.

[5]https://octoverse.github.com/
[6]http://flask.pocoo.org/
[7]https://www.djangoproject.com/

Para la búsqueda de este *framework* se ha tenido muy en cuenta, primero que el lenguaje que lo soporte sea sencillo, rápido, muy bien documentado, con muchas librerías adicionales, soportado en el máximo de navegadores y, que se puedan llegar a cumplir todos los requerimientos que la interfaz necesita, como por ejemplo, la capacidad de realizar peticiones HTTP a nuestra API o controlador.

Todos estos motivos nos hacen decantarnos por **javascript** como el lenguaje principal para esta interfaz. Además, este lenguaje nos proporciona diferentes *frameworks* para el desarrollo de nuestra interfaz[35]:

- AngularJS[8]
- ReactJS[9]
- VueJS[10]
- EmberJS[11]
- Aurelia[12]

Al estudiar en profundidad las características de cada uno[36], se ha escogido **ReactJS**. Este *framework* tiene la gran ventaja de que está siendo el más usado hoy en día por plataformas tales como Instagram o Facebook. Es más, ReactJS es una librería desarrollada por Facebook, lo cual quiere decir que tiene un soporte que garantiza una larga vida a largo plazo.

ReactJS tiene como inconveniente la curva de aprendizaje pues puede ser un poco complicado al principio. Sin embargo, gracias a la buena documentación y a los tutoriales a los que podemos acceder a través de la web oficial, y sobre todo, gracias a la gran comunidad que está apareciendo sobre este *framework*, se garantiza la alta probabilidad de alcanzar los objetivos en el período de tiempo estimado.

Un *framework* como ReactJS está basado en la creación de componentes y su reutilización. Esto es de gran ayuda pues en nuestra aplicación aparecerán distintos datos en pantalla, todos esto apoyados en una misma estructura visual que se mantendrá fija salvo, obviamente, que se solicite la muestra de información específica.

4.3.2. Tecnologías de despliegue

Una vez definidas las tecnologías que se usarán para cada bloque de la arquitectura modelo vista controlador, el objetivo será alojar cada uno de éstos en la nube y conectarlos entre sí.

[8] https://angularjs.org/
[9] https://reactjs.org/
[10] https://vuejs.org/
[11] https://www.emberjs.com/
[12] https://aurelia.io/

Existen distintos formas de desplegar en la nube[37][38][39][40]:

- Software como Servicio o **SaaS**. Es un modelo donde el soporte hardware de la infraestructura está gestionado por el proveedor del servicio. El servicio y el software están ya definidos. Los datos los tiene almacenados el hardware del proveedor del servicio. Un ejemplo de SaaS sería: Dropbox[13] o Gmail[14].

- Plataforma como Servicio o **PaaS**. Un PaaS proporciona un entorno *cloud* donde se puede crear una aplicación, es decir, podemos definir qué tipo de servicio queremos implementar. Por la parte hardware, es el proveedor el que se encarga de los recursos de infraestructura dedicados a ese servicio. Un PaaS está justo encima de la capa de infraestructura, nos proporciona la posibilidad de crear un servicio sin tener que preocuparnos por los recursos hardware que se necesitarán para el correcto funcionamiento del servicio.

- Infraestructura como Servicio o **IaaS**. Un IaaS es un modelo en el que tenemos la libertad de implementar cualquier servicio, pero con la posibilidad de definir una infraestructura que sustente ese servicio. Se deben de proveer los recursos necesarios para el arranque del servicio.

En esta aplicación se busca la velocidad y la sencillez, sin entrar demasiado en la definición de los recursos hardware a utilizar. Es por ello que un IaaS retrasaría bastante el desarrollo de la aplicación, pues no está contemplado la posibilidad de definir una infraestructura hardware para este proyecto.

Para la realización de este proyecto se usarán **PaaS** y **SaaS**.

Vista - Interfaz

En el mercado nos podemos encontrar diversos PaaS donde poder alojar y desplegar nuestro servicio, con infinidad de posibilidades para desplegar:

- OpenShift[15]
- DigitalOCean[16]
- Heroku[17]
- Zeit[18]

[13]https://www.dropbox.com/
[14]https://mail.google.com/
[15]https://www.openshift.com/
[16]https://www.digitalocean.com/
[17]https://www.heroku.com/
[18]https://zeit.co/now

En nuestro caso, para el despliegue de la interfaz se ha usado **Heroku**[19].

Esta plataforma nos ofrece la posibilidad de enlazar nuestro repositorio en GitHub con Heroku y automatizar proceso de despliegue con un simple comando. Es por ello que esta plataforma es la idónea, pues en este proyecto se busca exhaustivamente la automatización de procesos como el despliegue y la verificación.Además, esta plataforma está especialmente orientada a despliegues de proyectos no muy extensos, como es el caso.[53]

Controlador - API

Destacando otra vez la facilidad y automatización del despliegue. **Zeit**[20] es un PaaS que nos ofrece una herramienta llamada **Now** que nos hace increíblemente cómodo el despliegue de nuestra API.

Zeit nos permite desplegar nuestra API de manera automática, con solo un comando: *now*. Adicionalmente se necesitaría un archivo en el que se especifique el entorno de nuestra aplicación, esto es, por ejemplo:

- Lenguaje utilizado.
- Librerías a instalar que utiliza el servicio.
- Puerto por el que se transmitirá información. En nuestro caso, solamente realizaremos peticiones HTTP, por lo que el puerto será el 80.
- Localización de nuestro ejecutable y comando para ejecutarlo.

Modelo - Base de datos basada en grafos

En este apartado no se encuentran tantas alternativas para el despliegue en la nube de una base de datos basada en grafos. La propia documentación de Neo4j (nuestra base de datos escogida) nos recomienda **GrapheneDB**[21] como SaaS donde almacenar nuestra base de datos.

Es una decisión rotundamente acertada. Es muy sencillo el uso para la creación de una base de datos basada en grafos en dicha plataforma.

[19] https://dashboard.heroku.com/
[20] https://zeit.co/now
[21] https://www.graphenedb.com/

Capítulo 5

Diseño e Implementación

5.1. Arquitectura Cliente-Servidor

El proyecto en general está basado en la arquitectura cliente-servidor[41]. Esta arquitectura, como su nombre indica, define dos roles principales que se reparten tareas de demanda y respuesta: cliente y servidor. Esta separación se produce en nuestro proyecto cuando distinguimos entre *front-end* (cliente) y nuestro *back-end* (servidor).

Nuestra interfaz haría las funciones de cliente, siendo la demandante de información; por otro lado nuestra API junto con la base de datos tendrían el rol de servidor, encargado de procesar las peticiones y responder de manera adecuada a la petición original.

Con esta disposición se puede destacar la ventaja de que al tener bien diferenciado el *back-end*, nuestra aplicación será adaptable a cualquier tipo de *front-end*. Además, la función de mantenimiento es mucho más fácil de llevar a cabo en este tipo de arquitecturas, pues si aparece algún problema en la parte de interfaz, se puede corregir sin que la modificación aplicada altere el funcionamiento de nuestro *back-end* o API.

5.1.1. Arquitectura Cliente-Servidor y Modelo-Vista-Controlador

Aún cuando estas dos arquitecturas no son incompatibles, tal y como se ha comentado en la sección 4.2, en este proyecto se han diferenciado tres bloques distintos, basados en el *MVC*, con funciones asignadas: vista, controlador y modelo.

Una vez desarrollados esos bloques, la arquitectura cliente-servidor se engloba en un solo bloque, llamando servidor a los bloques: controlador y modelo. La función que esta arquitectura entiende por servidor es la que se realizaría gracias a la unión de estos dos bloques.

5.2. Diseño

El diseño es un proceso que permite construir una representación abstracta del software de forma que se puedan conocer aspectos como la arquitectura, funcionalidades o la calidad antes de comenzar la codificación. Además, permite detectar problemas en etapas muy tempranas del proyecto, ahorrando costes y reduciendo la presencia de riesgos. En SCRUM el diseño del software es continuo y se actualiza en cada *sprint*.

A continuación se mencionan algunos aspectos del diseño de la aplicación que resultan interesantes o relevantes para el correcto funcionamiento.

5.2.1. Representación de los datos

Los datos serán representados en nuestro bloque o módulo que actuará como vista. Para ello, como ya describimos en sección 4.3.1, lo desarrollaremos usando el *framework* ReactJS.

Los datos tienen que mostrarse en una interfaz que sea sencilla de usar, intuitiva y que se adapte a todo tipo de navegadores. Para la elección del aspecto de la misma, es necesaria la búsqueda de inspiración. Por lo que la interfaz de la aplicación Instagram[1] ha sido considerada como modelo a seguir. Observando esta interfaz en la figura 5.1, se puede comprobar que, en cualquier momento, un usuario tiene la posibilidad de cambiar la vista general gracias a los 4 botones que contiene en el extremo inferior de la pantalla. Esta funcionalidad, será clave a la hora de buscar una plantilla que ayude para el inicio del desarrollo de nuestra interfaz pues en nuestro diseño será fundamental que el usuario tenga la posibilidad de desplazarse por los distintos apartados de la aplicación siempre que lo necesite.

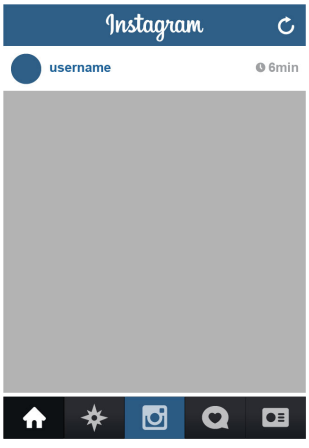


Figura 5.1: Interfaz de Instagram

La idea principal es la reutilización de código con licencia de código abierto. Gracias a la comunidad de software libre, se ha encontrado una interfaz que

[1] https://www.instagram.com/?hl=es

relacione la idea descrita anteriormente junto con un diseño sencillo e intuitivo. Esta plantilla se llama **adminLTE**[2], está liberada bajo la licencia de software libre *MIT*[3] y se encuentra alojada en la plataforma GitHub.

Gracias al uso de la misma, se ha podido desarrollar una interfaz sencilla de desarrollar y agradable a la vista. En la imagen 5.2 se puede observar como se ha desarrollado una interfaz en la que un usuario puede, en cualquier momento, cambiar el contenido de la vista principal.

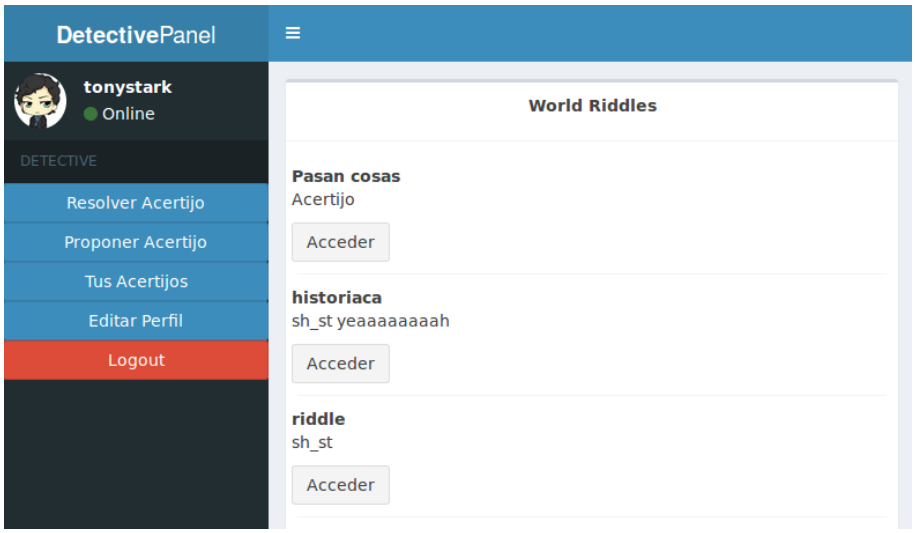


Figura 5.2: Interfaz de Riddling, nuestro proyecto

ReactJS y sus componentes

ReactJS es un *framework* que se basa en la reutilización de componentes. Un componente es cualquier estructura definida en la que se mostrará información. Cada componente se comporta como una instancia de una clase, por lo que cada uno de ellos tiene sus atributos definidos.

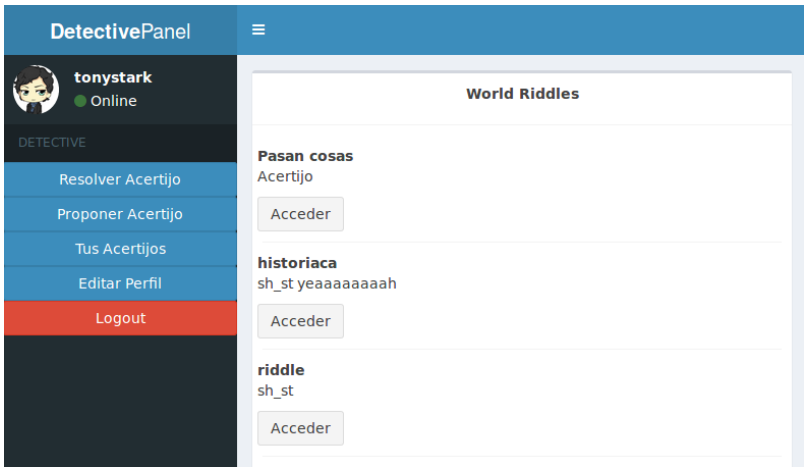
El procedimiento a seguir para la creación de la vista consiste en el diseño y desarrollo de las 4 vistas iniciales que dispondrá nuestra aplicación:

- **AcertijosView**. En esta sección se mostrarán todas las historias accesibles del sistema. Esta vista contendrá todos los componentes *SmallRiddle* que posteriormente describiremos. Esta vista se puede observar en la figura 5.3a.

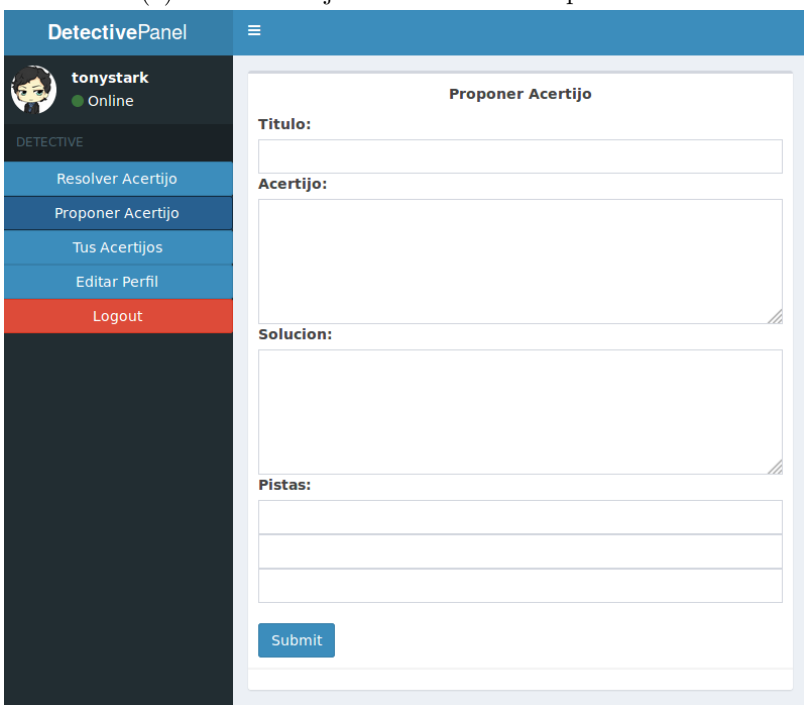
- **ProponerView**. En este apartado se mostrará un formulario para que un usuario introduzca un acertijo. Esta vista se puede observar en la figura 5.3b.

[2]https://github.com/almasaeed2010/AdminLTE
[3]https://github.com/almasaeed2010/AdminLTE/blob/master/LICENSE

- **TusAcertijosView**. En este apartado se mostrarán todas las historias propias de un usuario. Esta vista también contendrá todos los componentes *SmallRiddle* que posteriormente describiremos. Esta vista se puede observar en la figura 5.3c.
- **EditarPerfilView**. Gracias a esta vista un usuario será capaz de administrar su perfil. Aparecerá un formulario con los datos que podrá modificar. Nótese que esta interfaz se encuentra diseñada pero actualmente se encuentra sin funcionalidad. Se puede observar en la figura 5.3d.

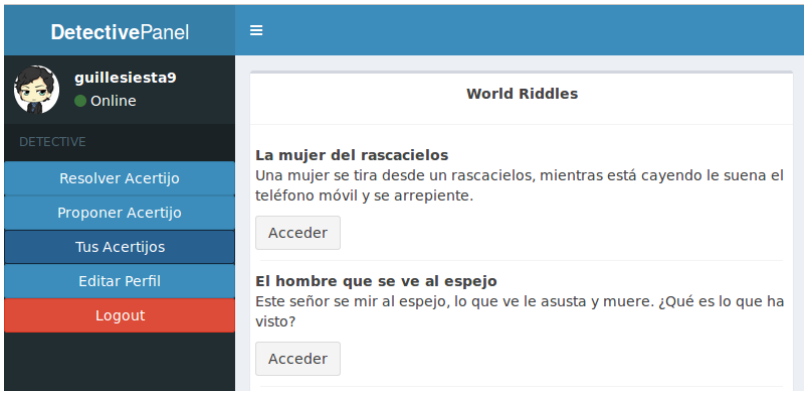


(a) Vista AcertijosView de nuestra aplicación

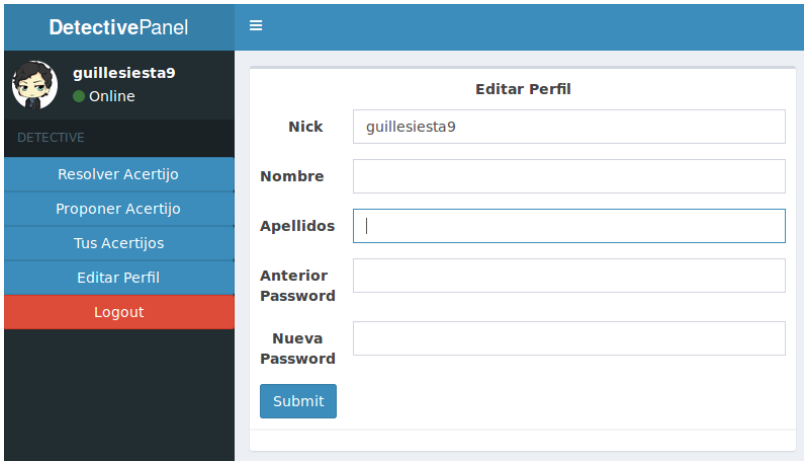


(b) Vista ProponerView de nuestra aplicación

Figura 5.3: Imágenes de las vistas que componen la aplicación



(c) Vista de TusAcertijosView de nuestra aplicación



(d) Vista de EditarPerfilView de nuestra aplicación

Figura 5.3: Imágenes de las vistas que componen la aplicación

Una vez definidas las vistas, éstas contendrán distintos tipos de componentes que interactuarán entre ellos para poder realizar las funciones descritas en los requisitos de nuestra aplicación. Estos componentes se dividen en:

- **Header**. En este componente encontramos la cabecera de la aplicación. Contiene un botón que al pulsarlo contrae hacia la izquierda el menú *Sidebar*. Está siempre renderizado.

- **Sidebar**. Este componente es el encargado de seleccionar qué vista es la que queremos escoger y visualizar. Este componente se encuentra siempre accesible.

- **FormLogin**. El componente inicial que nos permite acceder a la plataforma mediante un usuario y una contraseña.

- **SmallRiddle**. Es un componente que resume el título del acertijo y la descripción del mismo. Tenemos la posibilidad de acceder a él para así cargar el componente *Riddle*.

- **Riddle**. Muestra todos los datos del acertijo, a excepción de la solución. Este componente nos permite observar el estado en el que se encuentra el acertijo, la descripción del mismo, y las tres valiosas pistas que nos ayudarán a resolverlo.

  Como funcionalidad adicional, este componente puede cargar todas las soluciones propuestas de ese acertijo por los usuarios y el porcentaje de cercanía con respecto a la solución final.

- **Solución**. Componente que muestra el texto de la propuesta como solución de un acertijo y la puntuación otorgada por el creador del acertijo.

- **SolucionConCheck**. Este componente muestra la solución de un acertijo en concreto, y tiene la posibilidad de poder establecer una puntuación con respecto al porcentaje de cercanía a la solución del acertijo. Este componente se renderiza únicamente en la vista *TusAcertijosView*.

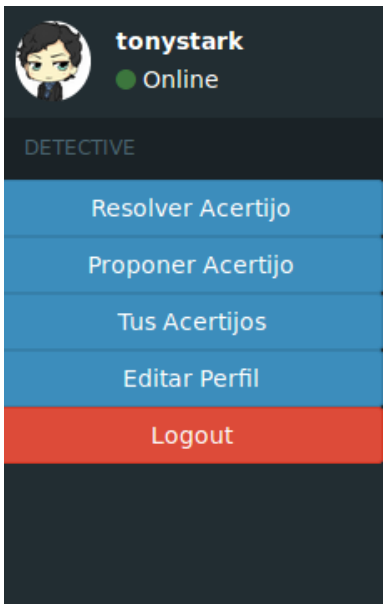
En la figura 5.4 se puede observar el aspecto de cada uno de los componentes descritos.

5.2.2. Modulado de los datos

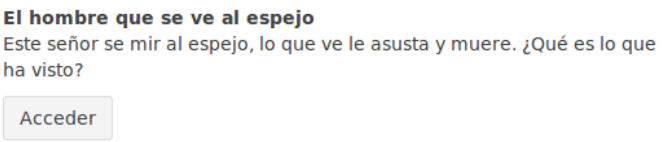
En esta aplicación se van a distinguir dos tipos de entidades: usuarios y acertijos. Esto, en términos de la base de datos basada en grafos, corresponde a dos tipos de nodos específicos: **Usuarios** y **Storie**.

Se representan en la base de datos tal y como se muestra en la figura 5.5.

(a) Componente Header de nuestra aplicación

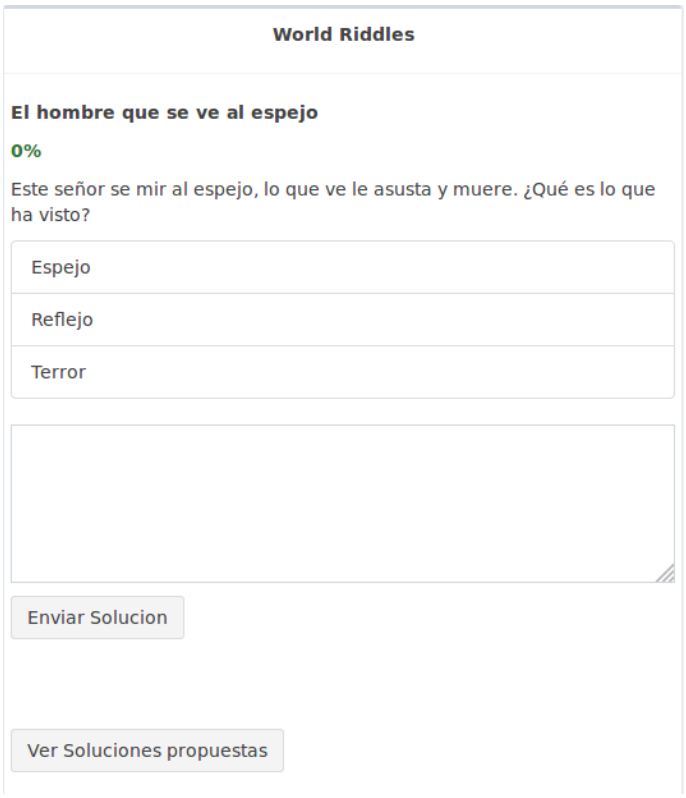


(b) Componente Sidebar de nuestra aplicación



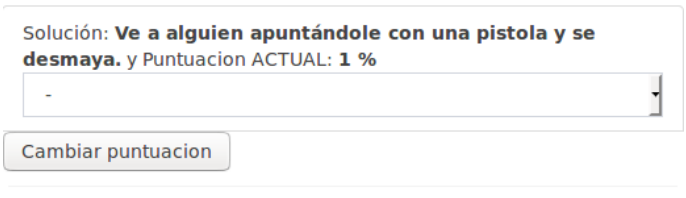
(c) Componente SmallRiddle de nuestra aplicación

Figura 5.4: Imágenes de los componentes que componen la aplicación



(d) Componente Riddle de nuestra aplicación

(e) Componente Solución de nuestra aplicación



(f) Componente SoluciónConCheck de nuestra aplicación

Figura 5.4: Imágenes de los componentes que componen la aplicación

(a) Nodo Usuario

(b) Nodo Storie

Figura 5.5: Nodos de nuestra base de datos

Atributos del usuario

En la base de datos se almacenarán los siguientes atributos por nodo usuario:

- **Nick**.
- **Contraseña**.
- **Nombre**.
- **Apellidos**.

Atributos del acertijo

En la base de datos se almacenarán los siguientes atributos por nodo Storie:

- **Título**.
- **Sh_Storie**. (acertijo)
- **Storie**. (solución)
- **Pista 1**.
- **Pista 2**.
- **Pista 3**.

Relaciones entre nodos

En el sistema se distinguen dos tipos de relaciones:

- **Escribe**. Relacionará cuando un usuario escribe un acertijo.

- **Propone**. Relacionará cuando un usuario propone una solución a un acertijo.

Una vez se establece el tipo de relaciones existentes en el sistema y se decida crear una relación *Usuario escribe historia*, la representación gráfica sería la que se muestra en la figura 5.6.

Figura 5.6: Relación (Usuario)-[Escribe]-(Storie)

Neo4j nos ofrece la posibilidad de poder añadir atributos a las relaciones. Esto será útil para la relación *Propone*, pues en esa relación será necesario registrar el texto propuesto como solución y la puntuación de esa propuesta (inicialmente será de 1). Esta relación quedaría representada tal y como describe la figura 5.7.

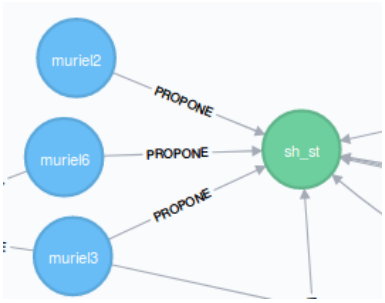


Figura 5.7: Relación (Usuario)-[Propone]-(Storie)

Conclusión sobre el modelado de grafos y su consulta

Como se puede observar, es realmente intuitiva y sencilla la representación gráfica que este tipo de base de datos nos propone. Sin embargo, más sencillo en intuitivo es aún el lenguaje Cypher necesario para la realización de consultas. Pues éste, con una breve ojeada a su manual de usuario[4], ya se es capaz de realizar cualquier tipo de consulta necesaria para el funcionamiento de la aplicación. Por ejemplo, en el caso de que necesitásemos extraer de la base de datos todos los acertijos propuestos por el usuario "aristoteles", la consulta sería la siguiente:

```
MATCH (u:Usuario)-[r:Escribe]-(s:Storie)
WHERE u.nick=aristoteles
RETURN s.storie
```

[4]https://neo4j.com/docs/developer-manual/current/cypher/

5.2.3. Control de los datos

Esta aplicación tiene que garantizar que todos los datos enviados y recibidos por los componentes o la base de datos son gestionados de manera adecuada. Es por ello que el controlador implementa una serie de métodos que aseguran la integridad de los datos y que se invocan desde la interfaz.

Este controlador debe de estar conectado a nuestra base de datos, y tiene que ser apto al recibo de peticiones por parte de la interfaz. Además, deberá limitarse única y exclusivamente a la ejecución de esos métodos, por lo que necesitamos que sea un servicio ligero. Flask nos garantiza todo lo anterior.

Nuestro controlador tendrá los siguientes métodos:

- **all_stories_titulo()**. Devuelve todos los títulos de las historias del sistema.
- **user_stories_titulo()**. Devuelve los títulos de las historias creadas por un usuario en concreto.
- **soluciones_por_titulo()**. Devuelve todas las soluciones propuestas para un acertijo y su puntuación.
- **enviar_comentario()**. Se añade en la base de datos la propuesta de solución de un usuario para un acertijo en concreto.
- **enviar_storie()**. Se añade en la base de datos un acertijo.
- **cambiar_puntuacion_titulo()**. Se modifica en la base de datos la puntuación de una posible solución de un acertijo. Una vez cambiada la puntuación, se procederá a la actualización del estado del acertijo buscando cuál es la mayor puntuación que existe de las soluciones propuestas.
- **acertijo_por_titulo()**.Devuelve el acertijo que coincide con un título especifico, así como el texto del mismo.
- **storie_por_titulo()**.Devuelve el acertijo que coincide con un título especifico y la solución del mismo.
- **todo_por_titulo()**.Devuelve el acertijo que coincide con un título especifico y todos sus atributos.
- **login()**. Se busca en la base de datos si el usuario existe y la contraseña coincide con la registrada en el sistema.

5.2.4. Flujo de los datos

El flujo de datos puede ir desde la interfaz hasta la base de datos y viceversa. Para que esto ocurra, se usa el controlador como intermediario.

El formato de los datos estará estructurado en JSON[5], debido a su alto uso en peticiones y respuestas mediante el protocolo HTTP. Esto es, en el caso de que desde nuestra interfaz decidamos añadir un acertijo, desde ahí se enviarán en formato JSON todos los datos al controlador, que posteriormente extraerá la información relevante de ese JSON y la añadirá a la base de datos.

Un ejemplo de flujo de datos y pasos a seguir es el descrito en los diagramas de secuencia de la figura 5.8. En estos diagramas se describe el funcionamiento de los eventos *enviar_storie(), enviar_comentario(),* en las que un usuario introduce un nuevo acertijo en el sistema.

5.3. Implementación

Una vez descrito el funcionamiento de la aplicación, el siguiente paso es la implementación. Se distinguen dos fases a la hora de llevar a cabo esta etapa. En primer lugar, se desarrollará la aplicación en la máquina local y posteriormente, se desarrollarán los automatismos necesarios para el despliegue de la aplicación.

5.3.1. Implementación en local

Para la implementación de la aplicación en local se han implementado los tres bloques de una manera independiente, comenzando por la instalación de la base de datos, siguiendo por la creación de la API y la conexión con la anterior, hasta la creación de la interfaz.

Base de datos

Para ello, en primer lugar se ha instalado la base de datos neo4j siguiendo el tutorial oficial. Tras realizar todos los pasos, se podría acceder a la base de datos en la dirección*localhost://7474*, asignada por defecto[44].

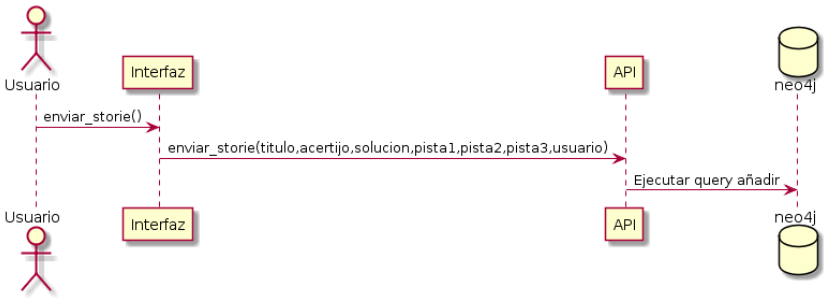
API

Para la creación de la API REST, se ha usado Flask, y se ha implementado este servicio de una manera cómoda y sencilla. Simplemente se ha ejecutado el comando `python nuestraapi.py`, que arranca el servicio local, accesible por el puerto 5000[45].

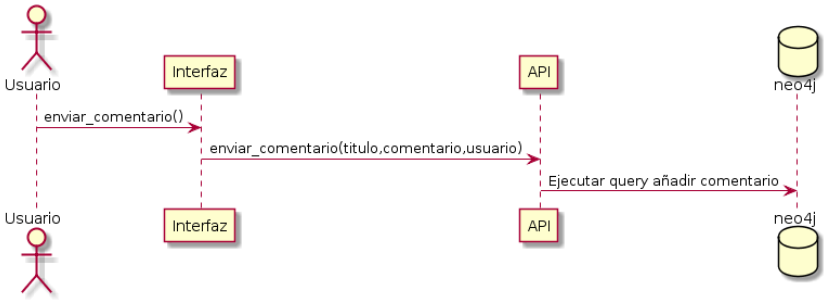
Interfaz

Para el desarrollo de la interfaz se ha seguido el tutorial oficial para el inicio y la instalación de ReactJS[46]. Este proceso se debe arrancar y por defecto se situará en local en el puerto 3000.

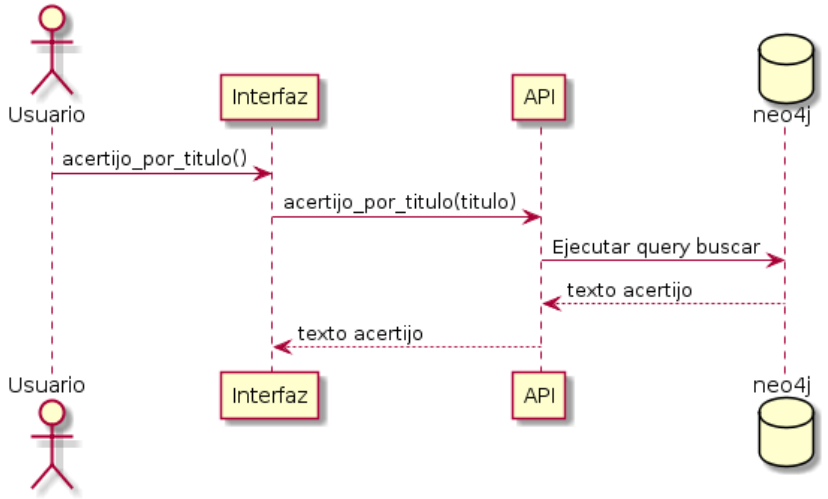
[5]https://www.json.org/



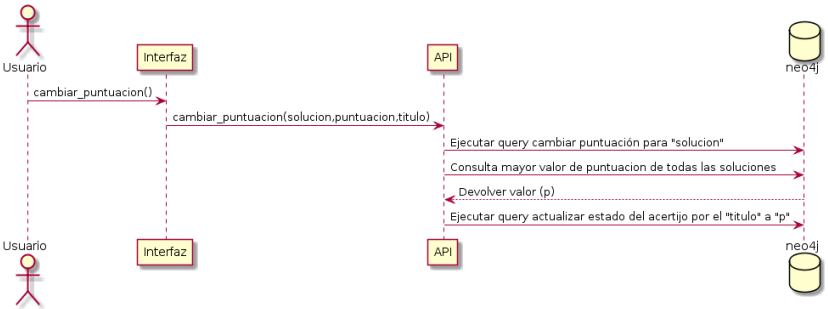
(a) Diagrama de secuencia del evento enviar_storie()



(b) Diagrama de secuencia del evento enviar_comentario()



(c) Diagrama de secuencia del evento acertijo_por_titulo()



(d) Diagrama de secuencia del evento cambiar_puntuacion()

Figura 5.8: Diagramas de flujo de algunos eventos de la aplicación

Conectando interfaz, controlador y modelo

Una vez desarrollado cada bloque, es necesario establecer una conexión para crear el cauce por donde fluirá la información.

Para ello, en nuestra API se debe establecer primero la conexión mediante las claves ofrecidas por la base de datos[47]. Posteriormente, se debe instalar la librería Py2neo[6] que, recomendada por la documentación oficial de neo4j, sera de ayuda a la hora de hacer consultas y establecer la conexión con la base de datos.

Una vez esté el *back-end* conectado entre sí (API+Neo4j), será necesario conectarlo con la interfaz.

En este sentido, una de las ventajas de javascript (el lenguaje oficial para la interfaz) es que dispone de una función *fetch*[7], que se encargará de realizar las peticiones HTTP a nuestra API.

5.3.2. Implementación en la nube

A la hora de implementar en la nube, se seguirá el mismo orden que en local.

Base de datos

En primer lugar, es necesario darse de alta en la plataforma GrapheneDB[8]. Previamente exportada la base de datos en local (en el caso de que se quiera mantener la información) se procede a la creación de una nueva base de datos en GrapheneDB. Una vez creada, se restaura la nueva creación con los datos exportados de la base de datos en local[48].

Siguiendo estos sencillos pasos, ya estaría la base de datos desplegada en la nube.

API REST

Para el despliegue de la API REST en la plataforma Zeit[9] primero es necesario especificar qué entorno se construirá. Para ello, es necesario crear dos archivos:

- **requirements.txt**[10]. Este archivo define un entorno mediante el cual la aplicación podrá ejecutar Python.

[6] https://py2neo.org/v4/
[7] https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch
[8] https://www.graphenedb.com/
[9] https://zeit.co/
[10] https://github.com/guillesiesta/ProjectX/blob/master/requirements.txt

- **Dockerfile**[11]. Este archivo, define un entorno en el cual Zeit podrá crear un contenedor donde alojar el servicio.

Una vez creados estos archivos, se debe comunicar en la API que dejará de comunicarse por el puerto 5000 y pasará a comunicarse por el puerto 80. Para ello, habrá que añadir la siguiente línea al final del documento: *app.run(host='0.0.0.0', port=80).*

Una vez efectuados los cambios, se ejecuta desde el directorio de la aplicación el comando: `now --public` y comenzará automáticamente el despliegue.

Zeit crea nombres de dominio aleatorios, en este caso el nombre de dominio, es decir, la API o servicio estará desplegada en el dominio: **https://projectx-wvueafqhpp.now.sh/**

Interfaz

Para el despliegue en Heroku de la interfaz, es necesario instalar primero el cliente en local[50]. Además, antes del despliegue es necesario cambiar la dirección de todas las funciones `fetch`, pues la API ahora estará alojada en la nube, y es la que se usará.

Una vez realizados los cambios correspondientes e instalado el cliente, se añadirá el repositorio local a los repositorios de Heroku[51].

Por último, se puede procede al despliegue de la interfaz con el comando: `heroku create`.

Automáticamente se desplegará una aplicación a la que, desde el panel principal de Heroku, se le podrá modificar el nombre, pero no la extensión que añade Heroku por defecto. Para tener un dominio propio se debería de escoger otro plan que no sea el gratuito.

La dirección oficial y, también la confirmación de que la aplicación está alojada en la nube es: **https://riddling.herokuapp.com/**

[11] https://github.com/guillesiesta/ProjectX/blob/master/Dockerfile

Capítulo 6

Pruebas

En esta sección se abordarán las distintas tareas relacionadas las diferentes pruebas realizadas para asegurar la calidad del producto final antes de su entrega. El objetivo del proceso de pruebas es la verificación del software; es necesario poner a prueba el comportamiento del software para identificar situaciones en las que no se obtengan los resultados esperados.

6.1. Integración continua en el repositorio

Todas estas pruebas se han ido realizando cada vez que se actualizaba el repositorio del proyecto en GitHub. Para ello, se ha usado Travis[1], un sistema que permite la integración continua en el repositorio. Una vez enlazado Travis con el repositorio, la realización de test es un proceso automático que se realizará siempre.

Gracias a esta herramienta libre, se puede comprobar en cualquier momento si el software se encuentra validado o no. En el caso de que el test falle, Travis notifica dónde está el error.

Travis necesita de un archivo de configuración *.travis.yml*[2] para el arranque, creando una máquina virtual que simulará el sistema y comenzará a examinar los cambios realizados. En este caso concreto, para realizar los tests del el módulo de la API(Python) y de la interfaz(Javascript), es necesario crear el archivo tal y como se muestra en la figura6.1.

6.2. Pruebas en *back-end* - API

Actualmente, en el mercado nos encontramos con infinidad de librerías para el desarrollo de tests en Python. Dos de las más destacadas son:

[1]https://travis-ci.org/

[2]https://github.com/guillesiesta/ProjectX/blob/master/.travis.yml

```
matrix:
  include:
    - language: python
      python:
        - "2.7"
      install:
        - pip install -r requirements.txt
      script:
        - pytest

    - language: node_js
      node_js:
        - "node"
      before_install:
        - cd stories
      install:
        - npm install
      script:
        - npm test
```

Figura 6.1: Archivo .travis.yml del repositorio en GitHub

- Unittest[3]
- Pytest[4]

Para el correcto funcionamiento y *testeo* de la API se ha potado por el uso de la librería **Pytest** y su plugin para Flask[5]. Pues la comodidad que ésta nos ofrecía a la hora de implementarla y el sencillo uso de la misma (todo el test alojado en un mismo archivo), además de su extensa documentación, han resultado fundamentales para su elección[52].

Esta herramienta permite la realización de tests unitarios. Para cada función de la API se ha realizado el siguiente testeo:

- **all_stories_titulo()**. Se cuentan todas las historias existentes en la base de datos y se guarda esa cantidad. Se añade una historia y se comprueba que las historias existentes son la cantidad anterior más 1.
- **user_stories_titulo()**. Se crea un acertijo y un usuario. Se crea la relación (usuario)-[escribe]-(acertijo) y se comprueba que la historia introducida existe y está escrita por ese usuario.
- **soluciones_por_titulo()**.Se crea un acertijo y un usuario. Se crea la relación (usuario)-[propone solución]-(acertijo) y se comprueba que esa solución para ese acertijo existe.

[3]https://docs.python.org/2/library/unittest.html
[4]https://docs.pytest.org/en/latest/
[5]https://pytest-flask.readthedocs.io/en/latest/

- **enviar_comentario()**. Se crea un acertijo y un usuario. Se crea la relación (usuario)-[propone solución]-(acertijo) y se comprueba que esa solución propuesta ha sido añadida en el sistema.
- **enviar_storie()**. Se crea un acertijo y un usuario, se crea la relación (usuario)-[escribe]-(acertijo), y se comprueba que ese acertijo tiene todos los campos iguales que el acertijo creado.
- **cambiar_puntuacion()**. Se crea un acertijo y un usuario. Se crea la relación (usuario)-[propone solución]-(acertijo). Se cambia la puntuación y se comprueba que la puntuación para esa solución propuesta ha sido modificada.
- **acertijo_por_titulo()**.Se crea un acertijo y se comprueba que, buscándolo en la base de datos por su título, éste existe.
- **storie_por_titulo()**. Se crea un acertijo y se comprueba que, buscándolo en la base da datos por su título, la solución a él existe.
- **todo_por_titulo()**. Se crea un acertijo y se comprueba que, a través de su título, se busca en la base de datos y devuelvo todos sus atributos.
- **login()**. Se crea un usuario en el sistema con nombre y contraseña y se comprueba que éste ha sido añadido al sistema correctamente.

Se puede acceder al código de todos los tests definidos en el repositorio de GitHub de la aplicación[6].

6.3. Pruebas en *front-end*

Para la realización de tests unitarios en Javascript existen también infinidad de herramientas. Las más destacables que encontramos son:

- Jest[7]
- VenusJS[8]
- Mocha[9]
- PhantomJS[10]
- Jasmine[11]

[6]https://github.com/guillesiesta/ProjectX/blob/master/stories/test_storie.py
[7]https://jestjs.io/
[8]https://mochajs.org/
[9]https://mochajs.org/
[10]http://phantomjs.org/
[11]https://jasmine.github.io/2.4/introduction.html

Las diferencias entre unos y otros son mínimas. Es por ello que para decidir cual de las herramientas anteriores es la adecuada, se ha accedido a consultar el manual oficial de ReactJS[12] para el *testeo* de una aplicación desarrollada con su propia tecnología.

Una vez consultado en profundidad, se ha decidido que para testear la interfaz se usará Jest[13] junto con Enzyme[14]. Esta combinación de herramientas viene confirmada como la mejor alternativa en la propia documentación de ReactJS[42].

La idea principal para las pruebas es comprobar que cada componente creado en ReactJS se renderiza de manera correcta. Para ello gracias a Enzyme, se puede crear un *snapshot* o instantánea de ese componente antes de renderizar, y una vez se renderice, se comprueba que el componente está creado de la manera esperada[43].

Dicho de otra manera: se le dice a Jest que se quiere estar seguro de que la salida de este componente nunca debe cambiar accidentalmente y Jest, con ayuda de Enzyme la guarda en un archivo, y después comprueba que el componente no ha sido modificado una vez se renderice.

6.4. Validación

Como forma de validación se han creado una serie de usuarios con contraseña para que quién lo desee pueda acceder a la plataforma y comprobar que puede crear un acertijo, responder los ya existentes y puntuar las propuestas de solución que le propongan.

Cada usuario tiene asignado un acertijo creado por defecto y asignado. Todas las contraseñas son iguales: 1234.

Los usuarios son:

- thor
- tonystark
- hulk
- ultron
- spiderman
- aristoteles
- socrates
- platon

[12] https://reactjs.org/docs/test-utils.html
[13] https://jestjs.io/
[14] https://github.com/airbnb/enzyme

- descartes
- epicuro

Se recuerda nuevamente la dirección web de la aplicación:

https://riddling.herokuapp.com/

Actualmente se está llevando a cabo este proceso, sin embargo aún no se han obtenido datos suficientes como para analizar el grado de agrado o desagrado general de la aplicación para los usuarios.

Capítulo 7

Conclusiones y posibles ampliaciones

El objetivo de este proyecto fue que la comunidad obtuviera un beneficio intelectual mediante la creación de una aplicación que pudiera crear una incertidumbre en el ser humano y que éste usara el intelecto para resolverla, para así crujir su rutina y desarrollar ese espíritu inquieto y aventurero que posee nuestra especie.

En resumen, este proyecto es totalmente apto para la **gestión de un juego conversacional**. Gracias la planificación que definió el esquema temporal en el que se desarrollaría el mismo y gracias a la correcta estimación finalización de cada uno de los *sprints*, se puede confirmar que el tiempo fue el necesario para su correcto desarrollo.

Teniendo en cuenta los distintos requisitos recogidos en la sección tal podemos destacar los siguientes puntos:

El diseño modular de la aplicación permite que cada uno de los componentes presente un bajo acoplamiento, permitiendo la reutilización de cada uno de los módulos y, lo que es más interesante, la ampliación de la aplicación para añadir nuevas funcionalidades.

Por otro lado, la metodología y la filosofía de trabajo elegidas fueron acertadas teniendo en cuenta la necesidad de validación continua, y los acotamientos temporales a los que fue sometido el proyecto.

Previo a un estricto proceso de investigación en el que se exploraron las distintas alternativas a aplicar en cada uno de los bloques y su posterior comparativa para así escoger aquella tecnología que mejor se adaptara a nuestro proyecto, nos lleva a que, junto con las tecnologías usadas en cada uno de los bloques o módulos desarrollados, se ha podido desplegar en la nube una aplicación, con una interfaz intuitiva y atractiva que enganche al usuario, (que hemos llamado Riddling, y que se puede acceder a ésta a través de la siguiente dirección: **https://riddling.herokuapp.com/**), de una manera satisfactoria.

Se completaron casi todas las historias de usuario, a excepción de las his-

torias de usuario HU-07 y HU-12+1 que, como se notificó, al ser estas de una prioridad de nivel bajo, esta funcionalidad se añadiría única y exclusivamente en el caso de que sobrara tiempo en el *sprint* correspondiente. Sin embargo, la no implementación de estos requisitos no suponen un inconveniente importante a la hora de la ejecución de la aplicación y su correcto funcionamiento.

Finalmente, se de muestra que un proyecto como este se puede desarrollar con precio asequible y, sobre todo, gracias al uso del software libre, se puede crear una aplicación compleja que reúna distintos tipos de lenguajes, filosofías y tecnologías y, que además entretenga a sus usuarios.

7.1. Posibles ampliaciones

Este proyecto se encuentra en una fase inicial de desarrollo si lo comparamos con cualquier otra aplicación comercial desarrollada por una empresa con recursos ilimitados, sobre todo, recursos tales como dinero y tiempo.

Si se dispusiera de más tiempo y recursos, a la aplicación se le podrían añadir, entre otras tantas funciones, aquellas que la conviertan en una plataforma con perfil de red social orientada a acertijos, por ejemplo:

- Posibilidad de que un usuario modifique sus datos.
- Creación de apartado de relaciones de amistad entre usuarios.
- Sistema de notificaciones al usuario creador de un acertijo en el caso de que se haya propuesto una solución al mismo.
- Mejora del diseño visual de la interfaz.
- Posibilidad de sincronización a los usuarios con su perfil de Facebook y demás redes sociales.
- Creación de temáticas distintas para agrupar los acertijos y así facilitar la búsqueda de los mismos por los usuarios.
- Notificar al usuario que propuso una respuesta a un acertijo de que ésta ha sido puntuada.
- Realización de ránkings en los que mostrar, por ejemplo
  - Acertijos más difíciles.
  - Usuarios con porcentajes más altos de acertijos solucionados.
  - Amigos con porcentajes más altos de acertijos solucionados.
  - Acertijo del mes.

Como se puede observar, las posibilidades de ampliación de este proyecto son inmensas. El límite de éste se encuentra en la imaginación.

Bibliografía

[1] GNU General Public License v3.0,
https://github.com/guillesiesta/ProjectX/blob/master/LICENSE

[2] Computación en la nube, Wikipedia
https://es.wikipedia.org/wiki/Computación_en_la_nube

[3] Computación en la nube, Wikipedia
https://es.wikipedia.org/wiki/Front-end_y_back-end

[4] ¿Qué es una API REST?, Jonathan Ordóñez
https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/

[5] Relationship to the World Wide Web and REST Architectures, W3C Working Group Note 11 February 2004
https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

[6] CHAPTER 5, Representational State Transfer (REST)
https://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm

[7] DevOps Culture (Part 1), May 1, 2012 by John Willis,
http://itrevolution.com/devops-culture-part-1/

[8] The Rise of DevOps, 02 Mar 2010, Dmitriy Samovskiy's Blog,
http://www.somic.org/2010/03/02/the-rise-of-devops/

[9] What is DevOps?,by Mike Loukides, June 7, 2012
http://radar.oreilly.com/2012/06/what-is-devops.html

[10] Desarrollo basado en pruebas, Material docente para Infraestructura Virtual, Juan Julián Merelo Guervós
http://jj.github.io/IV/documentos/temas/Desarrollo_basado_en_pruebas
#gestores-de-versiones-de-lenguajes-y-bibliotecas

[11] DevOps: cómo romper la barrera entre Desarrollo y Operaciones, Atlassian
https://es.atlassian.com/devops

[12] Vayamos al grano, ¿qué es eso de DevOps?, Ana M. del Carmen García Oterino
http://www.javiergarzas.com/2014/12/devops-en-10-min.html

[13] Exploring the ENTIRE DevOps Toolchain for (Cloud) Teams, Richard Seroter on May 28, 2014
https://www.infoq.com/articles/devops-toolchain

[14] Project X , Guillesiesta,
https://github.com/guillesiesta/ProjectX

[15] Memoria_TFG_GuillermoMuriel, Guillesiesta
https://github.com/guillesiesta/Memoria$_{T}FG_{G}uillermoMuriel$

[16] The Relationship Between Dev-Ops And Continuous Delivery: A Conversation With Jez Humble Of ThoughtWorks , Jeffrey Hammond,
https://go.forrester.com/blogs/11-09-09-the_relationship_between_dev_ops_and_continuous_delivery_a_conversation_with_jez_humble_of_thought/

[17] Continuous delivery, wikipedia
https://en.wikipedia.org/wiki/Continuous_delivery

[18] Control de acceso HTTP (CORS), MDN Dev Docs
https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

[19] Python Flask Cors Issue ,stackoverflow
https://stackoverflow.com/questions/28461001/python-flask-cors-issue

[20] Flask-CORS
https://flask-cors.readthedocs.io/en/latest/

[21] Guía Hays del mercado laboral 2018
http://guiasalarial.hays.es/trabajador/info-sector

[22] Coste Indirecto Proyecto UGR
http://investigacion.ugr.es/pages/proyectosja/2018/ayuda

[23] Modelo–vista–controlador, Wikipedia
https://es.wikipedia.org/wiki/Modelo-Vista-Controlador

[24] Simple Example of MVC (Model View Controller), Design Pattern for Abstraction
https://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design

[25] Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC), by Steve Burbeck
https://web.archive.org/web/20120429161935/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[26] Bases de datos ¿qué son? ¿qué tipos existen? Lo que necesitas saber como profesional, Maldeadora
https://platzi.com/blog/bases-de-datos-que-son-que-tipos-existen/

[27] Tipos de bases de datos no relacionales, Wikipedia
https://es.wikipedia.org/wiki/NoSQL

[28] Global Graph Database Market is Expected to To Reach Around USD 5.6 Billion by 2024, Allan , Sep 3, 2018
https://zmrnewsjournal.us/1265/global-graph-database-market-is-expected-to-to-reach-around-usd-5-6-billion-by-2024/

[29] Global Graph Database Market 2018 – 2025 triAGENS GmbH(Arango DB), Neo4j, Inc, OrientDB Ltd, Cayley, Titan, MarkLogic, johnsgaston, September 5, 2018
http://lakeviewgazette.com/2018/09/05/global-graph-database-market-2018-2025-triagens-gmbharango-db-neo4j-inc-orientdb-ltd-cayley-titan-marklogic/

[30] anybody tried neo4j vs titan - pros and cons [closed], StackOverflow
https://stackoverflow.com/questions/17269306/anybody-tried-neo4j-vs-titan-pros-and-cons

[31] What are the best programming languages for building APIs?, Antonio Cucciniello , June 11, 2017
https://hub.packtpub.com/what-are -best-programming-languages-building-apis/

[32] What Languages Should Your API Helper Libraries Support?, Bill Doerrfeld November 12,2015
https://nordicapis.com/what-languages-should-your-api-helper-libraries-support/

[33] What Languages Should Your API Helper Libraries Support?, Bill Doerrfeld November 12,2015
https://nordicapis.com/what-languages-should-your-api-helper-libraries-support/

[34] Flask vs Django, Scott Robinson, September 21, 2017
https://stackabuse.com/flask-vs-django/

[35] Los 5 mejores frameworks de JavaScript en 2017, CampusMVP
https://www.campusmvp.es/recursos/post/los-5-mejores-frameworks-de-javascript-en-2017.aspx

[36] JavaScript Framework Guide AND Top 15 JS Frameworks of 2019 Prediction, Gurdeep Singh, Friday, June 15, 2018
https://appinventiv.com/blog/javascript-framework-guide

[37] ¿Cuál es la diferencia entre IaaS, SaaS y PaaS?, Scott Carey
http://www.ciospain.es/gobierno-ti/ cual-es-la-diferencia-entre-iaas-saas-y-paas

[38] Cloud Computing, Wikipedia
https://en.wikipedia.org/wiki/Cloud_computing

[39] IaaS, PaaS y SaaS, ¿cuáles son sus diferencias?, Angel Carrero, 22 julio, 2016
https://revistacloud.com/iaas-paas-y-saas-cuales-son-sus-diferencias/

[40] Servicios en la nube: diferencias entre IaaS, SaaS y PaaS, T. Mora on 2 noviembre 2017
http://blog.neteris.com/stepforward/servicios-en-la-nube-diferencias-entre-iaas-saas-y-paas

[41] Cliente Server, Wikipedia
https://en.wikipedia.org/wiki/Client_server_model

[42] Testing, ReactJS
https://reactjs.org/community/testing.html

[43] Testing React components with Jest and Enzyme, Artem Sapegin
https://hackernoon.com/testing-react-components-with-jest-and-enzyme-41d592c174f

[44] 2.2.1. Debian , Neo4j Operations Manual
https://neo4j.com/docs/operations-manual/current/installation/linux/debian/

[45] Flask, Quickstart
http://flask.pocoo.org/docs/0.12/quickstart/

[46] Tutorial: Intro to React, ReactJS
https://reactjs.org/tutorial/tutorial.html

[47] Building a Python Web Application Using Flask and Neo4j, Nicole White
https://neo4j.com/blog/building-python-web-application-using-flask-neo4j/

[48] Managing Databases, GrapheneDB
https://docs.graphenedb.com/docs/managing-databases

[49] Getting Started, Zeit.co
https://zeit.co/now

[50] The Heroku CLI
https://devcenter.heroku.com/articles/heroku-cli

[51] Deploy Your React App To Heroku, Shakhor Smith, Mar 23
https://dev.to/smithmanny/deploy-your-react-app-to-heroku-2b6f

[52] pytest-vs-unittest, Renzon
https://github.com/renzon/pytest-vs-unittest

[53] Heroku vs Openshift: Which is a Better PaaS?, Kitty Gupta
https://www.freelancinggig.com/blog/2017/05/31/heroku-vs-openshift-better-paas/

[54] Navarro Cadavid, A., Fernández Martínez, J., Morales Vélez, J. "Revisión de metodologías ágiles para el desarrollo de software". *PROSPECTIVA*, vol. 11. no.2, pp. 30-39. julio - diciembre 2013.