Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑERÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Aplicación Web para Aprender a Programar Usando a Librería de Google Blockly

Estudante: Braulio Méndez Agra
Director: Manuel Álvarez Díaz

A Coruña, 5 de setembro de 2019.

Dedicatoria: a todos os que me animaron e deron forzas para rematar este proxecto.

Agradecementos

Aos meus familiares, sobre todo a meus pais, por todo o apoio recibido para seguir avanzando.

A todos os amigos e amigas cos que compartín estes anos de estudo e que fixeron desta etapa unha moi boa experiencia.

Ao meu titor, Manuel Álvarez, por toda a axuda recibida para levar a cabo este proxecto e pola paciencia ata o último día.

Resumo

Neste proxecto búscase brindarlle a calquera persoa a posibilidade de aprender as bases da programación mediante o desenvolvemento dunha aplicación web que lle permitirá familiarizarse cos elementos comúns na maioría das linguaxes de programación, amais de poder compartir e consultar os traballos realizados por outros usuarios.

Dentro da aplicación, os usuarios poden arrastrar e encadear diferentes *pezas* que representen valores, asignacións, operadores, bucles e outros elementos de programación, chegando a formar un pequeno programa que poidan executar e ver a súa saída por pantalla. Tamén poden visualizar a modo ilustrativo como se traduciría, a linguaxes de programación reais como Python ou JavaScript, o programa que acaban de crear. Para implementar os aspectos visuais dun editor que permita estas operacións faise uso da biblioteca Blockly. Ademais, un usuario pode rexistrarse na aplicación, o cal lle permite gardar eses programas e modificalos nun futuro, así como compartilos con outros usuarios e poder aprender dos traballos doutras persoas. O usuario poderá establecer diferentes permisos aos seus traballos para restrinxir que usuarios poden acceder a eles.

Aínda que a aplicación tamén pode ser usada por calquera persoa individualmente, ten un claro carácter educativo e os usuarios poden diferenciarse entre *profesores* e *alumnos*. Un usuario co rol *profesor* pode crear grupos nos que incluirá *alumnos* aos que lles propoñerá exercicios que eles poderán resolver. Máis tarde, o *profesor* poderá revisar as solucións dadas e dentro de cada grupo tamén dispoñen dun chat para comunicarse entre os seus membros.

Palabras chave:

- Aplicación Web
- Educación
- Programación
- Aprendizaxe
- Social
- Angular
- Blockly

Keywords:

- Web Application
- Education
- Programming
- Learning
- Social
- Angular
- Blockly

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

Dende hai unhas poucas décadas, o desenvolvemento software foi adquirindo cada vez máis protagonismo e o seu crecemento foi exponencial ata chegar ao punto de converterse a día de hoxe en imprescindible para calquera empresa, institución, goberno ou persoa na súa vida cotiá e con toda seguridade seguirá sendo así no futuro. Tal e como xa acontece na actualidade, as persoas cualificadas neste campo serán moi demandadas para traballar en todo tipo de áreas dando solucións aos problemas que xurdan e mellorando a vida da xente en todo o mundo.

Máis alá de adicarse a esto profesionalmente, o feito de coñecer as bases da programación e poder darlle instrucións a unha máquina para que faga o que nós desexamos, axuda indubidablemente no día a día das persoas mellorando as súas capacidades para razoar e solucionar problemas. Trátase duns coñecementos prácticos moi beneficiosos sobre todo para a xente nova na súa etapa de formación, para que poidan descubrir se lles gusta, perderlle o medo e adquirir novas habilidades e capacidades.

1.1 Determinación da Situación Actual

Na actualidade, se unha persoa desexa aprender a programar, ademais dos coñecementos teóricos que deberá adquirir, precisa practicar por el mesmo as leccións aprendidas e poñelas en práctica creando os seus propios programas, o cal o obriga a escoller unha determinada linguaxe na que iniciarse. Esto pode ter as súas dificultades xa que comezar directamente escribindo código poder ser complicado nun principio e programar nunha linguaxe en concreto fará que teña que aprender as súas peculiaridades sintácticas. Tendo esto en conta, sería de gran axuda dispoñer, nas etapas iniciais da aprendizaxe, dunha ferramenta para poder practicar sen ter que profundizar nunha linguaxe concreta, abstraéndose dos detalles da programación e que sexa amigable e entretido para o usuario.

Dende o punto de vista da docencia, unha ferramenta deste tipo tamén pode facilitar a

tarefa do ensino das bases da programación, facendo máis sinxela a introdución a este mundo e reducindo a curva de aprendizaxe. O profesor podería centrarse nos aspectos básicos sen adentrarse en detalles de implementación dunha determinada linguaxe e sería interesante que puidera aproveitar dita ferramenta para encargarlle exercicios aos seus alumnos e avalialos.

Para chegar ata este obxectivo, existen algunhas ferramentas que conseguen esa abstracción, respecto das linguaxes de programación, por medio de pezas que se poden encadear e agrupar formando secuencias executables e facilmente entendibles. As máis coñecidas son Scratch ou Blockly, empregadas noutras aplicacións web cun carácter similar a este aínda que, como se detalla máis en profundidade no capítulo 3 de Estado do Arte, sen cumplir todos os requisitos relativos á docencia nas aulas e á socialización entre usuarios que se piden para este proxecto.

1.2 Alcance e Obxectivos

Este proxecto ten como finalidade desenvolver unha aplicación web para ofrecer, dun xeito doado e entretido, a oportunidade de iniciarse no mundo da programación. Está orientada ao ámbito educativo, proporcionando ferramentas para o seu uso en clase por parte dun profesor e os seus alumnos pero tamén pode ser empregada por usuarios correntes de maneira individual que poden decidir facer os seus traballos públicos e aprender doutras persoas.

A aplicación usa unha biblioteca de JavaScript chamada *Blockly*[1], que permite crear un editor de programación mediante elementos visuais os cales se poden arrastrar e encadear formando pequenos programas. Estes elementos representan *pezas* de pseudocódigo como valores, asignacións, listas ou iteradores, que se deben colocar na orde e forma correcta para crear o algoritmo que se desexe programar e que despois se poderá executar e traducir a código en linguaxes reais. A partir desta función, desenvolveuse a funcionalidade necesaria para permitir a xestión de ditos programas por parte dos usuarios que se rexistren e acceder tamén a traballos doutras persoas así como compartir os seus propios permitindo valoracións e comentarios. A maiores existen opcións que permiten a un *profesor* encargarlles traballos aos seus *alumnos*. Dito *profesor* é quen de crear grupos, encargar exercicios e comunicarse con eles por medio dun chat.

En canto ao desenvolvemento da aplicación, preténdese que o seu deseño esté baseado en capas. Deste xeito, as distintas partes do sistema contarán cunha maior independencia, sen estar acopladas unhas a outras e serán máis facilmente extensibles favorencendo a súa posible evolución nun futuro.

O obxectivo que se persegue é dar soporte á funcionalidade requerida para poder aprender os conceptos básicos da programación permitindo crear programas propios nos que poder traballar de xeito continuado e interactuar con outros usuarios para aprender con eles.

Capítulo 2

Base Tecnolóxica

A continuación introdúcense as linguaxes e tecnoloxías relevantes que se empregaron neste proxecto.

2.1 Linguaxes

Deseguido detállanse as linguaxes utilizadas no desenvolvemento da aplicación:

- **Java**: trátase dunha linguaxe orientada a obxectos, aínda que non de forma pura xa que tamén contén tipos primitivos. Foi desenvolto coa intención de que os programadores poidan *escribir unha vez e executar en calquera plataforma* grazas á máquina virtual de Java (JVM) que pode executar código compilado (bytecode) en calquera computadora que dispoña dunha JVM [2].

- **JavaScript**: é unha linguaxe de programación interpretada que inclúe funcións da orientación a obxectos que permite executar lóxica complexa en páxinas web de forma que o usuario pode interactuar coa páxina sen necesidade de que esta se comunique co lado servidor creando así páxinas dinámicas [3].

- **TypeScript**: é un superconxunto tipado de JavaScript que compila a esta linguaxe engadindo tipos estáticos e obxectos baseados en clases pensado para desenvolver a nivel de aplicación [4].

- **HTML**: trátase da linguaxe máis común para elaborar páxinas web. Realmente é unha linguaxe de marcado coa que se define a estrutura do sitio web para mostralo no navegador [5].

- **CSS**: utilízase para describir a presentación de documentos HTML especificando como se deben mostrar os aspectos visuais da páxina, engadindo estilos e o deseño [6].

2.2 Frameworks e Bibliotecas

Neste apartado coméntase as ferramentas software que fixeron posible o desenvolvemento da aplicación.

2.2.1 Core

No núcleo da aplicación faise uso dos seguintes frameworks:

- **Maven**, unha ferramenta software para a xestión de proxectos Java. Baséase nun arquivo chamado *pom.xml* no que garda a información do proxecto, e da súa configuración e partir do cal pode construír o proxecto e xerar documentación. Grazas a este mecanismo fai máis doado e uniforme o proceso de contrución [7].
- **Spring Boot**, unha tecnoloxía que permite crear unha aplicación de forma rápida coa mínima configuración inicial de *Spring* axilizando os pasos de creación (e.g. usando Maven) e despregue [8]. **Spring** é un framework para o desenvolvemento de aplicacións e contedor de inversión de control para Java que proporciona todo o preciso para esta linguaxe nunha contorna empresarial, e permite centrarse na aplicación que se está levando a cabo sen preocuparse tanto da configuración [9].

2.2.2 Web

Na capa web utilízanse os seguintes frameworks e bibliotecas:

- **Angular**: un framework para construír aplicacións web de lado cliente en TypeScript e HTML [10].
- **Bootstrap**: unha ferramenta software para axudar a definir a interface dunha páxina web. Trátase dunha combinación de HTML, CSS e JavaScript [11].
- **Ngx-toastr**: un paquete para Angular que serve para mostrar notificacións emerxentes na propia web para poder ofrecerlle feedback ao usuario [12].
- **Font Awesome Icons**: un conxunto de ferramentas baseados en CSS que, entre outros recursos, proporciona gran cantidade de iconas [13].
- **Google Blockly**: unha biblioteca de JavaScript para engadir un editor de programación con elementos visuais a unha páxina web ou aplicación móbil [1].
- **Balsamiq**: unha ferramenta software para realizar esbozos da interface, sen necesidade de programar a lóxica [14].

2.2.3 Probas

Para a implementación das probas utilízanse as seguintes ferramentas:

- **JUnit4**: un framework para a automatización de probas unitarias e de integración en aplicacións Java [15].
- **JaCoCo**: unha biblioteca de código aberto para Java que se usa para medir a cobertura das probas no código [16].
- **Postman**: é unha ferramenta que se emprega para probar unha API REST [17].

2.3 Protocolos

Os protocolos empregados na aplicación son:

- **HTTP**, un protocolo a nivel de aplicación o cal permite a transmisión de datos a través da rede seguindo un modelo cliente-servidor [18].
- **JDBC**, unha especificación dunha interface de programación de aplicacións (API) estándar para aplicacións Java que permite realizar operacións sobre unha base de datos independentemente del cal sexa esta e do sistema operativo en concreto que se esté a usar [19].

2.4 Ferramentas de Desenvolvemento

Estas son as ferramentas empregadas durante o desenvolvemento do proxecto:

- **Eclipse** é un IDE (Entorno de Desenvolvemento Integrado) de código aberto para desenvolver principalmente en Java aínda que tamén permite outras linguaxes. Ten multitude de posibilidades para facilitar e mellorar o desenvolvemento e a súa funcionalidade pode ser extendida instalando *plug-ins* adicionais [20].
- **Visual Studio Code** é un editor de código de fonte desenvolto por Microsoft cunha gran cantidade de funcionalidades para o desenvolvemento software que tamén permite engadir complementos [21].
- **Git** é un sistema de control de versións distribuído *open source* pensado tanto para pequenos como para grandes proxectos destacando pola súa eficiencia e velocidade [22].
- **Angular CLI** é unha interface por liña de comandos que se pode usar para inicializar, desenvolver e manter aplicacións feitas en Angular [23].

2.5 Servidores de Aplicacións

Na fase de desenvolvemento:

- Na parte do servidor: Apache Tomcat [24] embebido no framework Spring Boot, unha implementación de código aberto das funcionalidades proporcionadas por Java para manexar un servidor web.
- Para a capa web: Angular CLI[23] prové un servidor incorporado que se arranca executando simplemente o comando *ng serve*.

En produción:

- En *backend*, para servir a aplicación tamén se empregará un servidor Apache Tomcat.
- En *frontend*, despois de compilar o proxecto, contamos cos arquivos necesarios para o despregue en calquera servidor capaz de servir arquivos HTML e JavaScript.

2.6 Sistemas de Xestión de Bases de Datos

Como Sistema de Xestión de Base de Datos utilízase MySQL[25]. Este sistema de código aberto é un dos máis utilizados e recoñecidos actualmente, que destaca pola súa velocidade, seguridade e facilidade de instalación, permitindo así o seu correcto funcionamento en computadoras sen un software ou hardware de alto rendemento. *MySQL* é especialmente popular para o seu uso na web e admite JDBC como interface de acceso.

Capítulo 3

Estado do Arte

NA actualidade podemos atopar moitas aplicacións web que teñen como obxectivo a introdución no mundo da programación de persoas sen experiencia previa. Algunhas delas, ao igual que este proxecto, non requiren iniciarse nunha linguaxe concreta senón que contan cun modelo máis abstracto e sinxelo de entender ou mesmo fan uso da biblioteca Blockly[1], ao igual que neste caso. A continuación, coméntanse algunhas das aplicacións con similitudes á proposta neste proxecto:

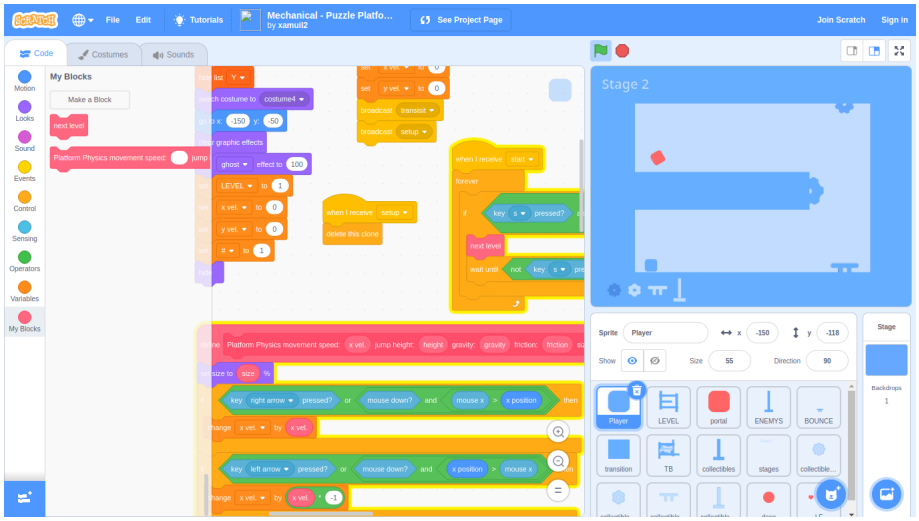


Figura 3.1: Captura de pantalla de Scratch.

https://scratch.mit.edu/: aínda que esta aplicación web non usa Blockly, trátase dunha das máis coñecidas neste ámbito e o sistema usado é semellante posto que tamén emprega elementos visuais que se poden arrastrar e encadear formando secuencias executables. O próposito nesta web, orientada sobre todo á xente nova, é o de crear historias, xogos ou animacións programando os movementos, sons, imaxes, etc. A figura 3.1 correspóndese coa interface desta aplicación [26].

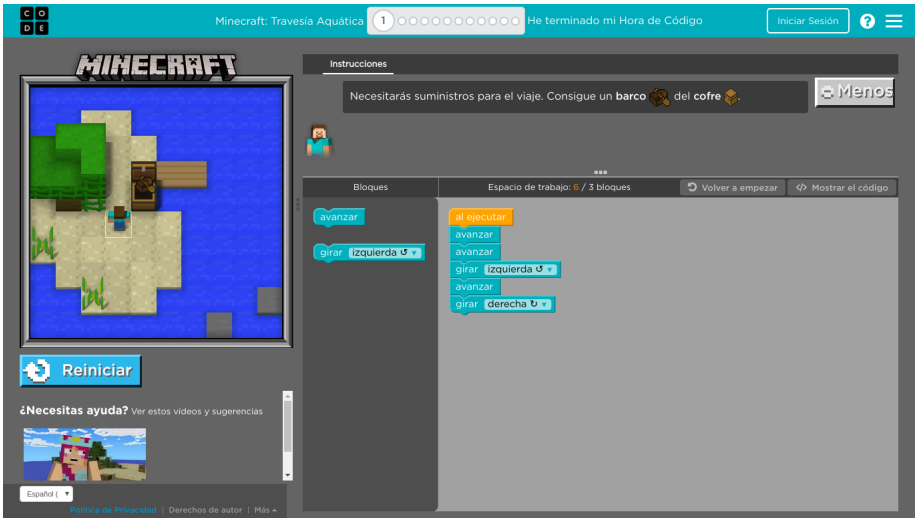


Figura 3.2: Captura de pantalla de code.org

https://code.org: é unha organización que promove a ensinanza das ciencias da computación aos nenos. Conta con multitude de cursos nos que propoñen a realización de actividades para aprender as bases da computación e tamén o desenvolvemento de pequenos proxectos, como a creación de xogos, por medio da biblioteca Blockly (como se aprecia na figura 3.2), a cal usan para levar a cabo os traballos [27].

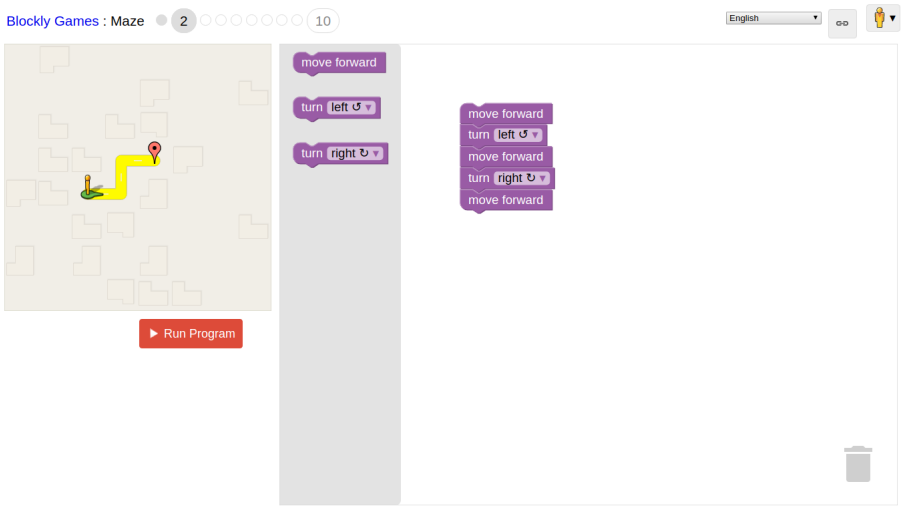


Figura 3.3: Captura de pantalla de Blockly Games

https://blockly-games.appspot.com: na figura 3.3 vese un dos xogos para resolver mediante a biblioteca Blockly cos que conta esta aplicación deseñados para nenos sen experiencia previa en programación [28].

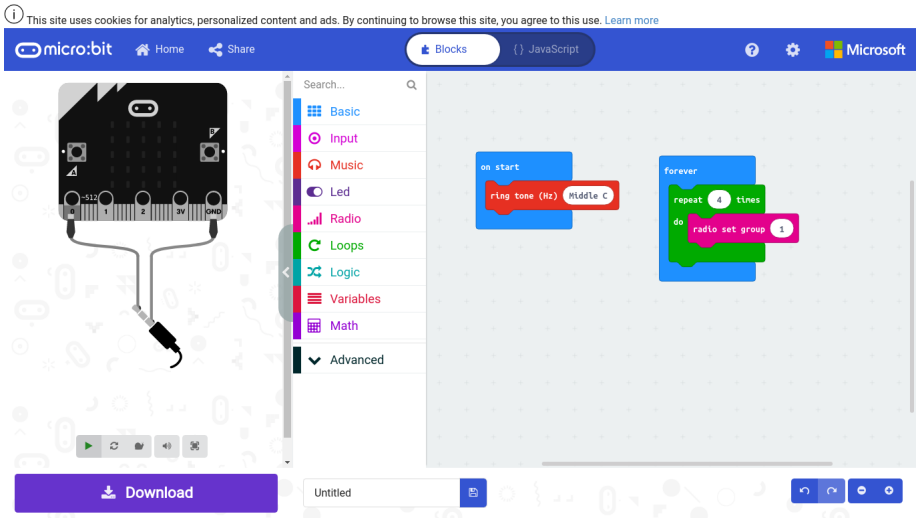


Figura 3.4: Captura de pantalla de microbit.org

https://microbit.org: nesta páxina, á que corresponde a figura 3.4, ofrecen un pequeno ordenador programable chamado "micro:bit" deseñado para facer a aprendizaxe máis doada e divertida e para programar esta pequena computadora faise uso da biblioteca Blockly. Neste sitio web tamén ofrecen recursos de cara a formación nas aulas para profesores (para escolas no Reino Unido) [29].

Na seguinte táboa móstranse algunhas das funcionalidades que se ofrecen dentro das propias aplicacións citadas anteriormente:

Aplicación	Compoñente social (Comentarios, explorar...)	Permite crear grupos dentro da aplicación	Dispoñible para calquera	Gratuíta
Scratch	Si	Non	Si	Non
Code.org	Si	Si	Non	Non
Blockly games	Non	Non	Si	Si
Microbit	Non	Non	Non	Non
Social Blockly	Si	Si	Si	Si

Táboa 3.1: Comparativa de Aplicacións Web.

Respecto da táboa anterior, no presente proxecto preténdese satisfacer todas esas funcionalidades xa que se pode practicar de xeito individual sen depender de terceiras persoas ou institucións así como explorar os traballos doutros usuarios e interactuar con eles. Tamén se ofrece a posibilidade, de forma integrada na aplicación, de crear grupos para que un *profesor*

poida encargar exercicios, consultar os traballos feitos polos *alumnos* e comunicarse con eles por medio dun chat. Non está pensada para un sector xeográfico nin de idade en concreto, ao contrario, a intención é que calquera persoa que o desexe poda usala e aprender libremente.

Capítulo 4

Análise de Viabilidade

PARA a consecución deste proxecto coméntase a continuación o estudo da súa viabilidade dende o punto de vista económico, temporal e tecnolóxico.

- O aspecto **económico**, dado que se trata dun proxecto de fin de grado, non é un parámetro importante para ter en conta. O gasto redúcese ao da luz eléctrica consumida ao longo do tempo de desenvolvemento e á conexión a internet. A maiores só é preciso un ordenador persoal co que xa se contaba e non se necesita alugar ningún local porque se pode realizar o traballo dende a propia casa polo que este aspecto non supón ningún inconveniente para o desenvolvemento do traballo.

- En canto ao **tempo** dispoñible, novamente por ser un traballo de fin de grado, o proxecto está debidamente dimensionado para o tempo do que se dispón ata a súa entrega final. A funcionalidade a desenvolver está pensada para ser realizada no tempo dispoñible tendo en conta as demais asignaturas que se cursan e non foi preciso descartar ningunha parte importante respecto da idea inicial que se puido adaptar sen problema. Por esta razón, o tempo do que se dispón é suficiente e non supón ningún problema para levar a cabo a aplicación.

- Respecto da **tecnoloxía**, para ofrecerlle a posibilidade ao usuario de programar cun sistema baseado no encadeamento de pequenos bloques para formar sinxelos programas utilízase unha das ferramentas máis coñecidas neste ámbito, a biblioteca Blockly[1]. Trátase dunha ferramenta de código aberto usada en moitos proxectos e que conta co respaldo dunha empresa mundialmente coñecida como é Google. En canto á implementación do proxecto, empréganse dúas ferramentas de desenvolvento utilizadas por miles de persoas en todo o mundo, Eclipse[20] e Visual Studio Code[21] e tamén dous frameworks probados con éxito en multitude de proxectos como son Spring Boot[8] para o *backend* e Angular[10] para o *frontend*. Como Sistema de Xestión de Bases de Datos utilízase unha das ferramentas máis empregadas neste ámbito na web, MySQL[25]. Por

todo isto, as tecnoloxías usadas están ampliamente probadas en multitude de escenarios polo que non supoñen ningún risco para o proxecto.

En conclusión, ningún destes tres factores analizados resulta crítico nin fai pensar que o proxecto non se poida levar a cabo polo que a súa finalización debería ser posible de forma satisfactoria en tempo e forma.

Capítulo 5

Introdución ao Desenvolvemento Realizado

5.1 Introdución

Esta aplicación divídese en dúas partes ben diferenciadas, por un lado temos a parte do servidor e por outro a capa web, ademais dunha base de datos relacional para almacenar a información. Obedece a unha arquitectura cliente-servidor na que o usuario interactúa co *frontend* e esta fai as peticións necesarias ao *backend* o cal consulta ou modifica a información na BBDD. No apartado seguinte xa se detallan as tecnoloxías empregadas en profundidade, pero cabe destacar o máis relevante que é o uso de Angular[10] para desenvolver a parte cliente, Spring Boot[8] para a parte servidor e o uso dunha base de datos MySQL[25]. Tamén é importante o uso da biblioteca Blockly[1], compoñente principal da aplicación na capa web para que os usuarios poidan levar a cabo un dos obxectivos principais da aplicación.

5.2 Tecnoloxías

Nesta sección coméntanse as tecnoloxías empregadas no desenvolvemento do proxecto en base ao diagrama de arquitectura da aplicación que se mostra na figura 5.1.

Comezamos o repaso das tecnoloxías aplicadas no desenvolvemento do proxecto pola parte servidor da aplicación. O IDE empregado para o desenvolvemento desta parte foi Eclipse[20] e para poñer en funcionamento a capa de *backend* faise uso dun servidor Tomcat embebido en Spring boot.

Para a construcción deste módulo empregouse a ferramenta software Maven[7], deseñada para a xestión, empaquetado e construcción de proxectos Java, como é o caso. Como se indica no diagrama, esta parte implementouse usando o framework Spring Boot[8] para desenrolar toda a lóxica do sistema. En primeiro lugar, impleméntanse as entidades que modelan toda

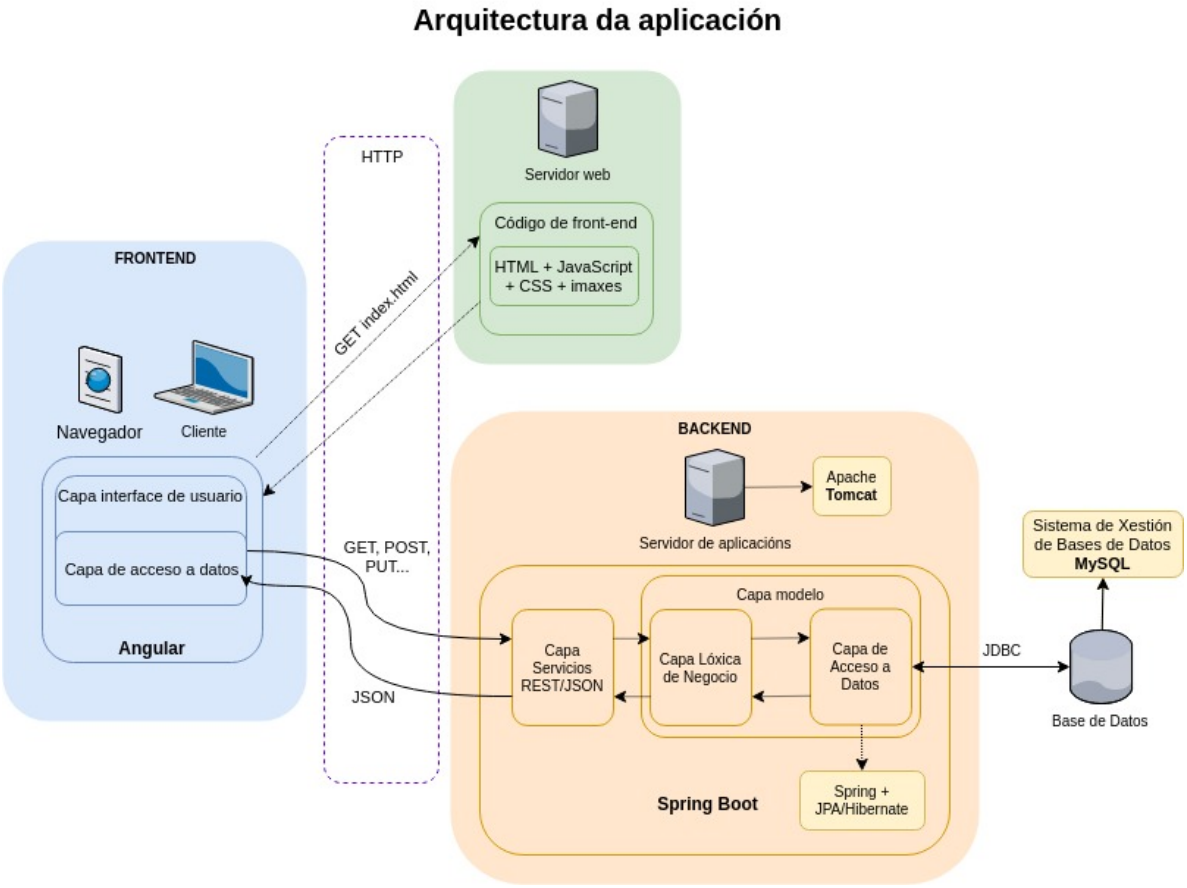


Figura 5.1: Arquitectura da Aplicación.

a información que vai manexar a aplicación relativa aos usuarios da aplicación, programas e grupos e que se vai persistir na Base de Datos a través dos DAOs (Obxectos de Acceso a Datos), clases que se encargan de abstraer o acceso á fonte de datos. A BBDD empregada é MySQL[25] e accedése a ela por medio da API JDBC[19]. O framework Spring Boot é o que se encarga desta comunicación co Sistema de Xestión de Base de Datos e mesmo ofrece implementación para operacións fundamentais sobre as entidades como creación, lectura, borrado ou actualización. Para operacións máis complexas sobre a BBDD foi necesario realizar implementacións adicionais coas consultas específicas pertinentes.

Apoiada na API de persistencia desenvolveuse a capa da lóxica de negocio, onde se implementan todas as operacións que a aplicación vai poder realizar e que traballan directamente cos DAOs. Tal e como se ve na imaxe, a esta capa non se accede directamente dende o exterior, senón que se desenvolve outra que proporciona un grao máis de abstracción, que recibirá as peticións e se comunicará co servizo. O feito de realizar esta abstracción sobre esta capa

permite realizar outras implementacións distintas que accedan a ela nun futuro, se fora necesario. Esta última capa que recibe as peticións é unha API REST, isto é, unha interface entre sistemas que se comunican mediante o protocolo HTTP[18]. Está formada por diferentes controladores, divididos segundo a funcionalidade que manexan, que reciben as peticións realizadas por un cliente e acceden á capa servizo esperando unha reposta que devolven contida nun DTO (Obxecto de Transferencia de Datos) que recolle a información requerida no formato correcto. Para a realización das probas de integración da capa servidor utilizouse o framework JUnit4[15].

Deseguido explícase o desenvolmento realizado na parte cliente. Levouse a cabo utilizando o framework para aplicacións web Angular[10] e coa axuda da ferramenta Angular CLI[23] que fai máis doada a xestión de aplicacións Angular facilitando tarefas como a inicialización, a xeración de novos compoñentes ou a execución da aplicación. O IDE empregado nesta parte foi Visual Studio Code[21]. Como se aprecia na figura 5.1, o cliente obtén os recursos do *frontend* e despois serán estes os que se comuniquen coa API REST para realizar as operacións necesarias en *backend*.

A capa web, seguindo a arquitectura de Angular, está composta de compoñentes. Cada compoñente define unha clase que contén información e lóxica, ademais dun modelo HTML asociado e opcionalmente unha folla de estilos CSS. A lóxica do compoñente codifícase en TypeScript e nalgún caso concreto fíxose uso de JavaScript.

Para mellorar a apariencia da interface web usouse a biblioteca Bootstrap[11] para os estilos e Font Awesome Icons[13] para as iconas. A maiores dos compoñentes e dos arquivos de configuración, tamén existen "servizos" que foron empregados para abstraer as operacións que realizan as peticións mediante HTTP á capa de *backend*. Co obxectivo de proporcionarlle feedback ao usuario nos casos de éxito ou de erro respecto das operacións que leve a cabo móstranse notificacións con texto e cor informativos grazas ao paquete de Angular ngx-toastr[12].

Por último en canto á capa web, para a realización dos programas por parte dos usuarios úsase a biblioteca Blockly[1] a cal se integra dentro dun compoñente que despois se pode reutilizar dentro doutros.

Para todo o proxecto, como sistema de control de versións utilizouse Git[22].

5.3 Metedoloxía e Iteracións

A metodoloxía empregada foi PUDS (Proceso Unificado de Desenvolvemento Software) polo que o proceso de desenvolvemento estivo orientado por Casos de Uso e levouse a cabo

de xeito iterativo e incremental. Para o modelado dos diagramas do proxecto fíxose uso de UML[30]. O traballo dividiuse en tres iteracións nas que cada unha engadía máis funcionalidade á anterior.

- Na primeira iteración, implementouse a arquitectura básica da aplicación, tanto *backend* coma *frontend* e a autenticación e rexistro de usuarios.
- Na segunda iteración, integrouse Blockly na aplicación e creáronse as funcionalidades necesarias para manexar programas usando esta biblioteca. Tamén se implementaron as operacións precisas para crear grupos e xestionar os seus membros.
- Na terceira iteración, levouse a cabo a parte social da aplicación como é engadir a posibilidade de establecer diferentes niveis de privacidade para os programas, poder engadirlles comentarios e valoracións, explorar traballos doutros usuarios, chat en cada grupo, etc. Tamén se agregou a posibilidade de crear exercicios dentro dun grupo e implementáronse as medidas básicas de seguridade para acceder aos diferentes recursos da aplicacións segundo as operacións que cada usuario ten permitido realizar.

Capítulo 6

Planificación e Análise de Custos

Na **planificación inicial** do proxecto decidiuse que o seu desenvolvemento estaría dividido en tres iteracións, tal e como se indica na sección 5.3, comezando polas funcionalidades básicas e rematando coas secundarias. Ademais destas, tívose en conta a fase inicial de análise de requisitos e formación nas tecnoloxías que se van a empregar, a cal denominamos "Iteración 0". Cada unha das outras tres debía contar coas fases de análise, deseño, implementación e probas, e nas tres se traballaría tanto no *backend* coma no *frontend*. Para cada iteración, a planificación foi a seguinte:

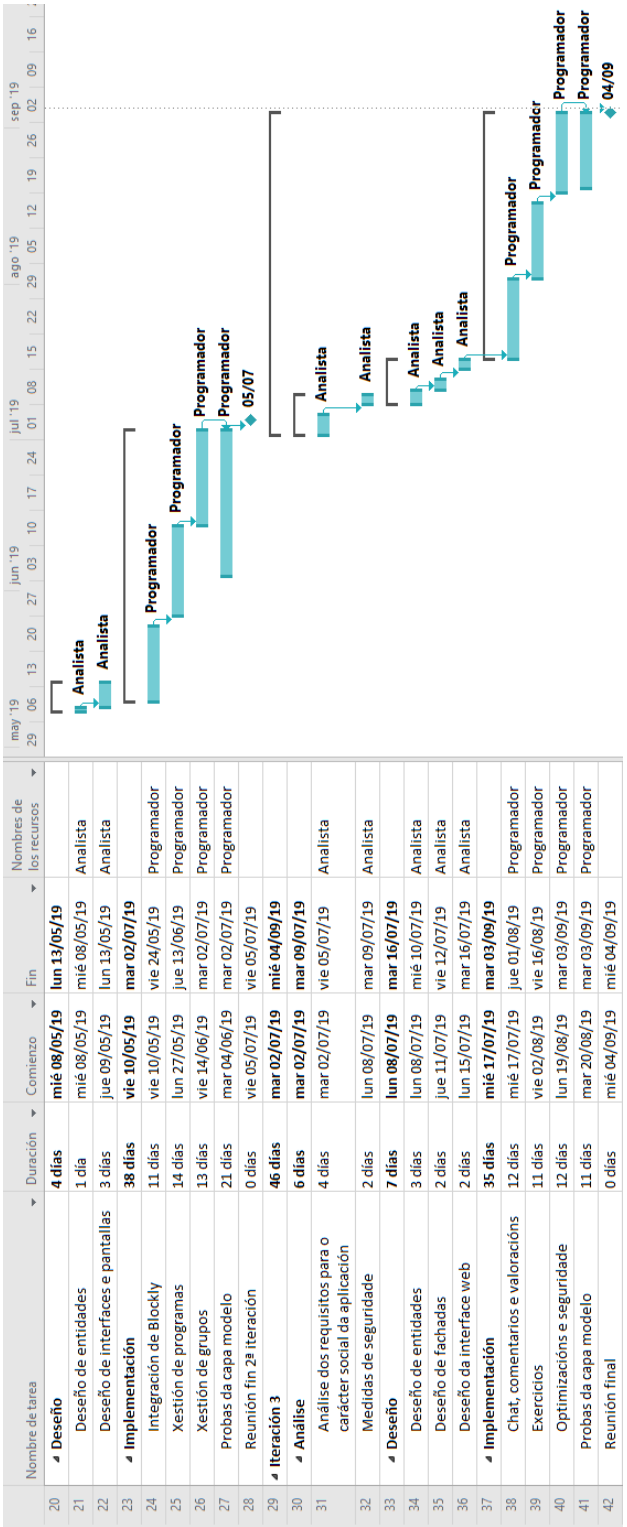
- Na fase inicial ou "**iteración 0**", analizar os requisitos de toda a aplicación en xeral e definir a funcionalidade a desenvolver. Familiarizarse cos frameworks e bibliotecas cos que se vai traballar.

- Na **primeira** iteración, implementar a estrutura básica da aplicación, tanto de *backend* como de *frontend*, con Spring Boot e Angular respectivamente. Desenvolver as funcionalidades de rexistro e inicio de sesión de usuarios, así como as operacións relativas á visualización e actualización dos datos do perfil. Tamén se levaron a cabo as probas de integración pertinentes para as funcións implementadas.

- Na **segunda** iteración, o traballo céntrase no desenrolo das funcionalidades que permitirán a xestión de programas (creación, actualización, borrado, etc) e a integración da biblioteca Blockly para dar forma a ditos programas. Tamén, realizar a parte correspondente ao manexo de grupos, ofrecendo a posibilidade de crealos, editalos, engadir e eliminar membros, etc. Para todas as operacións desenvoltas, realizar as probas necesarias para asegurar o seu correcto funcionamento.

	Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
1	**Iteración 0**	**12 días**	**jue 21/03/19**	**vie 05/04/19**	
2	Especificación de requisitos	9 días	jue 21/03/19	mar 02/04/19	Analista
3	Formación nas tecnoloxías	3 días	mié 03/04/19	vie 05/04/19	Programador
4	**Iteración 1**	**19 días**	**lun 08/04/19**	**vie 03/05/19**	
5	**Análise**	**4 días**	**lun 08/04/19**	**jue 11/04/19**	
6	Estudar requerimentos sobre funcións do usuario	4 días	lun 08/04/19	jue 11/04/19	Analista
7	**Deseño**	**3 días**	**vie 12/04/19**	**mar 16/04/19**	
8	Deseño de entidades	3 días	vie 12/04/19	mar 16/04/19	Analista
9	Deseño de DAOs e servizos	2 días	lun 15/04/19	mar 16/04/19	Analista
10	**Implementación**	**13 días**	**mar 16/04/19**	**jue 02/05/19**	
11	Estrutura básica da aplicación	3 días	mar 16/04/19	jue 18/04/19	Programador
12	Servizo de autenticación	6 días	vie 19/04/19	vie 26/04/19	Programador
13	Xestión de datos do perfil	4 días	lun 29/04/19	jue 02/05/19	Programador
14	Probas da capa modelo	6 días	jue 25/04/19	jue 02/05/19	Programador
15	Reunión fin 1ª iteración	0 días	vie 03/05/19	vie 03/05/19	
16	**Iteración 2**	**46 días**	**jue 02/05/19**	**vie 05/07/19**	
17	**Análise**	**5 días**	**jue 02/05/19**	**mié 08/05/19**	
18	Análise dos casos de uso para os programas	2 días	jue 02/05/19	vie 03/05/19	Analista
19	Análise dos casos de uso para os grupos	3 días	lun 06/05/19	mié 08/05/19	Analista
20	**Deseño**	**4 días**	**mié 08/05/19**	**lun 13/05/19**	
21	Deseño de entidades	1 día	mié 08/05/19	mié 08/05/19	Analista
22	Deseño de interfaces e pantallas	3 días	jue 09/05/19	lun 13/05/19	Analista

Figura 6.1: Diagrama de Gantt, parte 1.



	Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
20	**Deseño**	**4 días**	**mié 08/05/19**	**lun 13/05/19**	
21	Deseño de entidades	1 día	mié 08/05/19	mié 08/05/19	Analista
22	Deseño de interfaces e pantallas	3 días	jue 09/05/19	lun 13/05/19	Analista
23	**Implementación**	**38 días**	**vie 10/05/19**	**mar 02/07/19**	
24	Integración de Blockly	11 días	vie 10/05/19	vie 24/05/19	Programador
25	Xestión de programas	14 días	lun 27/05/19	jue 13/06/19	Programador
26	Xestión de grupos	13 días	vie 14/06/19	mar 02/07/19	Programador
27	Probas da capa modelo	21 días	mar 04/06/19	mar 02/07/19	Programador
28	Reunión fin 2ª iteración	0 días	vie 05/07/19	vie 05/07/19	
29	**Iteración 3**	**46 días**	**mar 02/07/19**	**mié 04/09/19**	
30	**Análise**	**6 días**	**mar 02/07/19**	**mar 09/07/19**	
31	Análise dos requisitos para o carácter social da aplicación	4 días	mar 02/07/19	vie 05/07/19	Analista
32	Medidas de seguridade	2 días	lun 08/07/19	mar 09/07/19	Analista
33	**Deseño**	**7 días**	**lun 08/07/19**	**mar 16/07/19**	
34	Deseño de entidades	3 días	lun 08/07/19	mié 10/07/19	Analista
35	Deseño de fachadas	2 días	jue 11/07/19	vie 12/07/19	Analista
36	Deseño da interface web	2 días	lun 15/07/19	mar 16/07/19	Analista
37	**Implementación**	**35 días**	**mié 17/07/19**	**mar 03/09/19**	
38	Chat, comentarios e valoracións	12 días	mié 17/07/19	jue 01/08/19	Programador
39	Exercicios	11 días	vie 02/08/19	vie 16/08/19	Programador
40	Optimizacións e seguridade	12 días	lun 19/08/19	mar 03/09/19	Programador
41	Probas da capa modelo	11 días	mar 20/08/19	mar 03/09/19	Programador
42	Reunión final	0 días	mié 04/09/19	mié 04/09/19	

Figura 6.2: Diagrama de Gantt, parte 2.

- Na **terceira** iteración, implementar as demais operacións do sistema. As correspondentes ao carácter social da aplicación como son poder valorar e comentar programas, responder a comentarios, explorar traballos e usuarios, etc. As relativas á xestión de exercicios, que un *profesor* poderá empregar para encargar traballos aos seus *alumnos*, e o chat a través do cal se poden comunicar todos os membros do grupo. Ao igual que nas iteracións anteriores, realizar as correspondentes probas para as funcionalidades implementadas. A maiores, nesta iteración, realizar tarefas de optimización para mellorar as funcionalidades que o necesiten e implantar as medidas de seguridade pertinentes para a aplicación.

En canto ás funcionalidades estimadas para cada iteración, a **planificación final** cumpriunas. En canto ao marco temporal, xurdiron algunhas desviacións por non poder implementar a aplicación dun xeito continuado, xa fora pola realización de exames ou por motivos persoais no período estival. No diagrama de Gantt, dividido nas figuras 6.1 e 6.2, apréciase a duración temporal das tarefas. Como se comentou, algunha delas exténdese máis do debido por algunha interrupción, como é o caso das fases de implementación das iteracións 2 e 3. O esforzo real de horas de traballo empregadas no desenvolvemento do proxecto detállase na táboa 6.1.

Fase	**Recurso**	**Esforzo**
Iteración 0	Analista	55 horas
Iteración 1	Analista	32 horas
	Programador	58 horas
Iteración 2	Analista	28 horas
	Programador	117 horas
Iteración 3	Analista	45 horas
	Programador	115 horas
Esforzo total		**450 horas**

Táboa 6.1: Horas de Esforzo para o Desenvolvemento do Proxecto.

Respecto da **análise de custos**, elabórase unha aproximación ao coste asociado que tería este proxecto nun ámbito profesional. Para os cálculos reflexados na táboa 6.2 tivéronse en conta os seguintes perfís: o programador que desenvolve o proxecto que cobra 10€ a hora e o analista, quen cobra 15€ a hora.

Fase	**Custo**
Iteración 0	825€
Iteración 1	1.060€
Iteración 2	1.590€
Iteración 3	1.825€
Total desenvolvemento	5.300€
Luz	150€
Ordenadores	100€
Conexión a internet	200€
Alquiler	600€
TOTAL PROXECTO	**6.350€**

Táboa 6.2: Análise de Custos do Desenvolvemento do Proxecto.

En cada iteración sumáronse os soldos dos dous recursos e para obter unha cifra real do custo total do proxecto, ademais do desenvolvemento, tivéronse en conta os demais parámetros que se detallan na táboa. En canto ao precio dos ordenadores, dividiuse o seu prezo aproximado entre o tempo de desenvolvemento, xa que o seu uso pódese estender máis alá deste. Con todo, o custo total do proxecto sería de **6.350€**.

Capítulo 7

Requisitos do Sistema

7.1 Introdución

Os requisitos deste proxecto foron identificados pensando nas necesidades que un usuario podería requirir á hora de satisfacer a seu desexo de aprender a programar do xeito máis doado e intuitivo posible amais da posibilidade de axudarse doutras persoas que usaran a aplicación para mellorar a súa formación, interactuando con elas e aprendendo do seu traballo.

En primeiro lugar, un usuario que acceda á aplicación pode facerse unha idea do que esta lle pode ofrecer e ten a posibilidade de probar a biblioteca *Blockly*[1] modificando e executando un programa proposto como exemplo. A continuación, para poder ofrecer unha experiencia personalizada a cada usuario, é preciso ofrecer un sistema de autenticación que lle permitirá realizar e gardar o seu propio traballo, así como identificarse para poder intercambiar coñecementos coas demais persoas. Unha vez o usuario se rexistra proporcionando os seus datos persoais, xa ten acceso á maioría das funcionalidades da aplicación. Xa identificado, o usuario poderá consultar a información persoal do seu perfil e modificar calquera dos seus datos (a excepción do nome de usuario que o identifica).

En canto á xestión de programas, pode crear tantos programas como desexe asignándolles un nome e unha descrición. En cada programa, pode crear unha secuencia de pezas que proporciona *Blockly*, que se mostrará "traducida" en tempo real en código Python e JavaScript e que poderá executar para ver a súa saída. En calquera momento pode gardar o traballo realizado, desfacer un cambio ou limpar tanto espazo de traballo como a consola de saída.

Respecto dos grupos, calquera usuario identificado pode ver a lista de grupos á que pertence. No caso de contar co rol *Profesor*, ten a posibilidade de crear un grupo novo asignándolle un nome, do cal pasará a formar parte automaticamente e do cal será o seu único administrador. Poderá modificar o nome do grupo cando o desexe e ten a capacidade de engadir ou suprimir membros, consultando antes a información relativa a eles para confirmar a súa identidade. O administrador do grupo tamén pode eliminar o grupo e, por tanto, toda a in-

formación relativa a el. No caso dun usuario co rol *Alumno*, deberá esperar a que un *Profesor* o engada a algún grupo e apareceralle na súa lista de grupos. Non terá a posilibilidade de xestionar os demais membros do grupo pero si poderá abandonalo cando desexe. Cada grupo conta cun chat no que todos os membros do grupo poderán escribir e comunicarse entre eles. En cada mensaxe do chat mostrarase o nome do usuario que o realizou e través del poderase acceder á información do seu perfil (a información que se mostra nese apartado coméntase máis adiante).

Dentro dun grupo, o seu administrador poderá crear exercicios para que os demais membros os resolvan. Cada exercicio contará cun enunciado, cun grupo de pezas para resolvelo e cunha data e hora de expiración. Todos os membros do grupo terán acceso á lista de exercicios e en cada un deles poderán crear programas que fagan referencia a el. O administrador do grupo pode modificar o enunciado e data de expiración dos exercicios e mesmo elimininalos. Borrar o grupo tamén supón eliminar os exercicios pero, aínda así, en ningún caso borrar un exercicio supón eliminar os programas que fan referencia a el e cada membro poderá acceder á súa solución no apartado no que se mostran os seus programas.

En canto ao compoñente social da aplicación, un usuario pode xestionar os permisos dos seus programas. Estes poderán ser públicos, privados ou compartidos (o propietario decide quen o pode ver). A aplicación ofrece a posibilidade de consultar os programas compartidos por outros usuarios cun mesmo e explorar perfís doutros usuarios, polo seu nome, ou directamente programas que sexan públicos, por título e descrición. Ao explorar usuarios, pódese acceder aos datos do seu perfil, ver os seus programas públicos e os que teña compartidos coa persoa que accedeu, así como os grupos que teñan en común. Todos os programas ofrecen a posibilidade de realizar comentarios e valoracións por parte dos usuarios que teñan acceso a eles. A valoración consiste nunha nota de 0 a 10 e para cada programa mostrarase a media de todas as que recibiu.

7.2 Actores

Os actores que se distinguen dentro do sistema desenvolto son os seguintes:

- **Usuario non autenticado**: fai referencia aos usuarios que acceden á aplicación pero non están identificados mediante credenciais.
- **Usuario autenticado**: este actor representa a todos os usuarios que se identifican na aplicación proporcinando as súas credenciais, independentemente do seu rol, e del van herdar funcionalidades os actores *Profesor*, *Alumno* e *Administrador* que se comentan a continuación.
- **Alumno**: tamén ten acceso ás mesmas funcionalidades que o *Usuario autenticado* e o

seu rol indica a posibilidade de pertenza a grupos creados por un actor *Profesor* pero con menor responsabilidades ca este.

- **Profesor**: este actor herda as operacións permitidas ao actor *Usuario autenticado* e maiores conta con permisos especiais para a xestión de grupos.
- **Administrador**: este actor fai referencia ás operacións de xestión que pode realizar un *Profesor* nun grupo cando é o seu creador.

No diagrama da figura 7.1 móstrase a xerarquía entre os actores citados:

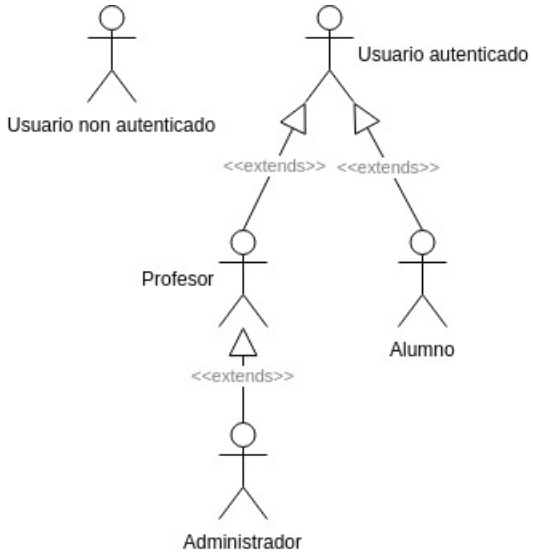


Figura 7.1: Xerarquía de actores.

7.3 Casos de Uso

A continuación enuméranse os Casos de Uso da aplicación:

CU-01	**Modificar o programa de exemplo**
Precondicións	O usuario non está autenticado.
Fluxo normal	O usuario modifica o programa de exemplo na páxina principal e proba as funcións relativas a el como desfacer cambios (CU-20), refacelos(CU-21), limpar o *workspace*(CU-22), executalo (CU-24) ou limpar a consola de saída (CU-23).
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-02	**Rexistro de usuario**
Precondicións	O usuario non está autenticado.
Fluxo normal	O usuario cobre todos os campos, pulsa o botón de *Rexistrarse* e móstrase a páxina principal.
Post-condicións	Créase unha conta nova e o usuario inicia sesión automaticamente.
Excepcións	Algúns dos campos baleiros, o nome de usuario xa exista, o contrasinal non cumpre coa lonxitude mínima e o *e-mail* non ten o formato correcto.

CU-03	**Inicio de sesión**
Precondicións	O usuario non está autenticado
Fluxo normal	O usuario introduce nome de usuario e contrasinal e pulsa o botón de *Login*.
Post-condicións	O usuario ten a sesión iniciada.
Excepcións	Algúns dos campos baleiros e o nome de usuario e contrasinal coinciden.

CU-04	**Peche de sesión**
Precondicións	O usuario está autenticado.
Fluxo normal	O usuario pulsa o botón de *Logout* e móstrase a pantalla de benvida.
Post-condicións	A sesión está finalizada e o usuario xa non está antenticado.
Excepcións	Ningunha.

CU-05	**Consultar datos do perfil**
Precondicións	O usuario está autenticado.
Fluxo normal	O usuario pulsa o botón *O meu perfil* no menú de usuario e na pantalla móstranse os datos do usuario coa sesión iniciada.
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-06	**Modificar contrasinal**
Precondicións	O usuario está autenticado e accedeu ao seu perfil (CU-05).
Fluxo normal	O usuario pulsa o botón *Cambiar contrasinal* e introduce o seu contrasinal vello e o novo dúas veces para confirmar que non se equivoca.
Post-condicións	O contrasinal da conta coa sesión iniciada modificouse.
Excepcións	Contrasinal vello incorrecto, os dous campos de novo contrasinal non coinciden, o contrasinal vello e o novo coinciden e contrasinal novo non ten o mínimo de caracteres esixidos.

CU-07	**Modificar nome**
Precondicións	O usuario está autenticado e accedeu ao seu perfil (CU-05).
Fluxo normal	O usuario pulsa o botón *Actualizar* correspondente ao *nome* e introduce o novo valor.
Post-condicións	O nome da conta coa sesión iniciada modificouse.
Excepcións	O campo está baleiro.

CU-08	**Modificar apelido**
Precondicións	O usuario está autenticado e accedeu ao seu perfil (CU-05).
Fluxo normal	O usuario pulsa o botón *Actualizar* correspondente ao *apelido* e introduce o novo valor.
Post-condicións	O apelido da conta coa sesión iniciada modificouse.
Excepcións	O campo está baleiro.

CU-09	**Modificar correo electrónico**
Precondicións	O usuario está autenticado e accedeu ao seu perfil (CU-05).
Fluxo normal	O usuario pulsa o botón *Actualizar* correspondente ao *e-mail* e introduce o novo valor.
Post-condicións	O correo electrónico da conta coa sesión iniciada modificouse.
Excepcións	O campo está baleiro e o *e-mail* non ten un formato válido.

CU-10	**Consultar programas creados**
Precondicións	O usuario está autenticado.
Fluxo normal	O usuario pulsa o botón *Os meus programas* no menú de usuario e lístanse os programas que creou o usuario coa sesión iniciada. Por cada un móstrase o título, a descrición, a data de creación, a data da última modificación e a súa visibilidade (público, privado ou compartido).
Post-condicións	Ningunha.
Excepcións	O usuario non ten ningún programa.

CU-11	**Crear programa**
Precondicións	O usuario está autenticado e accedeu á súa lista de programas (CU-10).
Fluxo normal	O usuario pulsa o botón + para crear un programa novo. Introduce o nome do programa e a descrición (opcional) e pulsa o botón de *Gardar*. Automaticamente ábrese o editor para ese programa.
Post-condicións	O programa creado é público por defecto.
Excepcións	O campo no nome do programa está baleiro.

CU-12	**Editar programa**
Precondicións	O usuario está autenticado e accedeu á súa lista de programas (CU-10).
Fluxo normal	O usuario pulsa o botón de editar para o programa que desexe. Introducirá o novo nome e/ou descrición e pulsará o botón de *Gardar*.
Post-condicións	O programa actualízase cos novos datos na BBDD.
Excepcións	O nome do programa está baleiro.

CU-13	**Borrar programa**
Precondicións	O usuario está autenticado e accedeu á súa lista de programas (CU-10).
Fluxo normal	O usuario pulsa o botón de borrar para o programa que desexe. Na modal de confirmación pulsa o botón de *Borrar*.
Post-condicións	O programa queda eliminado así como todos os datos asociados a el (comentarios e valoracións).
Excepcións	Ningunha.

CU-14	**Modificar os permisos dun programa**
Precondicións	O usuario está autenticado e accedeu á súa lista de programas (CU-10).
Fluxo normal	O usuario pulsa o botón de cambiar permisos para o programa que desexe. Na nova pantalla que aparece, o usuario establece o programa como público, privado ou compartido.
Post-condicións	No caso de esté compartido, mentres non seleccione usuarios con quen compartilo, equivale a que sexa privado.
Excepcións	Ningunha.

CU-15	**Compartir programa cun usuario**
Precondicións	O usuario está autenticado, accedeu á pantalla para modificar os permisos dun programa concreto (CU-14) e estableceuno como *compartido*.
Fluxo normal	No campo de búsqueda procura un usuario (CU-49). Para un dos usuarios que aparecen na búsqueda, pulsa no que se desexa engadir e aparecen os seus datos persoais para confirmar a súa identidade. Por último, pulsa o botón *Compartir*.
Post-condicións	O usuario engadido aparece na lista de usuarios cos que se comparte o programa.
Excepcións	Ningunha.

CU-16	**Deixar de compartir programa cun usuario**
Precondicións	O usuario está autenticado, accedeu á pantalla para modificar os permisos dun programa concreto (CU-14) e este está *compartido* polo menos cun usuario.
Fluxo normal	Da lista de usuarios cos que está compartido o programa, pulsa no botón de eliminar de un deles. Na modal de confirmación pulsa o bóton *Borrar*.
Post-condicións	O usuario eliminado desaparece da lista de usuarios cos que se comparte o programa.
Excepcións	Ningunha.

CU-17	**Acceder a un programa**
Precondicións	O usuario está autenticado e accedeu á súa lista de programas (CU-10).
Fluxo normal	O usuario pulsa nun dos programas que creou e ábrese unha pantalla nova na que se mostra o editor, as valoracións e os comentarios para dito programa.
Post-condicións	Ningunha.
Excepcións	O programa non existe (no caso de acceder directamente pola URL).

CU-18	**Modificar as *pezas* dun programa**
Precondicións	O usuario está autenticado e accedeu a un dos seus programas (CU-17).
Fluxo normal	Modificar a secuencia de pezas que conforman o programa. Automaticamente, actualízase a "traducción" en código real (Python e JavaScript). Móstrase un aviso de que os cambios non están gardados.
Post-condicións	Os cambios non están gardados e perderíanse no caso de recargar a páxina.
Excepcións	Ningunha.

CU-19	**Actualizar as *pezas* dun programa**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) e realizou algunha modificación nas pezas que forman o programa (CU-18).
Fluxo normal	O usuario pulsa o botón de Gardar.
Post-condicións	Os cambios actualízanse na BBDD.
Excepcións	Ningunha.

CU-20	**Desfacer un cambio**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) e realizou algunha modificación nas pezas que forman o programa (CU-18).
Fluxo normal	O usuario pulsa o botón de *Undo* e a secuencia de pezas do programa volve ao seu estado anterior.
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-21	**Refacer un cambio**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17), realizou algunha modificación nas pezas que forman o programa (CU-18) e desfixo algunha vez os cambios (CU-20).
Fluxo normal	O usuario pulsa o botón de *Redo* e a secuencia de pezas do programa volve ao seu estado anterior antes de facer desfacer os cambios.
Post-condicións	Ningunha.
Excepcións	O usuario non desfixo ningún cambio e non se modifica nada.

CU-22	**Limpar o *workspace***
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17).
Fluxo normal	O usuario pulsa o botón de limpar o *workspace* e este queda baleiro.
Post-condicións	Ningunha.
Excepcións	Se o *workspace* xa estaba baleiro e non se modifica nada.

CU-23	**Limpar a consola de saída**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) e executouno (CU-24).
Fluxo normal	O usuario pulsa o botón de limpar a consola de saída na que se mostra o resultado de execución do programa e esta queda baleira.
Post-condicións	Ningunha.
Excepcións	Se a consola de saída xa estaba baleira e non se modifica nada.

CU-24	**Executar un programa**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) e desenvolveu algún programa no *workspace*.
Fluxo normal	O usuario pulsa o botón de executar e na consola de saída móstrase o resultado de execución do programa.
Post-condicións	Ningunha.
Excepcións	Se non hai código que executar, na consola non se mostra nada.

CU-25	**Realizar un comentario**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) ou a calquera outro ao que teña acceso.
Fluxo normal	O usuario escribe un comentario e pulsa o botón de enviar. Automaticamente móstrase o novo comentario debaixo do programa.
Post-condicións	Ningunha.
Excepcións	O campo de texto do comentario está baleiro.

CU-26	**Responder a un comentario**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) ou a calquera outro ao que teña acceso.
Fluxo normal	O usuario pulsa o botón de responder nun comentario en concreto, escribe o texto que desexa e pulsa no botón *Enviar*. Automaticamente móstrase o novo comentario de forma aniñada respecto do comentario ao que se respondeu.
Post-condicións	Ningunha.
Excepcións	O campo de texto do comentario está baleiro.

CU-27	**Borrar un comentario**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) ou a calquera outro ao que teña acceso.
Fluxo normal	O usuario pulsa o botón de eliminar nun dos comentarios (ou respostas) que el mesmo realizou. Na modal de confirmación pulsa o botón *Borrar* e o comentario desaparece automaticamente.
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-28	**Valorar un programa**
Precondicións	O usuario está autenticado, accedeu a un dos seus programas (CU-17) ou a calquera outro ao que teña acceso.
Fluxo normal	O usuario desplaza a barra de puntuación para asignar un nota de 0 a 10 ao programa e pulsa o botón de *Enviar*. Despois disto, actualízase a nota media do programa e o número de usuarios que votaron. Cada usuario só pode valorar un programa unha vez.
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-29	**Consultar grupos**
Precondicións	O usuario está autenticado.
Fluxo normal	O usuario pulsa o botón *Os meus grupos* no menú de usuario e lístanse os grupos aos que pertence o usuario coa sesión iniciada. Por cada un móstrase o nome do grupo a súa data de creación.
Post-condicións	Ningunha.
Excepcións	O usuario non pertence a ningún grupo.

CU-30	**Crear un grupo**
Precondicións	O usuario está autenticado e accedeu á súa lista de grupos (CU-29).
Fluxo normal	O usuario pulsa o botón + para crear un grupo novo. Introduce o nome do grupo (obrigatorio) e pulsa o botón de *Gardar*. Automaticamente ábrese unha nova pantalla na que se mostra a lista de exercicios (baleira), o chat e a lista de membros.
Post-condicións	Automaticamente o usuario que creou o grupo é membro do mesmo e el será o único administrador.
Excepcións	O campo no nome do grupo está baleiro.

CU-31	**Acceder a un grupo**
Precondicións	O usuario está autenticado e accedeu á súa lista de grupos (CU-29).
Fluxo normal	O usuario pulsa nun dos grupos aos que pertence e ábrese unha nova pantalla na que se mostra a lista de exercicios, o chat e a lista de membros.
Post-condicións	Ningunha.
Excepcións	O grupo non existe (no caso de acceder directamente pola URL).

CU-32	**Engadir un membro a un grupo**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa no botón de *Xestionar* enriba da lista de membros. Na nova pantalla que se abre, busca a un usuario (CU-49), pulsa no botón da frecha do que se desexa engadir, móstranse os seus datos persoais para confirmar identidade e púlsase o botón *Engadir*.
Post-condicións	O usuario aparece na lista de membros do grupo.
Excepcións	Ningunha.

CU-33	**Eliminar un membro do grupo**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa no botón de *Xestionar* enriba da lista de membros. Na nova pantalla que se abre, dos usuarios que aparecen na lista de membros, pulsa no botón de eliminar de un deles. Na modal de confirmación pulsa o botón *Eliminar*.
Post-condicións	O usuario desaparece da lista de membros do grupo.
Excepcións	Ningunha.

CU-34	**Filtrar membros dun grupo por nome**
Precondicións	O usuario está autenticado e accedeu a un grupo (CU-31).
Fluxo normal	O usuario escribe nun cadro de búsqueda e na lista de membros móstranse soamente os que cuxo *username*, nome, apelido ou *e-mail* coincide co da búsqueda.
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-35	**Editar grupo**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa o botón de editar o nome do grupo, escribe o novo valor e pulsa *Gardar*.
Post-condicións	Ningunha.
Excepcións	O campo de texto está baleiro.

CU-36	**Borrar grupo**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa o botón de eliminar o grupo e na modal de confirmación pulsa o botón *Eliminar*.
Post-condicións	Elimínase o grupo e toda a información relativa a el como son os exercicios e as mensaxes do chat. Os programas que están vinculados aos exercicios non se borran.
Excepcións	Ningunha.

CU-37	**Enviar mensaxe no chat**
Precondicións	O usuario está autenticado e accedeu a un grupo (CU-31).
Fluxo normal	O usuario escribe unha mensaxe no cadro de texto do chat e pulsa o botón de enviar. Automaticamente móstrase a mensaxe no chat xunto coa data e hora de envío. Para os demais membros do grupo tamén se mostrará o nome de usuario do remitente que será un enlace ao seu perfil (CU-38).
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-38	**Acceder a información de usuario dende o chat**
Precondicións	O usuario está autenticado e accedeu a un grupo (CU-31).
Fluxo normal	O usuario pulsa no nome de usuario que aparece como remitente dunha mensaxe no chat e accede á información dese usuario: datos persoais, programas compartidos por ese usuario co que accede ao perfil, programas públicos e grupos en común.
Post-condicións	Ningunha.
Excepcións	O usuario non existe (no caso de acceder directamente pola URL).

CU-39	**Inhabilitar o chat do grupo**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31) e é o administrador.
Fluxo normal	O usuario pulsa o botón de *Inhabilitar* e aparece no seu lugar o botón *Habilitar*.
Post-condicións	Ningún membro do grupo pode enviar mensaxes no chat.
Excepcións	Ningunha.

CU-40	**Habilitar o chat do grupo**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31), é o administrador e inhabilitou o chat do grupo (CU-39).
Fluxo normal	O usuario pulsa o botón de *Habilitar* e aparece no seu lugar o botón *Inhabilitar*.
Post-condicións	Os membro xa poden enviar mensaxes no chat.
Excepcións	Ningunha.

CU-41	**Abandonar un grupo**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31) e **non** é o administrador.
Fluxo normal	O usuario pulsa o botón de *Deixar grupo* e é redirixido á pantalla na que se mostran os seus grupos.
Post-condicións	Entre os seus grupos xa non figura o que acaba de abandonar.
Excepcións	Ningunha.

CU-42	**Crear un exercicio**
Precondicións	O usuario está autenticado, accedeu a un grupo (CU-31) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa o botón de *Novo exercicio* e introduce un enunciado, os grupos de pezas para resolvelo e unha data límite. A continuación, ábrese unha nova pantalla mostrando a información do exercicio e a lista de programas propostos polos membros do grupo como solucións do exercicio (baleira).
Post-condicións	Ningunha.
Excepcións	O texto do enunciado está baleiro e a data é inválida.

CU-43	**Acceder a un exercicio**
Precondicións	O usuario está autenticado e accedeu a un grupo (CU-31).
Fluxo normal	O usuario pulsa nun dos exercicios que aparecen no grupo e accede a unha nova páxina na que se mostrar a información do exercicio e a lista de programas propostos polos membros do grupo como solucións do exercicio.
Post-condicións	Ningunha.
Excepcións	O exercicio non existe (no caso de acceder directamente pola URL).

CU-44	**Editar exercicio**
Precondicións	O usuario está autenticado, accedeu a un exercicio de un grupo (CU-43) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa no botón de editar o exercicio e edita o enunciado e/ou a data límite.
Post-condicións	Ningunha.
Excepcións	O texto do enunciado está baleiro e a data é inválida.

CU-45	**Borrar exercicio**
Precondicións	O usuario está autenticado, accedeu a un exercicio de un grupo (CU-43) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa no botón de borrar o exercicio e na modal de confirmación pulsa o botón *Eliminar*.
Post-condicións	Os exercicios propostos como solución non se borran e os seus propietarios poden seguir accedendo a eles dende os seus programas (CU-10).
Excepcións	Ningunha.

CU-46	**Crear solución a un exercicio**
Precondicións	O usuario está autenticado, accedeu a un exercicio de un grupo (CU-43) e el é o administrador do mesmo.
Fluxo normal	O usuario pulsa o botón *Nova solución* e crea un programa novo abríndose automaticamente o editor para ese programa.
Post-condicións	O programa creado aparece na lista de solucións do respectivo exercicio.
Excepcións	Ningunha.

CU-47	**Consultar programas compartidos co usuario actual**
Precondicións	O usuario está autenticado.
Fluxo normal	O usuario pulsa o botón *Explorar* no menú de usuario e móstranse, ademais das barras de búsqueda de programas e usuarios, os programas doutros usuarios compartidos con el.
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-48	**Buscar un programa**
Precondicións	O usuario está autenticado e accedeu á paxina de *Explorar* (CU-47).
Fluxo normal	Na barra de búsqueda correspondente á búsqueda de programas, escribe as palabras clave e debaixo aparecen os programas públicos doutros usuarios cuxo nome ou descrición coinciden coa búsqueda. O usuario pulsa no programa que lle interesa e accede a el.
Post-condicións	Ningunha.
Excepcións	Ningunha.

CU-49	**Buscar un usuario**
Precondicións	O usuario está autenticado e accedeu á paxina de *Explorar* (CU-47).
Fluxo normal	Na barra de búsqueda correspondente á búsqueda de usuarios, escribe o texto que desexa e debaixo aparecen os usuarios cuxo *username*, nome, apelido ou *e-mail* coincide ca búsqueda. Pulsando nalgún deles, aparece un resumo dos seus datos persoais e pulsando despois no botón *Ir a perfil* accédese a outra páxina na que se mostran os seus datos persoais, programas compartidos por ese usuario co que accede ao perfil, programas públicos e grupos en común.
Post-condicións	Ningunha.
Excepcións	Ningunha.

7.4 Modelo de Casos de Uso

Para facilitar a súa visualización, o modelo de Casos de Uso dividiuse por funcionalidades e segundo a xerarquía de actores, definida na figura 7.1.

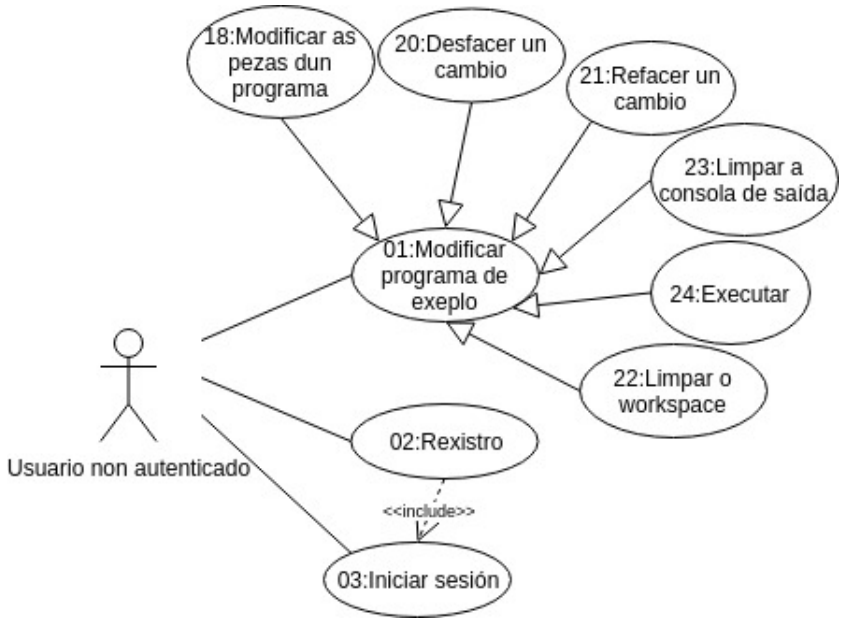


Figura 7.2: Casos de Uso para o usuario non autenticado.

En primeiro lugar móstrase o diagrama correspondente as operacións dispoñibles para un usuario non autenticado na figura 7.2. Nela vemos os tres Casos de Uso dos que dispón este actor e apréciase o feito de que rexistrarse na aplicación implica automaticamente o inicio de sesión.

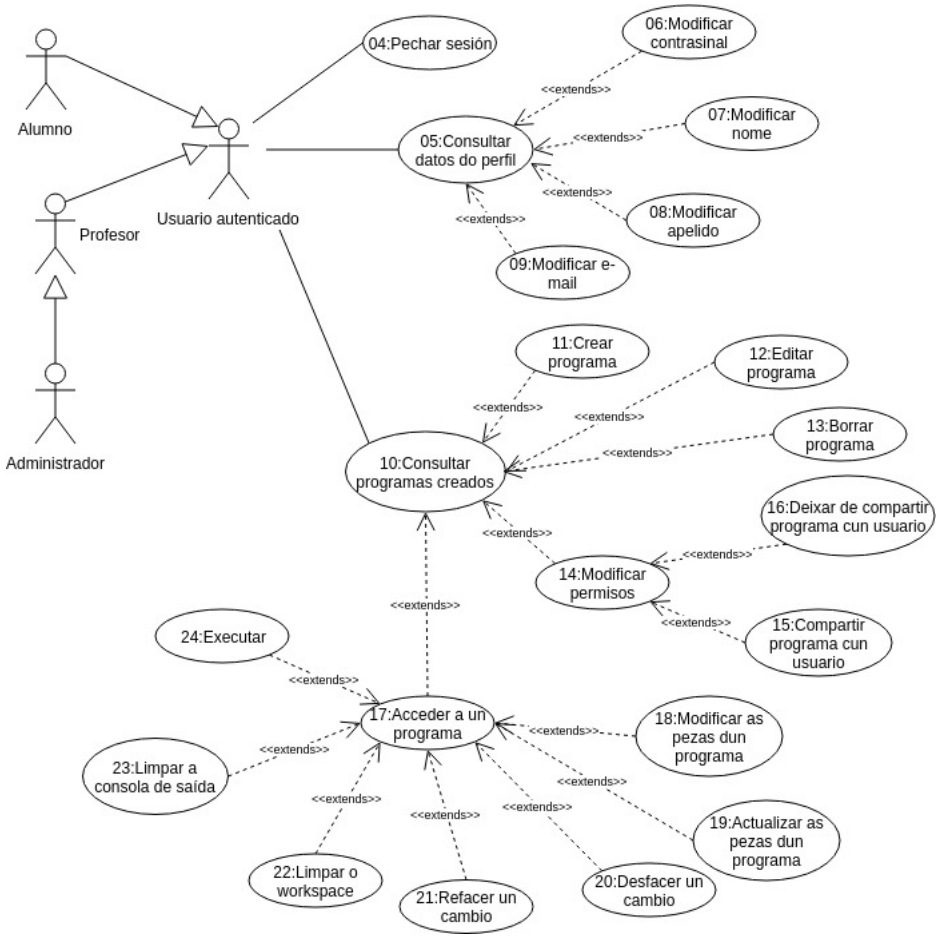


Figura 7.3: Casos de Uso para o usuario autenticado.

Na figura 7.3 defínense os Casos de Uso para calquera usuario autenticado. Para unha maior claridade, os relativos á xestión de grupos amósanse na figura 7.4 e os referidos ao compoñente social da aplicación na figura 7.5. Como se observa, calquera usuario autenticado ten acceso ao seu perfil e aos seus programas. A partir destes dous Casos de Uso, conta cun abanico de posibilidades segundo a operación que desexe levar a cabo, todas elas opcionais, polo que extenden o Caso de Uso base.

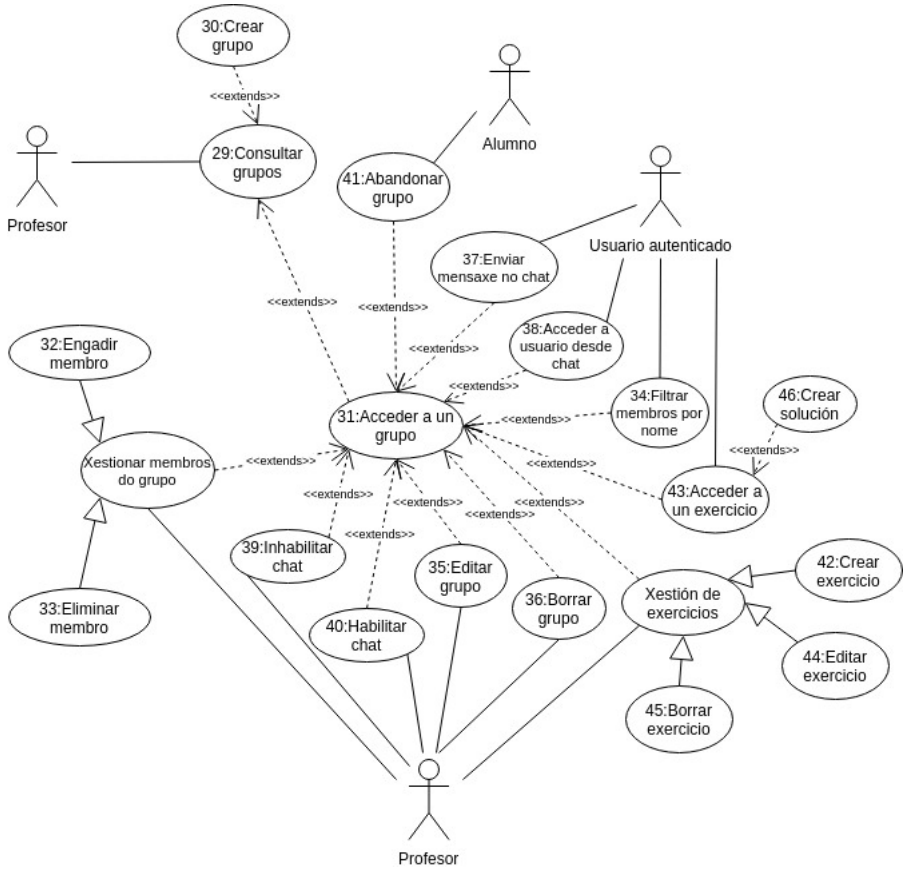


Figura 7.4: Casos de Uso para a funcionalidade de grupos.

A continuación, na figura 7.4 detállanse os Casos de Uso para as funcionalidades relativas aos grupos. Neste diagrama, trátase de diferenciar as responsabilidades coas que contan os diferentes actores. Calquera usuario identificado pode acceder a un grupo ao que pertence e acceder a un exercicio concreto, propoñer solucións a este ou enviar mensaxes no chat. As operacións de xestión do grupo e dos seus membros, así como a xestión de exercicios, só as pode realizar un *Profesor* que sexa o *Administrador* do grupo. Pola súa banda un *Alumno*, é dicir, calquera outro membro do grupo que non sexa o *Administrador*, pode abandonalo en calquera momento.

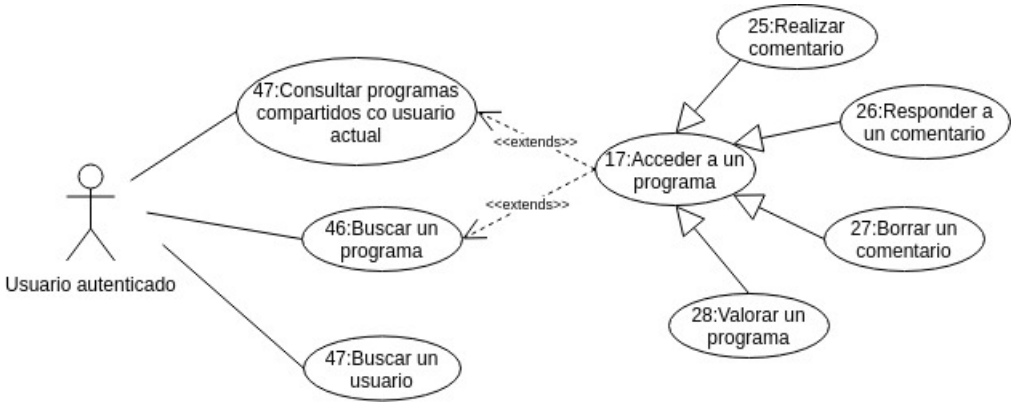


Figura 7.5: Casos de Uso para a funcionalidade social.

Por último, na figura 7.5 atópanse as funcionalidades das que dispón a aplicación para permitir a exploración doutros traballos e usuarios. Ademais da posibilidade de buscar usuarios, pode explorar os programas alleos aos que ten acceso. Nestes programas ten permitido realizar comentarios e valoracións para expresar a súa opinión acerca dese traballo.

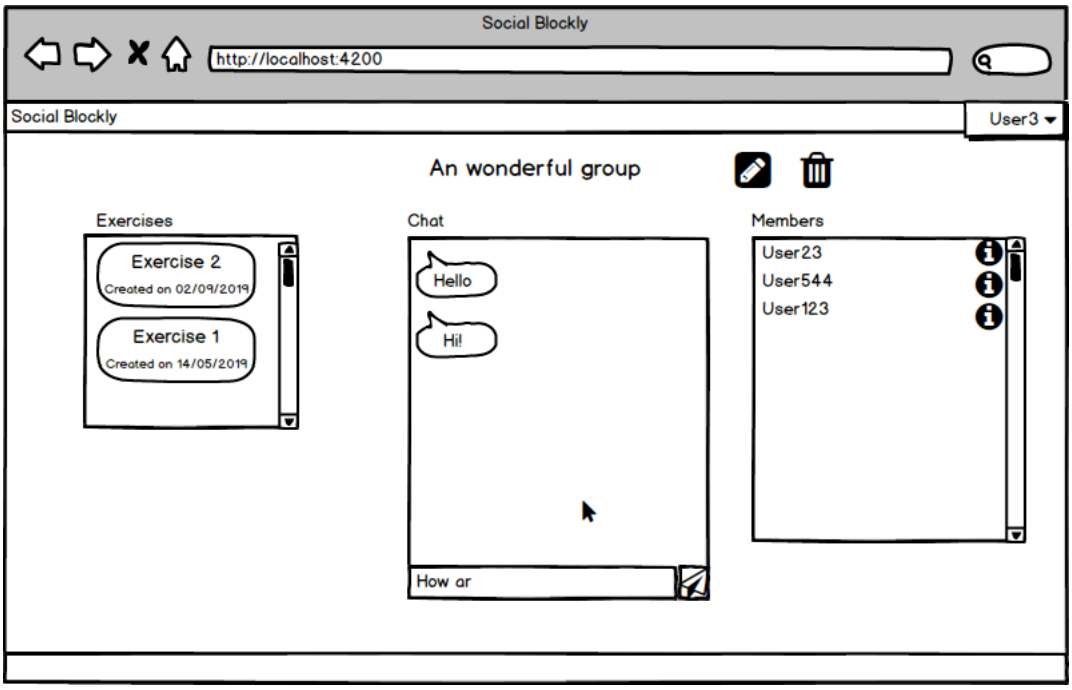


Figura 7.6: Mockup da interface da información dun grupo.

Para axudar a modelar aplicación, a veces é preciso utilizar unha ferramenta que permita dar forma á interface e centrarse unicamente neste aspecto, sen necesidade de programar. Neste caso fíxose uso de Balsamiq[14], coa que se recrearon algunhas das *vistas* máis com-

plexas para axudar a decidirse sobre o deseño. Na figura 7.6 vese un *mockup* da pantalla que aparece ao acceder a un grupo.

Capítulo 8

Deseño da Aplicación

8.1 Introdución e Obxectivos

Para o deseño da aplicación é importante ter en conta as características que fan, dun software, un proxecto facilmente extensible no futuro e que conte cun mantemento o máis doado e eficiente posible. As fundamentais son a independencia entre os diferentes módulos para un baixo acoplamento permitindo que modificar un anaco de código non implique cambios noutro e unha alta cohesión obrigando a que cada compoñente se adique unicamente ás súas responsabilidades. Deste xeito, cando se desexe modificar ou ampliar a funcionalidade dunha parte, poderase facer de forma cómoda e sen que surxan problemas adicionais noutras seccións do código. Esta filosofía de deseño tamén favorece a reutilización dalgunha capa noutro software que se queira desenvolver.

O obxectivo é conseguir que este software cumpra na súa totalidade con estas características aplicando boas prácticas de deseño en todos os módulos e empregando os patróns de deseño axeitados.

8.2 Resumo de Patróns Usados

Os patróns de deseño máis destacables que se empregaron neste proxecto son:

- Deseño baseado en **capas**: para conseguir que cada parte da aplicación se poida manter e extender facilmente de forma independente sen afectar ás demais, sepárase en varias capas que se comunican entre elas nunha orde determinada. Calquera modificación ou ampliación nunha das capas non debería afectar ás demais mentres non se modifiquen as interfaces, que seguirían funcionando sen cambios adicionais. Igualmente, pódense desenvolver outras implementacións a maiores de calquera capa que consuma os mesmos recursos que a orixinal ou reutilizar algunha parte noutra aplicación coa que comparta requisitos.

- Uso de **DAOs** (Data Access Objects): en relación ao punto anterior, a utilización de DAOs ofrece unha capa máis, dentro da capa modelo, que abstrae e encapsula o acceso á fonte de datos e coa que se comunica a capa da lóxica de negocio. Neste caso proporciona unha interface para a comunicación cunha Base de Datos MySQL[25].

- Uso de **DTOs** (Data Transfer Objects): nesta aplicación, a comunicación entre *frontend* e *backend* lévase a cabo a través da rede, cos problemas inherentes que pode orixinar (velocidade, latencia, etc) polo que é desexable realizar o menor número de peticións posibles. Para solucionar este problema, utilízanse DTOs que recollen toda a información precisa para unha determinada petición e a envían nun determinado formato.

- Uso de **Inxección de dependencias**: para non depender das implementacións concretas das capas inferiores, faise uso deste patrón aproveitando as facilidades do contedor de dependencias de Spring. Cando unha clase precisa unha instancia doutra dunha capa inferior, non a crea ela mesma senón que mediante unha anotación (*@Autowired*), indícaselle a Spring que debe proporcionarlla e é o framework o encargado de facelo.

8.3 Arquitectura Xeral

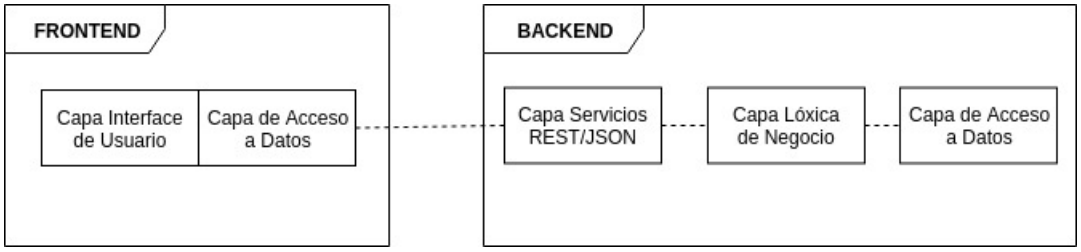


Figura 8.1: Subsistemas da Aplicación.

Neste proxecto distínguense dous subsistemas. Un deles é o servizo web (*backend*) que, se comunica coa Base de Datos para consultar e modificar información, contén a lóxica de negocio e ofrece unha API REST para ser consumida por un cliente. O outro subsistema é a aplicación web (*frontend*) que, accede á API REST do servizo web e se encarga de mostrarlle a información ao usuario. Na figura 8.1 móstrase a división da aplicación nos dous subsistemas comentados.

8.4 Subsistema Servizo Web (Backend)

8.4.1 Obxectivos

Este primeiro subsistema fai referencia ao servizo web desenvolto con Spring Boot[8] (*backend*). O principal obxectivo nesta parte do proxecto, en canto ao deseño, é a implementación dunha arquitectura baseada en capas que facilite, por medio da abstracción e a división por responsabilidades, o seu desenvolvemento e posterior mantemento. Preténdese que calquera arranxo nunha parte do código sexa o máis trivial posible evitando erros encadeados. Tamén se busca que as capas superiores non dependan de implementacións concretas de outras capas inferiores, seguindo o principio de inversión da dependencia, mediante a inxección de dependencias que proporciona Spring.

8.4.2 Arquitectura

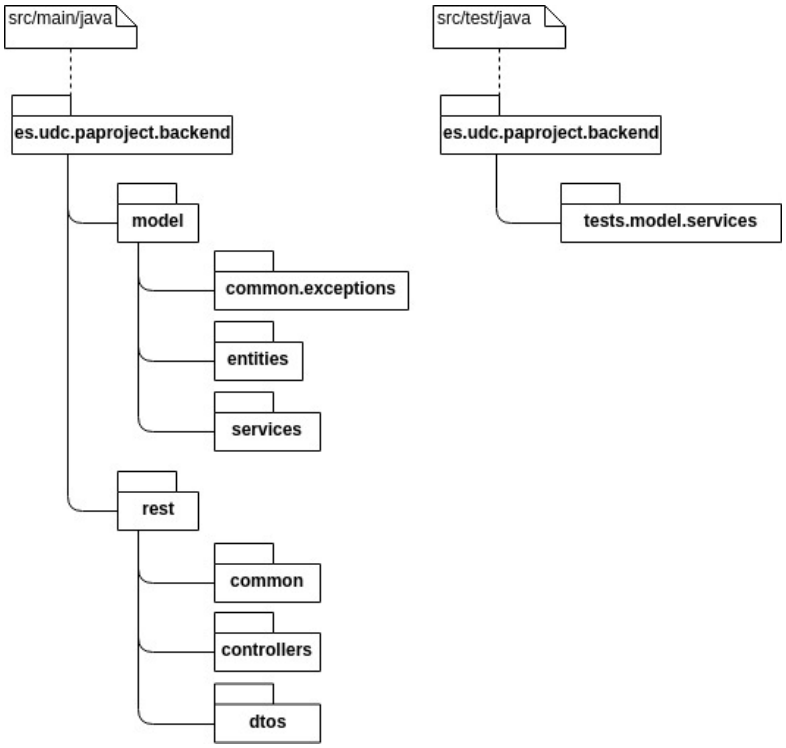


Figura 8.2: Diagrama de Paquetes do Servizo Web.

Na figura 8.2 amósase a estrutura de paquetes do servizo web. Nela apréciase a división, dentro da capa modelo, entre a capa de acceso a datos, que corresponde ao paquete *entities* e o da lóxica de negocio, o paquete *services*. A parte deso, pendurando do paquete *rest* atopamos

as clases pertencentes á capa de servizos que ofrece a API REST. A continuación, explícanse estas partes en máis profundidade. Para desenvolver o *backend* da aplicación, en primeiro lugar implementáronse as entidades que modelan toda a información que vai xestionar a aplicación, e os DAOs (Obxectos de Acceso a Datos) que se encargan da comunicación coa Base de Datos para persisitir esa información. O framework Spring Boot abstrae esta comunicación coa BBDD para recuperar ou modificar os datos que se desexen, e o propio framework ofrece mecanismos para as operacións básicas como creación, lectura, borrado ou actualización. Para operacións máis complexas desenvolvéronse implementacións adicionais con consultas específicas. A continuación, desenvolveuse a capa servizo na cal se implementa toda a lóxica de negocio, é dicir, todas as operacións que a aplicación vai poder realizar e que traballan directamente cos DAOs. Seguindo co deseño por capas para facilitar o desenvolvemento e mantemento do *backend*, a esta capa non se accede directamente dende o exterior, senón que se desenvolve outra que proporciona un grao máis de abstracción, que recibirá as peticións e se comunicará co servizo. O feito de realizar esta abstracción tamén permite realizar outras implementacións distintas que accedan a ela nun futuro, se fora necesario. Esta última capa que recibe as peticións é unha API REST formada por diferentes controladores, divididos segundo a funcionalidade que manexan. Estes controladores reciben as peticións realizadas por un cliente vía HTTP e acceden á capa servizo esperando unha reposta que devolven contida nun DTO (Obxecto de Transferencia de Datos) que recolle a información requerida no formato correcto. Esa información é devolta en formato JSON.

8.4.3 Modelo do Dominio

Diagrama de Entidades

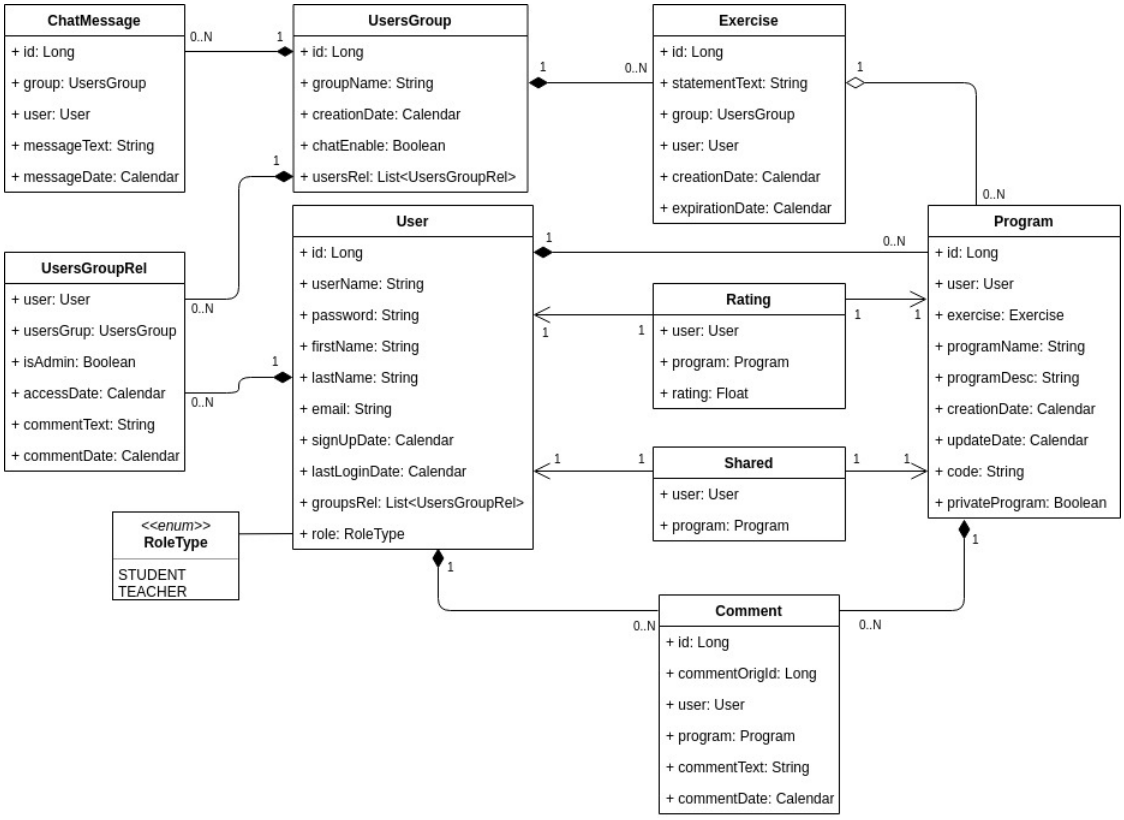


Figura 8.3: Diagrama de Entidades.

Na figura 8.3 móstrase o diagrama de entidades, con todas as clases modeladas na capa de acceso a datos do servidor web. Todas elas contan cun identificador autoxerado (*id*) a excepción de *Rating*, *Shared* e *UsersGroupRel*, que se distinguen polas entidades ás que fan referencia de tal xeito que non se poden repetir. Deseguido, detállase cada entidade por separado:

- **User**: almacena os datos persoais do usuario como son o *username*, o contrasinal, o nome, apelido, *e-mail*, a data na que se rexistra e a última vez que iniciou sesión. Para controlar o rol de cada usuario existe un *enumerado* cos valores *Alumno* e *Profesor*.
- **Program**: fai referencia aos programas que vai crear e almacenar cada usuario.
- **Comment**: modela os comentarios que calquera usuario, con acceso a un programa, pode engadirlle.

- **Rating**: representa as valoracións que os usuarios fan sobre os programas aos que poden acceder.
- **Shared**: modela a relación entre os programas e os usuarios cos que se comparte un programa. Un programa pode estar compartido con moitos usuarios e moitos programas puideron ser compartidos cun só usuario.
- **Exercise**: representa os exercicios que un *Profesor* propón dentro de un grupo determinado.
- **UsersGroup**: correspóndese cos grupos de usuarios que se poden crear na aplicación.
- **UsersGroupRel**: modela a relación entre usuarios e grupos. Foi necesaria para xestionar algunhas operacións de búsqueda tendo en conta que un usuario pode pertencer a varios grupos e un grupo pode contar con varios usuarios.
- **ChatMessage**: fai referencia ás mensaxes dentro do chat de cada grupo.

A continuación, explícanse as relacións entre as distintas entidades. No caso das relación de un a moitos (1:N), detállanse desde a entidade propietaria da relación.

- **Program**: ten unha relación N:1 con *User*, xa que un usuario pode posuír varios programas, e outra N:1 con *Exercise*. No caso da segunda, indícase que a existencia dun programa ten sentido por si mesma e, de feito, tal e como se indica nos requisitos, borrar un exercicio non provoca que se eliminen os programas asociados.
- **Comment**: posúe unha relación N:1 con *User*, porque un usuario pode realizar moitos comentarios, e outra relación N:1 con *Program*, xa que esta entidade tamén pode contar con moitos comentarios.
- **Rating**: conta con dúas relacións 1:1. Unha deleas con *User* e a outra con *Program*, para modelar as valoracións que realiza un usuario sobre un programa.
- **Shared**: ten dúas relacións 1:1, unha con *User* e outra con *Program*, que se corresponde co feito de que un programa se pode compartir con varios usuarios e varios programas poden ser compartido cun só usuario.
- **Exercise**: posúe cunha relación N:1 con *UsersGroup*, dado que nun grupo poden propoñerse distintos exercicios.
- **UsersGroupRel**: ten unha relación N:1 con *User* e *UsersGroup*, para modelar os membros dun grupo concreto que, ademais, poden ser membros doutros grupos distintos.
- **ChatMessage**: conta unha relación N:1 con *UsersGroup*, para modelar todas as mensaxes que se poden enviar polo chat dun grupo.

Modelo de Datos

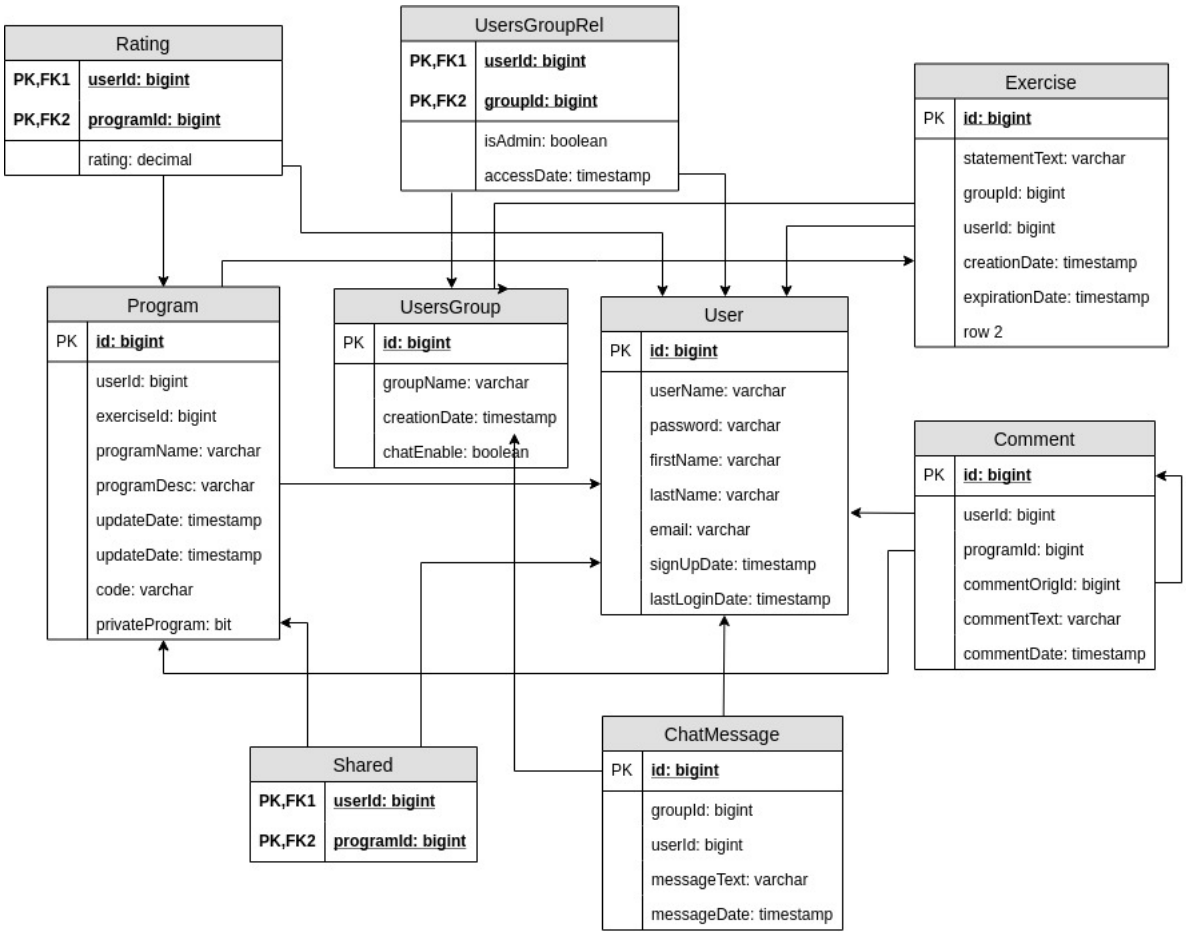


Figura 8.4: Modelo de Datos.

Na figura 8.4 amósase o modelo de datos da aplicación, isto é, as táboas que manexa o sistema na Base de Datos. En relación á sección anterior, todas as táboas contan cunha columna (*id*) para identificar cada fila, excepto *Rating*, *Shared* e *UsersGroupRel*. Nestes casos, á chave primaria é compartida e está formada por dúas chaves foráneas ás táboas correspondentes. No caso da táboa *Comment*, existe unha relación consigo mesma para os casos nos que un usuario responde a outro comentario e poder facer referencia a el.

Consideración Adicionais de Modelado de Datos

Mediante anotacións, defínese o modelado das relacións entre clases indicándolle a Spring como debe levar a cabo as relacións (e.g. columna da táboa á que referencia un atributo). Por

defecto, cando se recupera unha entidade de Base de Datos que conta cunha relación *moitos-a-un* (N:1), tamén se recupera a entidade á que *apunta* (política *EAGER*). Isto pode ocasionar que se recuperen varias entidades de forma encadeada, o cal supón un gasto de recursos inútil. Para solucionalo, engádese a anotación *@ManyToOne(optional=false, **fetch=FetchType.LAZY**)*, onde o parámetro *LAZY* indica que só se recupere esa entidade cando sexa preciso, retrasándoo o máximo posible.

8.4.4 Capa de Acceso a Datos

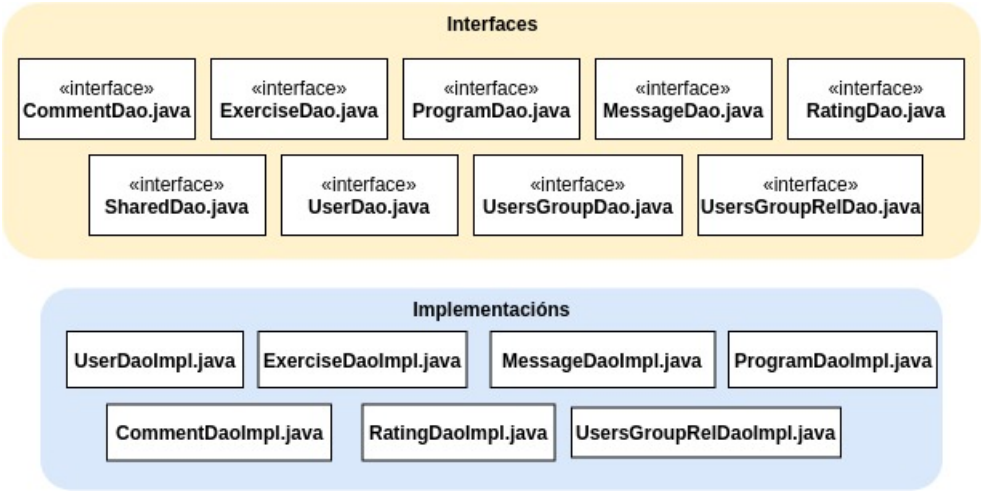


Figura 8.5: Diagrama Capa de Acceso a Datos.

Na figura 8.5 enuméranse os DAOs da capa de acceso a datos e as implementacións que foron precisas nalgúns casos para realizar consultas especiais que non proporcionaban por defecto as interfaces. Estas interfaces fan uso das facilidades que ofrece o framework de Spring extendendo a súa clase *PagingAndSortingRepository*[31] que proporciona as funcionalidades necesarias para facer as consultas e modificacións precisas contra o sistema de persistencia. Nos casos nos que se realizaron implementacións adicionais para determinadas consultas sobre a BBDD, utilizouse a interface *EntityManager* de Java e a anotación de Spring *PersistenceContext* para asocialo coa dependencia da fonte de datos empregada.

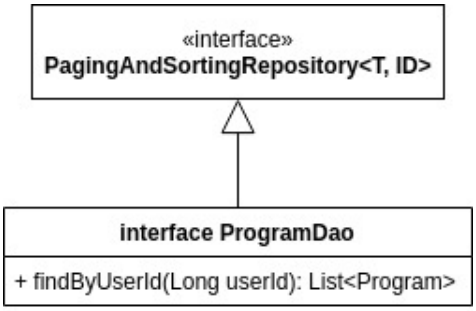


Figura 8.6: Arquitectura dun DAO.

Para mostrar a arquitectura dun DAO, posto que a estrutura de todos eles é similar, amósase un de exemplo na figura 8.6, neste caso *ProgramDao*. Como se comentou, extende a interface *PagingAndSortingRepository* a cal precisa como parámetros o tipo de dominio que manexará (*T*) e o tipo de identificador da entidade (*ID*).

A interface *PagingAndSortingRepository* extende á súa vez a interface *CrudRepository*. Esta outra interface conta coas operacións precisas para devolver o número de entidades dispoñibles, borrar unha entidade, borrar todas as entidades, borrar por identificador, devolver todas as instancias dun tipo, buscar unha instancia por *id*, gardar unha entidade e algunha máis similar a estas pero con diferentes parámetros. As operacións que engade *PagingAndSortingRepository* son relativas á paxinación e ordenamento de elementos.

A maiores, como se pode ver na figura 8.6 pódese definir usando a semántica correcta e mediante o nome dun atributo, neste caso *userId* en vez de *id*, unha nova operación sen necesidade de que o programador a implemente. Nesta aplicación, as demais operacións definidas deste xeito foron en *UserDao*:

- *boolean existsByUserName(String userName)*: devolve *verdadeiro* ou *falso* dependendo se existe un usuario co nome de usuario indicado.
- *Optional<User> findByUserName(String userName)*: se existe, devolve un usuario co nome de usuario indicado.

Aínda así, hai certas operacións que debido á súa complexidade é necesario implementalas por separado. É o caso das operacións contidas nos ficheiros agrupados na área azul da figura 8.5 (*Implementacións*) que se detallan a continuación. En *UserDaoImpl* definíronse dúas operacións para obter usuarios por palabras chave e para devolver a información dun usuario concreto en base ao seu *username*. No arquivo *ExerciseDaoImpl.java* incluíuse unha función para devolver os exercicios relacionados cun grupo. Na clase MessageDaoImpl, creáronse dúas funcións para devolver as mensaxes dun grupo, unha para devolvelas todas e outra que precisa especificarlle un intervalo temporal. No DAO *ProgramDaoImpl*, definíronse funcións

para devolver todos os programas dun usuario, devolver só os públicos, atopar os programas compartidos cun usuario, encontrar os usuarios cos que se compartiu un programa, devolver os programas compartidos por un usuario con outro en concreto, achar programas por palabras chave, deixar de compartir un programa con todos os usuarios que tiñan acceso a el e atopar os programas asociados a un exercicio. No arquivo *CommentDaoImpl.java*, creáronse operacións para devolver os comentarios dun programa e para devolver as respostas a un comentario. No caso de *RatingDaoImpl* só se engadiu unha función nova que devolve as valoracións dun programa en base ao seu identificador. Por último, para *UsersGroupRelDaoImpl* as operacións implementadas son para devolver todos os usuarios pertencentes a un grupo e todos os grupos aos que pertence un usuario.

8.4.5 Capa Servizos do Modelo

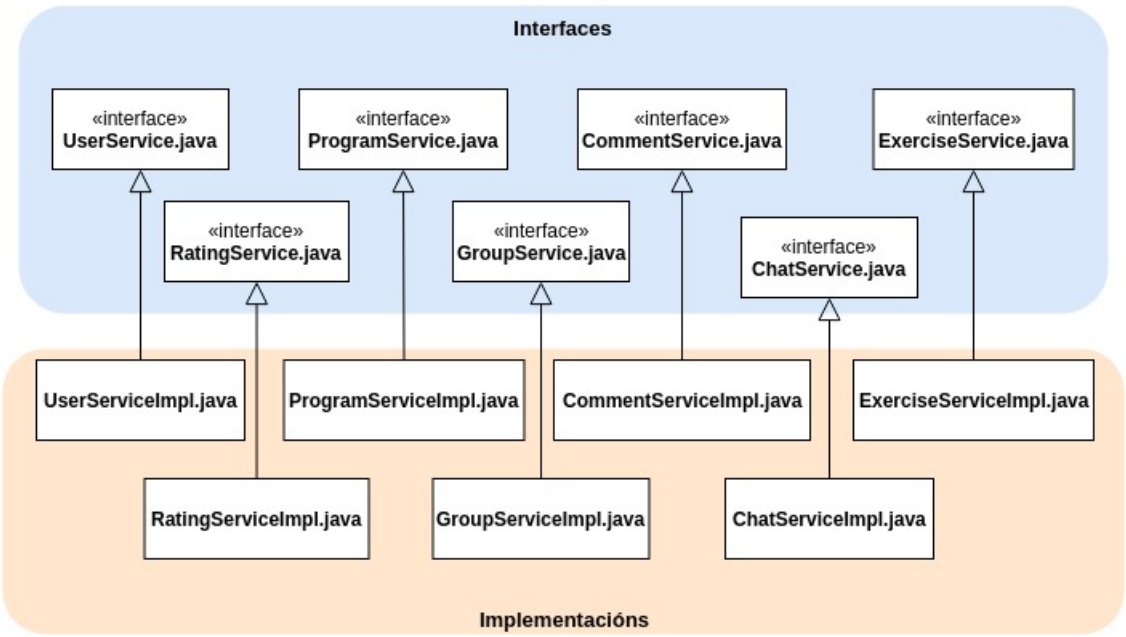


Figura 8.7: Diagrama dos Servizos da Capa Modelo.

Para a implementación da capa da lóxica de negocio, creouse unha interface para cada grupo de funcionalidades relacionadas entre si, que será consumida pola capa de servizos REST. En *UserService* agrúpanse as operacións referentes ao manexo de usuarios como rexistro, inicio de sesión, cambiar datos do perfil, etc. En *ProgramService* atópanse as funcións correspondentes á xestión dos programas por parte dos usuarios como pode ser creación, consulta ou actualización, entre outras. No servizo *CommentService* xuntáronse as funciona-

lidades que teñen que ver coa creación, borrado e consulta de comentarios. En *ExerciseService*, todas as operacións que se refiren ao tratamento de exercicios dentro dos grupos. Na interface *RatingService* encóntranse as funcionalidades para levar a cabo valoracións dun programa. En *GroupService*, as funcións correspondentes ao manexo de grupos. E por último, no servizo *ChatService* inclúese a funcionalidade relativa á creación e consulta das mensaxes dun chat nun grupo determinado.

Para cada interface, desenvolveuse a súa correspondente clase de implementación, na que se inxectan os DAOs que necesita por medio do contedor de dependencias de Spring. Cada clase realiza todas operacións requeridas contra a capa de acceso a datos que lle solicita a capa superior 8.4.6. Na figura 8.7 detállanse todos os servizos existentes na capa modelo da aplicación. A continuación, móstranse todas as entradas das que dispón cada servizo e unha breve descrición para cada unha.

Interface de **UserService**:

- *void* ***signUp****(User user) throws DuplicateInstanceException;*

  Este método serve para rexistrar un usuario novo.

- *User* ***login****(String userName, String password) throws IncorrectLoginException;*

  Inicia sesión para un usuario rexistrado.

- *User* ***saveLastLoginDate****(Long userId) throws InstanceNotFoundException;*

  Almacendar en BBDD a última data e hora de inicio sesión para mostrar a próxima vez que se inicie sesión.

- *User* ***loginFromId****(Long id) throws InstanceNotFoundException;*

  Inicia sesión a partir do identificador de usuario.

- *User* ***updateProfile****(Long id, String firstName, String lastName, String email) throws InstanceNotFoundException;*

  Actualiza os datos de perfil dun usuario.

- *void* ***changePassword****(Long id, String oldPassword, String newPassword) throws InstanceNotFoundException, IncorrectPasswordException;*

  Cambia o contrasinal dun usuario.

- *User* ***getUserInfo****(Long id) throws InstanceNotFoundException;*

  Devolve a información dun usuario a partir do seu identificador.

- *User* ***getUserInfo****(String username) throws InstanceNotFoundException;*

  Devolve a información dun usuario a partir do seu *username*.

- *List<User>* ***getUsersByKeyword****(String keyword);*

  A partir de palabras chave, devolve unha lista de usuarios cuxos nomes, apelidos ou email coinciden coa búsqueda.

Interface de **ProgramService**:

*- Program **createProgram**(Long userId, Long exerciseId, String programName, String programDesc, String code, Boolean privateProgram) throws InstanceNotFoundException;*

Crea un programa novo para un usuario. Opcionalmente pode existir un *exerciseId* que indica que ese programa fará referencia ao exercicio con ese identificador e será unha das súas solucións.

*- void **deleteProgram**(Long id) throws InstanceNotFoundException;*

Elimina un programa.

*- Program **getProgram**(Long id) throws InstanceNotFoundException;*

Devolve os datos dun programa a partir do seu identificador.

*- Program **updateProgram**(Long id, String programName, String programDesc, String code) throws InstanceNotFoundException;*

Actualiza os datos dun programa.

*- List<Program> **getProgramsByUser**(Long userId) throws InstanceNotFoundException;*

Devolve a lista de programas creados por un usuario concreto.

*- Program **setPublic**(Long id) throws InstanceNotFoundException;*

Establece un programa como público en base ao seu identificador.

*- Program **setPrivate**(Long id) throws InstanceNotFoundException;*

Establece un programa como privado en base ao seu identificador.

*- List<Program> **getPublicProgramsByUser**(Long userId) throws InstanceNotFoundException;*

Devolve a lista de programas **públicos** dun usuario.

*- void **shareProgram**(Long userId, Long programId) throws InstanceNotFoundException;*

Comparte un programa co usuario que se lle indica.

*- void **unshareProgram**(Long userId, Long programId) throws InstanceNotFoundException;*

Deixa de compartir un programa co usuario indicado.

*- List<Program> **getSharedProgramsWithMe**(Long userId) throws InstanceNotFoundException;*

Devolve unha lista cos programas compartidos cun determinado usuario.

*- List<User> **getSharedUsersByProgram**(Long programId) throws InstanceNotFoundException;*

Devolve unha lista cos usuarios cos que se compartiu un programa.

*- List<Program> **getSharedProgramsWithMeByUser**(Long myUserId, Long otherUserId) throws InstanceNotFoundException;*

Devolve unha lista de programas compartidos co usuario con id *myUserId*, que son propiedade do usuario con id *otherUserId*.

*- List<Program> **getProgramsByKeyword**(String keyword);*

Devolve unha lista de programas buscando coincidencias co título e coa descrición en base ás palabras da búsqueda.

- List<Program> ***getProgramsByExercise****(Long exerciseId) throws InstanceNotFoundException;*

Devolve a lista de programas asociados a un exercicio, é dicir, as súas solucións propostas polos membros do grupo no que se propuxo o exercicio.

Interface de **CommentService**:

- Comment ***createComment****(Long userId, Long programId, String commentText) throws InstanceNotFoundException;*

Crea un novo comentario para un programa por parte dun usuario calquera que ten acceso a ese programa.

- Comment ***replyComment****(Long commentOrigId, Long userId, Long programId, String commentText) throws InstanceNotFoundException;*

Crea un novo comentario que fai referencia a outro xa creado, é dicir, unha resposta ao orixinal con identificador *commentOrigId*.

- void ***deleteComment****(Long commentId) throws InstanceNotFoundException;*

Elimina un comentario.

- List<Comment> ***getCommentsByProgram****(Long programId) throws InstanceNotFoundException;*

Devolve os comentarios asociados a un programa con identificador *programId*.

- List<Comment> ***getCommentReplies****(Long commentId) throws InstanceNotFoundException;*

Devolve as respostas dun comentario concreto.

- Comment ***getComment****(Long commentId) throws InstanceNotFoundException;*

Devolve a información dun comentario en base ao seu identificador.

Interface de **ExerciseService**:

- Exercise ***createExercise****(String statementText, Long groupId, Long userId, Calendar expirationDate) throws InstanceNotFoundException;*

Un usuario (*userId*) crea un novo exercicio dentro dun grupo determinado (*groupId*), cunha data de expiración.

- void ***deleteExercise****(Long exerciseId) throws InstanceNotFoundException;*

Elimina un exercicio.

- Exercise ***getExercise****(Long exerciseId) throws InstanceNotFoundException;*

Devolve a información dun exercicio determinado.

- Exercise ***updateExercise****(Long exerciseId, String statementText, Calendar expirationDate) throws InstanceNotFoundException;*

Actualiza os datos dun exercicio.

- List<Exercise> ***getExercisesByGroup****(Long groupId) throws InstanceNotFoundException;*

Devolve a lista de exercicios asociados a un grupo concreto.

RatingService

*- Rating **createRating**(Long userId, Long programId, Float rating) throws InstanceNotFoundException, DuplicateInstanceException;*

Crea unha nova valoración para un programa por parte dun usuario concreto.

*- void **deleteRating**(Long userId, Long programId) throws InstanceNotFoundException;*

Elimina unha valoración.

*- List<Rating> **getProgramRatings**(Long programId) throws InstanceNotFoundException;*

Devolve as valoracións realizadas para un programa.

Interface de **GroupService**:

*- UsersGroup **createGroup**(Long userId, String groupName) throws InstanceNotFoundException;*

Crea un novo grupo.

*- void **deleteGroup**(Long groupId) throws InstanceNotFoundException;*

Elimina un grupo creado previamente.

*- UsersGroup **getGroup**(Long groupId) throws InstanceNotFoundException;*

Devolve a información dun grupo.

*- UsersGroup **updateGroup**(Long groupId, String groupName, Boolean chatEnable) throws InstanceNotFoundException;*

Actualiza os datos dun grupo.

*- List<User> **getAllUsersByGroup**(Long groupId) throws InstanceNotFoundException;*

Devolve todos os usuarios que son membros dun grupo.

*- List<UsersGroup> **getAllGroupsByUser**(Long userId) throws InstanceNotFoundException;*

Devolve todos os grupos aos que

*- void **addMember**(Long userId, Long groupId) throws InstanceNotFoundException, DuplicateInstanceException;*

Engade un novo membro a un grupo.

*- void **removeMember**(Long userId, Long groupId) throws InstanceNotFoundException, DuplicateInstanceException;*

Elimina un membro dun grupo.

Interface de **ChatService**:

*- ChatMessage **createMessage**(Long groupId, Long userId, String messageText, Calendar messageDate) throws InstanceNotFoundException;*

Crea unha nova mensaxe no chat dun grupo por parte dun usuario.

*- List<ChatMessage> **getGroupMessages**(Long groupId) throws InstanceNotFoundException;*

Devolve as mensaxes do chat dun grupo concreto.
- List<ChatMessage> ***getGroupMessagesByDate****(Long groupId, Calendar begin, Calendar end)throws InstanceNotFoundException;*

Devolve unha lista coas mensaxes dun chat dun grupo a partir dunha data e hora determinada.

Por último, explícanse as excepcións que aparecen na firma dos métodos listados:

- *InstanceNotFoundException*: indica que a entidade á que se fai referencia non existe.
- *DuplicateInstanceException*: este erro prodúcese ao querer crear unha entidade cos mesmos datos que outra xa existente, cando eso non está permitido por regras de unicidade.
- *IncorrectLoginException*: inicio de sesión incorrecto por proporcionar unhas credenciais inválidas.

8.4.6 Capa Servizos Web

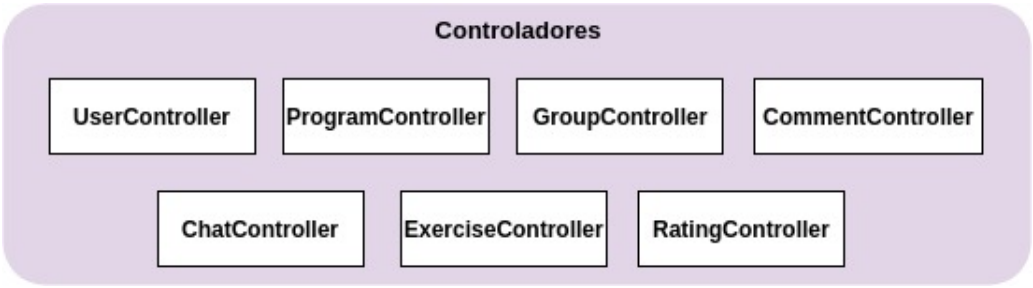


Figura 8.8: Controladores do Servizo Web.

A capa que ofrece a API REST tamén está divida segundo os distintos grupos de funcionalidades presentes na aplicación. Existen sete controladores, tal e como se ve na figura 8.8, que recollen as peticións que realizará o cliente. Acto seguido, acceden á capa de servizos do modelo e devolven a resposta en formato JSON cos datos necesarios encapsulados nun DTO. En cada controlador inxéctanse os servizos da capa da lóxica de negocio que precisa por medio da anotación *Autowired* que proporciona Spring para a inxección de dependencias.

Para cada recurso respetouse a semántica dos tipos de petición HTTP. Para as consultas de datos, peticións GET; para as modificacións, peticións PUT; e para creacións e borrado fíxose uso en ambos casos da petición POST. A continuación documéntanse os *endpoints* da API REST. A maiores do indicado, cabe destacar que para todas as operacións é preciso enviar o *token* de acceso.

Para o controlador de **usuarios** definíronse as seguintes operacións:

- /users/signUp: rexistra un novo usuario.
- /users/login: inicia sesión para un usuario rexistrado.
- /users/lastlogin: devolve a data e hora do último inicio de sesión.
- /users/{id}: actualiza os datos dun usuario.
- /users/{id}/changePassword: cambia o contrasinal dun usuario.
- /users/info/{id}: devolve a información dun usuario.
- /users/publicinfo/{username}: devolve só a información de carácter público dun usuario.
- /users/search/{keywokd}: busca usuarios por palabras chave (*username*, nome, apelido e *email*).

Recurso	**Método**	**Excepcións**
/users/signUp	POST	DuplicateInstanceException
/users/login	POST	IncorrectLoginException, InstanceNotFoundException
/users/lastlogin	POST	InstanceNotFoundException
/users/ loginFromServiceToken	POST	InstanceNotFoundException
/users/{id}	PUT	InstanceNotFoundException, PermissionException
/users/{id}/ changePassword	POST	PermissionException, InstanceNotFoundException, IncorrectPasswordException
/users/info/{id}	GET	InstanceNotFoundException
/users/publicinfo/ {username}	GET	InstanceNotFoundException
/users/search/{keywokd}	GET	Ningunha

Táboa 8.1: API de UserController.

Para o controlador dos **programas** defínense os seguintes recursos:

- /programs/create: crea un programa novo.
- /programs/delete: elimina un programa.
- /programs/{programId}: devolve os datos dun programa concreto.

- /programs/example: devolve o programa de exemplo, e.g. para mostrar na páxina principal.

- /programs/fullinfo/{programId}: devolve toda a información do programa e lista de usuarios cos que se compartiu.

- /programs/udpate: actualiza os datos dun programa.

- /programs/all/{id}: devolve todos programas dun usuario.

- /programs/setpublic: establece un programa como público.

- /programs/setprivate: establece un programa como privado.

- /programs/public/{userId}: devolve os programas públicos dun usuario.

- /programs/share: comparte un programa cun usuario.

- /programs/unshare: deixa de compartir un programa cun usuario.

- /programs/shared/{id}: devolve os programas que foron compartidos cun usuario.

- /programs/shared/{myUserId}/{otherUserId}: devolve os programas que foron compartidos cun usuario por parte doutro usuario concreto.

- /programs/search/{keyword}: busca programas por palabras chave (título e descrición).

- /programs/exercise/{exerciseId}: devolve os programas asociados a un exercicio (solucións).

Recurso	Método	Excepcións
/programs/create	POST	InstanceNotFoundException, PermissionException
/programs/delete	POST	InstanceNotFoundException, PermissionException
/programs/{programId}	GET	InstanceNotFoundException, PermissionException
/programs/example	GET	InstanceNotFoundException
/fullinfo/{programId}	GET	InstanceNotFoundException, PermissionException
/programs/update	PUT	InstanceNotFoundException, PermissionException
/programs/all/{id}	GET	InstanceNotFoundException
/programs/setpublic	PUT	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/programs/setprivate	PUT	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/programs/public/{userId}	GET	InstanceNotFoundException
/programs/share	PUT	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/programs/unshare	PUT	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/programs/shared/{id}	GET	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/programs/shared/{myUserId}/{otherUserId}	GET	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/programs/search/{keyword}	GET	Ningunha
/programs/exercise/{exerciseId}	GET	InstanceNotFoundException, DuplicateInstanceException, PermissionException

Táboa 8.2: API de ProgramController.

Dentro do controlador que contén as operacións de **grupos** están:

- /groups/create: crea un grupo.
- /groups/delete: borra un grupo.
- /groups/{groupId}: devolve a información dun grupo en concreto.

- /groups/update: actualiza os datos dun grupo.
- /groups/allgroups/{id}: devolve todos os grupos aos que pertence un usuario.
- /groups/addmember: engade un novo membro ao grupo.
- /groups/removemember: elimina un membro dun grupo.

Recurso	**Método**	**Excepcións**
/groups/create	POST	InstanceNotFoundException, PermissionException
/groups/delete	POST	InstanceNotFoundException, PermissionException
/groups/{groupId}	GET	InstanceNotFoundException, PermissionException
/groups/update	PUT	InstanceNotFoundException, PermissionException
/groups/allgroups/{id}	GET	InstanceNotFoundException
/groups/addmember	PUT	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/groups/removemember	PUT	InstanceNotFoundException, DuplicateInstanceException, PermissionException

Táboa 8.3: API de GroupController.

No controlador que manexa as operacións correspondentes aos **comentarios** temos as seguintes operacións:

- /comment/create: crea un comentario.
- /comment/reply: crea unha resposta a un comentario.
- /comment/delete: borra un comentario.
- /comment/program/{programId}: devolve os comentarios dun programa.
- /comment/comment/{commentId}: devolve as respostas a un comentario.

Recurso	Método	Excepcións
/comment/create	POST	InstanceNotFoundException, PermissionException
/comment/reply	POST	InstanceNotFoundException, PermissionException
/comment/delete	POST	InstanceNotFoundException, PermissionException
/comment/program/{programId}	GET	InstanceNotFoundException, PermissionException
/comment/comment/{commentId}	GET	InstanceNotFoundException, PermissionException

Táboa 8.4: API de CommentController.

O controlador que contén as operacións relativas ao **chat** dos grupos contén os seguintes recursos:

- /chat/create: crea unha mensaxe.
- /chat/all/{groupId}: devolve todas as mensaxes dun grupo.
- /chat/allbydate: devolve as mensaxes dun grupo a partir dunha data e hora.

Recurso	Método	Excepcións
/chat/create	POST	InstanceNotFoundException, PermissionException
/chat/all/{groupId}	GET	InstanceNotFoundException, PermissionException
/chat/allbydate	GET	InstanceNotFoundException, PermissionException

Táboa 8.5: API de ChatController.

No controlador para as operacións relativas a **exercicios** atopamos as seguintes operacións:

- /exercise/create: crea un exercicio.
- /exercise/delete: borra un exercicio.
- /exercise/{exerciseId}: devolve a información dun exercicio.
- /exercise/update: actualiza os datos dun exercicio.
- /exercise/group/{groupId}: devolve os exercicios dun determinado grupo.

Recurso	Método	Excepcións
/exercise/create	POST	InstanceNotFoundException, PermissionException
/exercise/delete	POST	InstanceNotFoundException, PermissionException
/exercise/{exerciseId}	GET	InstanceNotFoundException, PermissionException
/exercise/update	PUT	InstanceNotFoundException, PermissionException
/exercise/group/{groupId}	GET	InstanceNotFoundException, PermissionException

Táboa 8.6: API de ExerciseController.

Por último, no controlador que xestiona as operacións das **valoracións** atopamos os seguintes recursos:

- /rating/create: crear unha nova valoración.
- /rating/delete: borrar unha valoración previamente enviada.
- /rating/all/{programId}: devolver as valoracións dun programa.

Recurso	Método	Excepcións
/rating/create	POST	InstanceNotFoundException, DuplicateInstanceException, PermissionException
/rating/delete	POST	InstanceNotFoundException, PermissionException
/rating/all/{programId}	GET	InstanceNotFoundException, PermissionException

Táboa 8.7: API de RatingController.

En canto ás excepcións que se indican para os recursos da API REST, o código de erro asociado é:

- InstanceNotFoundException: **404 Not Found**
- PermissionException: **403 Forbidden**
- DuplicateInstanceException: **400 Bad Request**
- IncorrectLoginException: **404 Not Found**

8.5 Subsistema Aplicación Web (Frontend)

8.5.1 Obxectivos

Este segundo subsistema corresponde á aplicación web desenvolta co framework Angular[10] (*frontend*). Nesta parte do sistema trátase de seguir a mesma filosofía que ata o de agora, e codificar tendo en conta as mellores estratexias posibles para favorecer o desenvolvemento e que o software resultante sexa facil de entender e de manter.

8.5.2 Arquitectura

Este módulo divídese en dúas capas diferentes, por un lado a capa correspondente á interface de usuario e por outro a capa de acceso a datos. Deste xeito, a modificación das funcións que conteñen as peticións que se realizan ao *backend* non inflúen na interface que as consulta e viceversa.

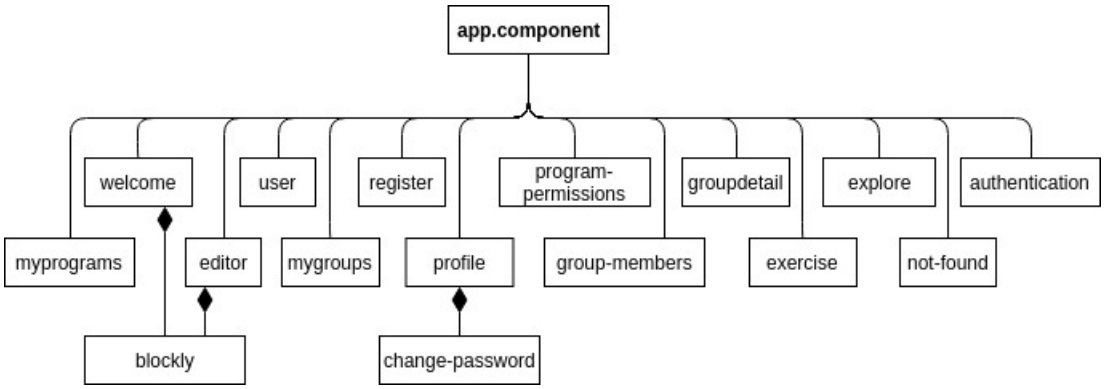


Figura 8.9: Diagrama de Compoñentes da Capa Web.

A primeira delas, seguindo o modo de traballo de Angular, está formada por compoñentes que equivalen a páxinas ou partes dunha páxina da aplicación web. Existe un compoñente principal do que parten todos os demais grazas ao *Router* de Angular. Este *Router* permite que nun ficheiro se configure con que URL se corresponde cada compoñente para redirixir a el. Dentro dun compoñente pode haber máis compoñentes que se separen debido a que conforman unha sección dunha páxina demasiada complexa. Na figura 8.9 móstranse todos os compoñentes que forman esta capa. Nela apréciase que todos eles, parten do compoñente principal *app.component.ts* e o *Router* é o encargado de decidir que compoñente se carga. No caso dos compoñentes *blockly* e *change-password*, están incluídos dentro doutros compoñentes para separar a súa lóxica que conta con certa complexidade.

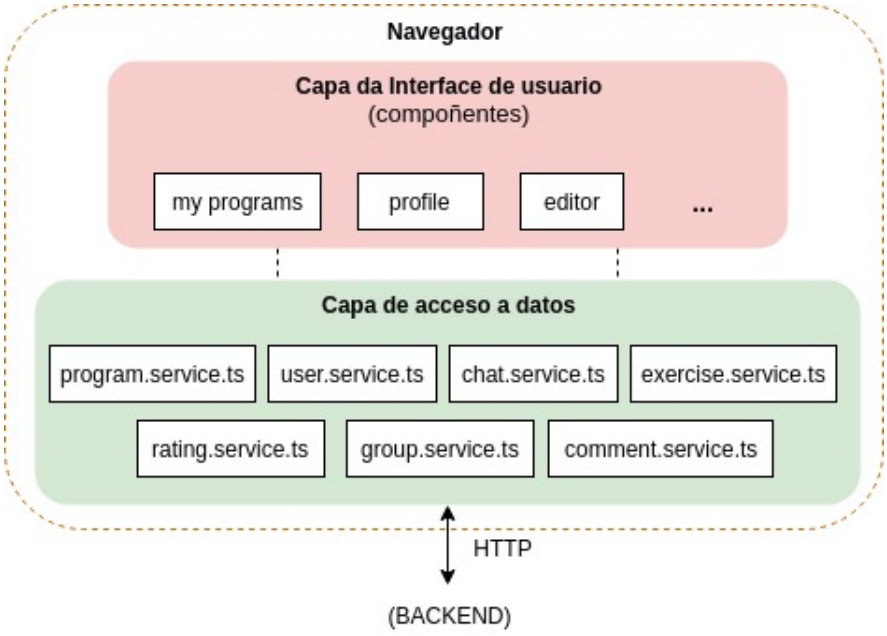


Figura 8.10: Diagrama da Capa de Acceso a Datos de Frontend.

A segunda capa está formada por *servizos*, segundo a nomenclatura de Angular, que se inxectan nos compoñentes para que estes os poidan empregar. Cando un compoñente precisa realizar unha operación no *backend*, accede ao servizo correspondente que leva a cabo unha petición por HTTP á API REST do servidor web e espera unha resposta en formato JSON. Na figura 8.10 detállanse todos os arquivos que prestan algún servizo na capa web. Os que están agrupados na área verde forman a capa de acceso a datos e cada un deles manexa funcionalidades similares, de feito coinciden cos controladores do servizo REST que se comentan na sección 8.4.6. A maiores, existen outros servizos para o funcionamento da biblioteca Blockly[1] (*toolbox.service.ts* e *blockly-aux.service.ts*) e para xestionar a autenticación na aplicación web (*auth.service.ts* e *auth-guard.service.ts*).

8.5.3 Capa Web

A capa de interface de usuario está formada por unha serie de compoñentes que corresponden a funcionalidade concreta nunha páxina dada. Cada compoñente está definido por unha clase que contén a súa propia lóxica e na que garda información, ademais dun modelo HTML asociado e opcionalmente unha folla de estilos CSS. Cando a plantilla HTML se carga, pode depender dos valores que a clase que contén a lóxica lle proporciona. Mentres o usuario fai uso da aplicación, a clase da lóxica e interface HTML comunícanse e esta última modífica-se de forma dinámica. Tamén cabe destacar, respecto da internacionalización da aplicación, que nos modelos HTML inclúense *pipes* (transforman información dada en base a algún pa-

rámetro) para coller o texto requerido dun arquivo de traducións. A aplicación ten soporte para galego e inglés. Por último, en canto ao listado de elementos (e.g. programas ou grupos) na interface web, implementouse un sistema de paxinación para evitar que as páxinas se extendan indefinidamente.

Capítulo 9

Implementación

9.1 Software Requerido

Neste apartado indícase o software preciso para a implementación do proxecto. En primeiro lugar, cabe destacar que o sistema operativo empregado para o desenvolvemento e proba do proxecto foi **Ubuntu**, unha distribución de Linux. Respecto ao servidor web (*backend*), para a súa implementación precísase:

- Un Sistema de Xestión de Bases de Datos **MySQL 5+**[25].
- O kit de desenvolvemento de Java, **Java SE 8+**.
- A ferramenta de xestión de proxectos Java, **Maven**[7], para o empaquetado da aplicación.

Pola súa parte, para o desenvolvemento do *frontend*, o software necesario é:

- O entorno en tempo de execución, Node.js (**Node 8+**).
- Debido a que Angular e Angular CLI dependen das características e funcionalides proporcionadas polas bibliotecas que están dispoñibles como paquetes npm, é preciso instalar o Sistema de Xestión de Paquetes de Node, (**npm 3+**).
- **Angular CLI**, para a creación, desenvolvemento, construción e despregue da aplicación en Angular.

9.2 Estrutura

Como se comentou no capítulo de deseño, a aplicación divídese en dúas partes ben diferenciadas e así se reflexa no directorio raíz do proxecto. Nunha das carpetas está o *backend* e na outra o *frontend* (ademais do arquivo README.md).

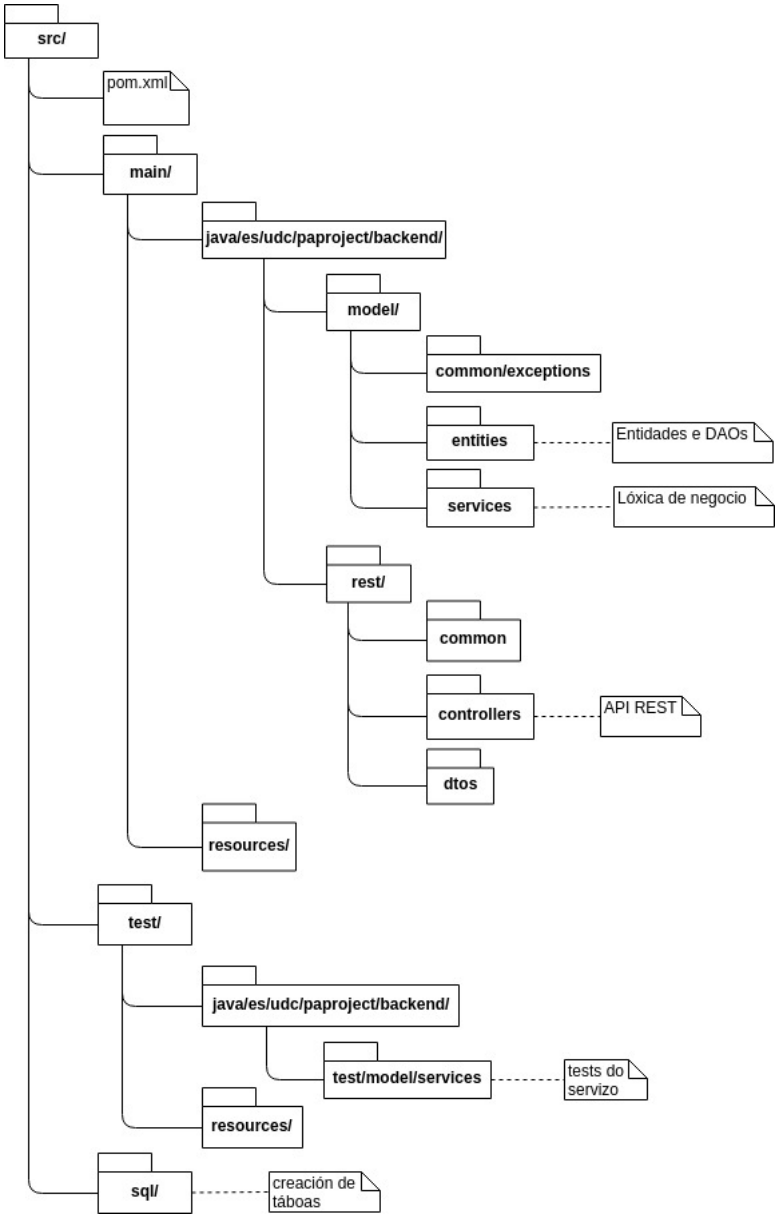


Figura 9.1: Estrutura do Backend.

Nesta parte do proxecto faise uso da ferramenta Maven[7], a cal segue un determinado patrón de configuración. É por isto, que a estrutura de directorios debe ter a forma que se amosa na figura 9.1.

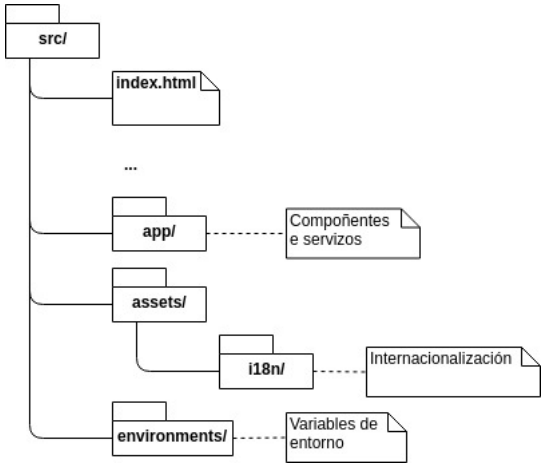


Figura 9.2: Estrutura do Frontend.

Neste caso, tamén se depende das ferramentas empregadas en canto á estrutura do *frontend* e é Angular[10] o que define automaticamente a xerarquía de directorios que se aprecia na figura 9.2 ao crear a aplicación.

9.3 Instrucións de Compilación

Para a compilación do proxecto, en primeiro lugar debe estar preparada a Base de Datos e para arrancar o servidor de MySQL execútase o comando:

- **$ mysqld**

e a continuación creamos as BBDD de desenvolvemento e de probas:

- **$ mysqladmin -u root create socialblockly**
- **$ mysqladmin -u root create socialblocklytest**
- **$ mysql -u root** (entramos na consola de mysql)
- CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';
- GRANT ALL PRIVILEGES ON socialblockly.* to 'admin'@'localhost' WITH GRANT OPTION;
- GRANT ALL PRIVILEGES ON socialblocklytest.* to 'admin'@'localhost' WITH GRANT OPTION;
- exit

Para arrancar o servidor web é necesario entrar na carpeta (*backend*). Soamente a primeira vez que se arranque o servidor, débese executar previamente o seguinte comando para crear as táboas:

- **$ mvn sql:execute**

e para arranchar o servidor web debe escribirse en consola:

- **$ mvn spring-boot:run**

No caso de querer executar as probas, débese poñer na liña de comandos:

- **$ mvn test**

A continuación, para a parte do *frontend* débese entrar no seu directorio correspondente e executar o seguinte comando para descargar todas as bibliotecas necesarias (só a primeira vez):

- **$ npm i**

e para arracar a aplicación web é suficiente con executar este último comando e automaticamente abrirase o navegador web predeterminado en *http://localhost:4200/*:

- **$ ng serve -o**

Capítulo 10

Probas

10.1 Introdución

No desenvolvemento de calquera software, e máis nunha aplicación empresarial, non é suficiente con que funcione, senón que debe funcionar ben. É por iso que o desenvolvemento de probas é fundamental para probar todos os casos que se poden producir ante o uso do sistema por parte do usuario final.

Seguindo as boas prácticas recomendadas á hora de implementar *tests*, todas as probas que se realizan son independentes unhas das outras, polo que a orde de execución das mesmas é indiferente e a execución dunha non influe nas outras. Ademais, cóntase cunha Base de Datos diferente da utilizada pola aplicación cando está en execución para evitar así comprometer os seus datos e que estes non afecten ás probas.

10.2 Probas de Integración

Contra a capa modelo do servidor web realizánse probas para todas as operacións presentes no servizos da capa lóxica de negocio, tendo en conta os casos de erro forzando que salten as excepcións pertinentes. Na figura 10.1 móstranse os ficheiros de *tests*, cada un relacionado coa interface do servizo sobre o que realiza as probas. Aínda que, a pesar de que cada ficheiro está orientado a probar as operación dun determinado servizo, pode que precise algún outro servizo para realizar os pasos previos necesarios para chegar a ese punto xa que cada proba debe ser independente.

Para a automatización de ditas probas faise uso do framework JUnit4[15]. Cada caso de proba defínese coa anotación *@Test* e no caso de esperar unha excepción, explicítase nesa mesma anotación, e.g. *@Test(expected = InstanceNotFoundException.class)*. En cada ficheiro de probas inxéctanse as interfaces dos servizos necesarios para cada arquivo de tests coa anotación *@Autowired* para que Spring se encargue de proporcionar unha instancia da im-

plementación de cada servizo.

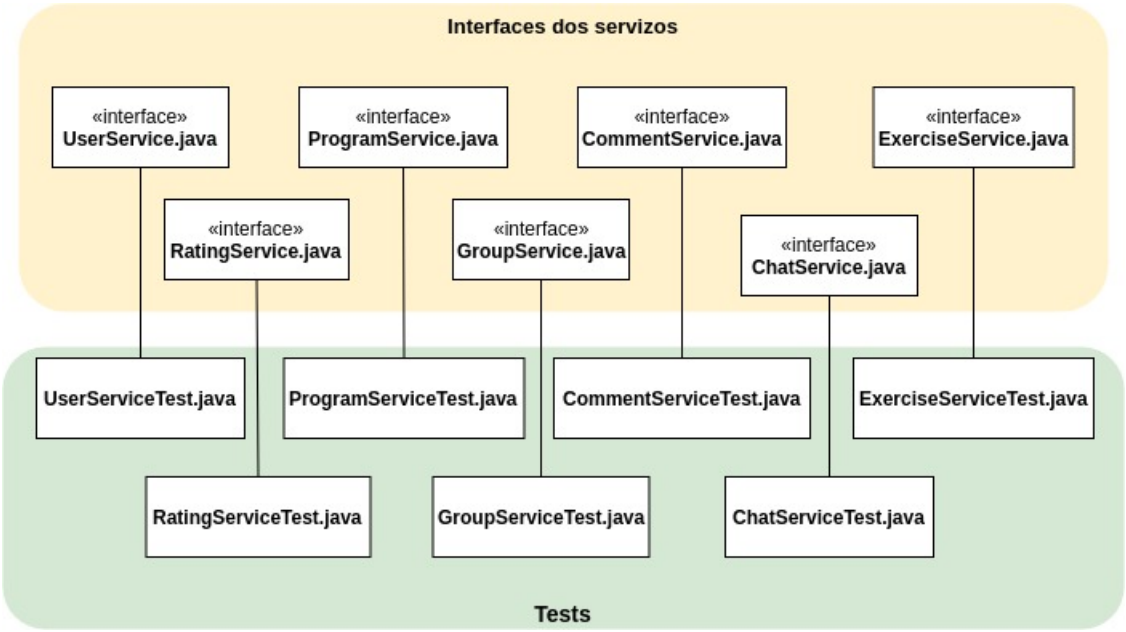


Figura 10.1: Diagrama das Clases de Proba.

Para lanzar os *tests* dende a consola pódese executar o comando **$mvn test**. Utilizando este mesmo comando, co fin de analizar a cobertura das probas no código, tamén se lanza o plugin de JaCoCo[16] o cal nos mostra os resultados obtidos e que se mostran na figura 10.2.

Social blockly > es.udc.paproject.backend.model.services

es.udc.paproject.backend.model.services

Element	Missed Instructions	Cov.	Missed Branches	Cov.
UserServiceImpl		71%		80%
GroupServiceImpl		97%		82%
ProgramServiceImpl		96%		94%
ChatServiceImpl		87%		75%
RatingServiceImpl		95%		92%
IncorrectLoginException		60%		n/a
CommentServiceImpl		100%		100%
ExerciseServiceImpl		100%		100%
PermissionCheckerImpl		100%		100%

Figura 10.2: Cobertura das Probas de Integración.

10.3 Probas sobre a API REST

Para realizar as probas contra a capa de servizos REST, é dicir, os controladores en *backend* que reciben as petición para o servidor web, empregouse Postman[17]. Con esta ferramenta leváronse a cabo as peticións contra os *endpoints* definidos na sección 8.4.6, como se aprecia na figura 10.3, lembrando incorporar un *token* de acceso válido, que se extraeu coas ferramentas de desenvolvemento do navegador.

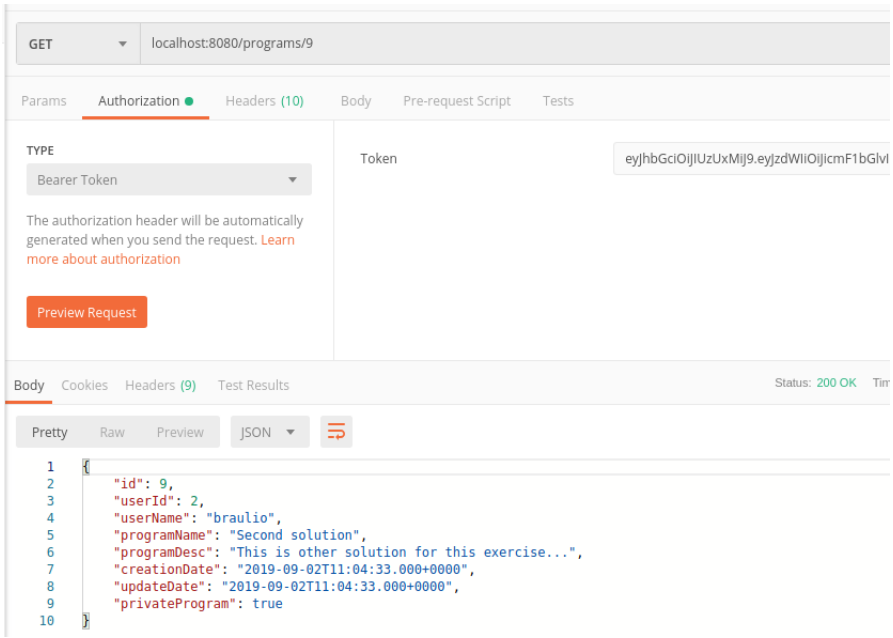


Figura 10.3: Captura da Ferramenta Postman.

10.4 Probas de Aceptación

Para asegurar que a aplicación construída cumpre cos requisitos que demandaba o cliente é necesario realizar probas de aceptación. Estas probas executáronse sobre a aplicación web xa rematada, sen atender á implementación da mesma, senón á funcionalidade e, idealmente, levaríaas a cabo o cliente. Nesta fase débense seguir os requisitos que propuxo o cliente e todos eles foron plasmados nos Casos de Uso (apartado 7.3), polo que para realizar o plan de probas seguiuse esta lista, executando os pasos indicados para cada un e comprobando que o resultado é o correcto.

Capítulo 11

Conclusións e Futuras Liñas de Traballo

11.1 Conclusións

Neste proxecto pretendéronse varios obxectivos relativos a un propósito común, a iniciación no mundo da computación de calquera persoa. As metas establecidas foron as seguintes:

- Satisfacer as necesidades de algúen que desexara aprender a programar, sen ningunha experiencia previa, traballando cunha ferramenta que lle proporcionara unha forma sinxela e intuitiva de facelo.
- Integrar unha biblioteca que se encargara dos aspectos visuais dun editor que permitise arrastrar e encadear pezas para formar secuencias executables.
- Permitir a xestión dos seus propios programas, por parte dos usuarios, dentro da súa conta persoal e facelos públicos ao resto do mundo se así o desexa.
- Orientar a aplicación a un uso nas aulas para que profesores de escolas e institutos puideran empregar este software como unha ferramenta docente nas súas clases, engadindo funcionalidades para a creación e manexo de grupos.
- Desenvolver a aplicación cun deseño baseado en capas que favoreza a súa posible evolución e o seu mantemento.

En canto a estes aspectos globais, cumpríronse todos os obxectivos propostos dende un principio e deuse forma a unha aplicación que respectou todos estes requisitos. Un obxectivo adicional, e non pouco importante, foi tamén familiarizarse coas tecnoloxías usadas e adquirir experiencia no desenvolvemento dunha aplicación web destas características, e esta meta tamén se conseguiu.

11.2 Futuras Liñas de Traballo

Grazas ao deseño da aplicación, é posible e estender a súa funcionalidade para satisfacer máis necesidades que poidan surxir por parte dos usuarios. Algunhas das melloras poderían ser:

- Amosar recomendacións de programas ou usuarios na pantalla principal ou na de explorar sen necesidade de ter que buscar algo en concreto.
- Integrar o sistema con algunha rede social para que os usuarios poidan compartir facilmente os traballos realizados e axudar, así, á promoción da aplicación.
- A aplicación xa conta cun sistema de internacionalización, polo que sería interesante traducila a máis idiomas.
- Adaptar a aplicación a dispositivos móbiles para que poida ser usada por exemplo nunha tablet.

Apéndices

Apéndice A

Glosario de acrónimos

IDE *Integrated Development Environment*

BBDD *Base de Datos*

JDBC *Java Database Connectivity*

DAO *Data Access Object*

DTO *Data Transfer Object*

Apéndice B

Glosario de termos

Blockly Biblioteca de JavaScript para construir editores de programación mediante elementos visuais.

Backend Parte do servidor dunha aplicación web.

Frontend Parte cliente ou capa web no caso dunha aplicación web.

Workspace Lugar de traballo

Mockup Esbozo dunha parte da interface dun software.

Apéndice C

Material adicional

C.1 Instalación do Software

PARA o funcionamento da aplicación establécense os seguintes pre-requisitos:

- Ter instalado un servidor Apache.
- Ter instalado o Sistema de Xestión de Bases de Datos MySQL[25] (MySQL 5+).

Para arrancar a aplicación débense seguir os pasos que se detallan a continuación:

- En primeiro lugar, débese descomprimir o arquivo MendezAgra_Braulio_TFG_2019_anexo_executables.zip contido no CD.
- Despois débense crear as táboas na Base de Datos:
  $ mysqladmin -u root create socialblockly
  $ mysql -u root (entramos na consola de mysql)
  CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';
  GRANT ALL PRIVILEGES ON socialblockly.* to 'admin'@'localhost' WITH GRANT OPTION;
  source /path/to/1-MySQLCreateTables.sql
  exit
- Para arrancar o *backend* da aplicación hai que executar o comando **java -jar social-blockly-1.0.0.jar**, onde *social-blockly-1.0.0.jar* é un dos arquivos extraídos no primeiro paso.
- A continuación, para arrancar o *frontend*, hai que incluír na ruta **/var/www/html** todos os arquivos contidos na carpeta *frontend*, a cal se extraeu no paso 1.

- Para o correcto funcionamento da aplicación, engadiuse un arquivo **.htaccess** na carpeta *frontend*. A maiores, é preciso engadir no arquivo **/etc/apache2/sites-enabled/000-default.conf** o seguinte:
  <Directory "/var/www/html">
  AllowOverride All
  </Directory>

- Por último, débense executar os comandos:
  sudo a2enmod rewrite
  sudo service apache2 restart.

- Para acceder á aplicación, é suficiente con abrir un navegador e entrar en **localhost**.

C.2 Contido do CD

Os arquivos contidos no CD son:

- **MendezAgra_Braulio_TFG_2019.pdf**: memoria do proxecto do fin de grao.

- **MendezAgra_Braulio_TFG_2019_resumo.pdf**: resumo do proxecto do fin de grao.

- **MendezAgra_Braulio_TFG_2019_anexo_codigo_fonte.zip**: código fonte da aplicación.

- **MendezAgra_Braulio_TFG_2019_anexo_executables.zip**: executables da aplicación.

C.3 Manual de Usuario

Neste apartado explícanse todas as accións posibles dentro da aplicación para que o usuario poida levalas a cabo coa maior facilidade posible. Como exemplo, rexistrarase un usuario e levaranse a cabo os diferentes casos de uso existentes, dando por suposto que conta con programas e grupos para os exemplos.

As explicacións deste manual agrúpanse por funcionalidades, de tal xeito que, en primeiro lugar, fálase do usuario non autenticado. A continuación, detállanse as posibles accións que pode levar a cabo un usuario xa autenticado divididas en: as relativas aos datos do seu perfil persoal, a xestión de programas, a xestión de grupos e a posibilidade de explorar outros programas e usuarios dentro da aplicación.

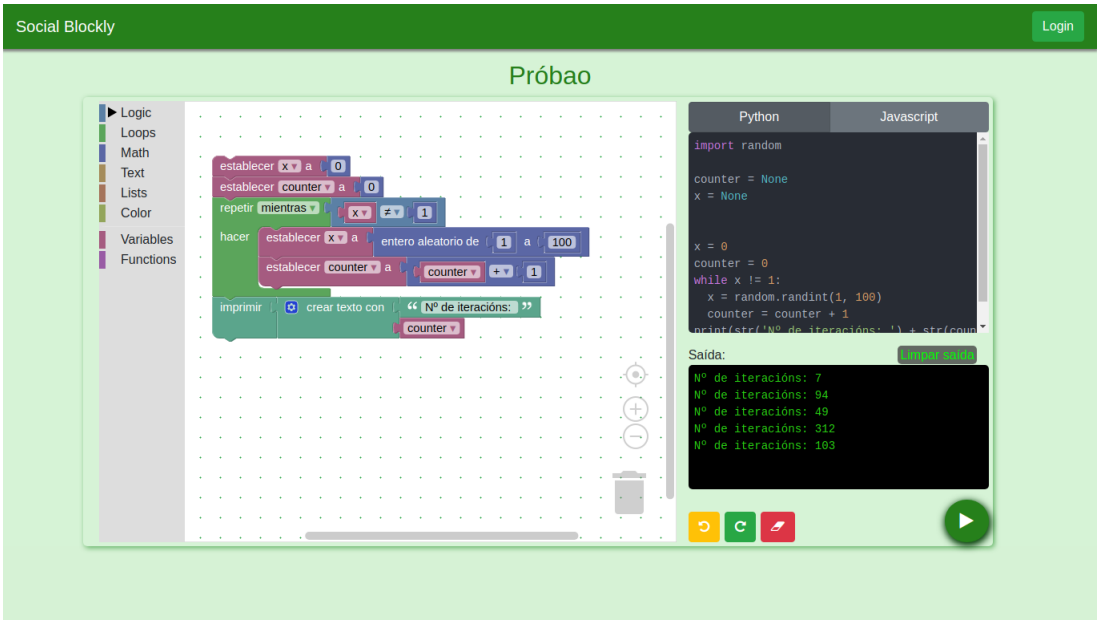


Figura C.1: Páxina principal da aplicación.

Na figura C.1 temos a páxina principal na que o usuario pode xogar coas pezas do programa de exemplo e executalo.

C.3.1 Rexistro e autenticación

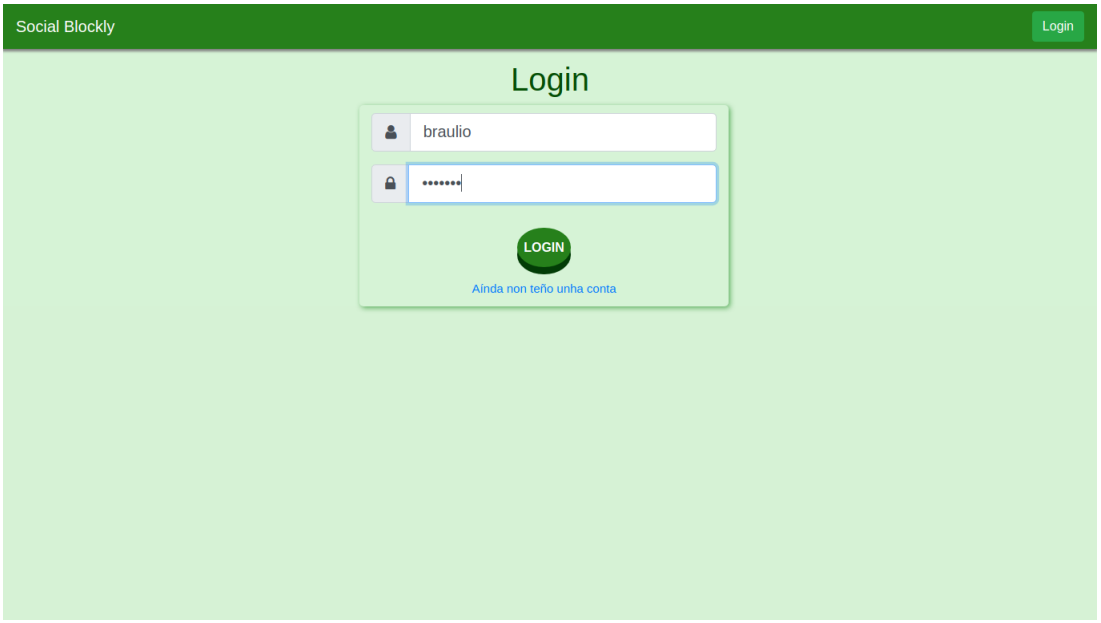


Figura C.2: Páxina de inicio de sesión.

Desde a páxina principal, pulsando no botón de "Login", accede á páxina de inicio de sesión que se ve na figura C.2. Nesa páxina, o usuario introduce as súas credenciais ou, no caso de que aínda non esté rexistrado, accede á páxina de rexistro da figura C.3.

Figura C.3: Páxina de rexistro.

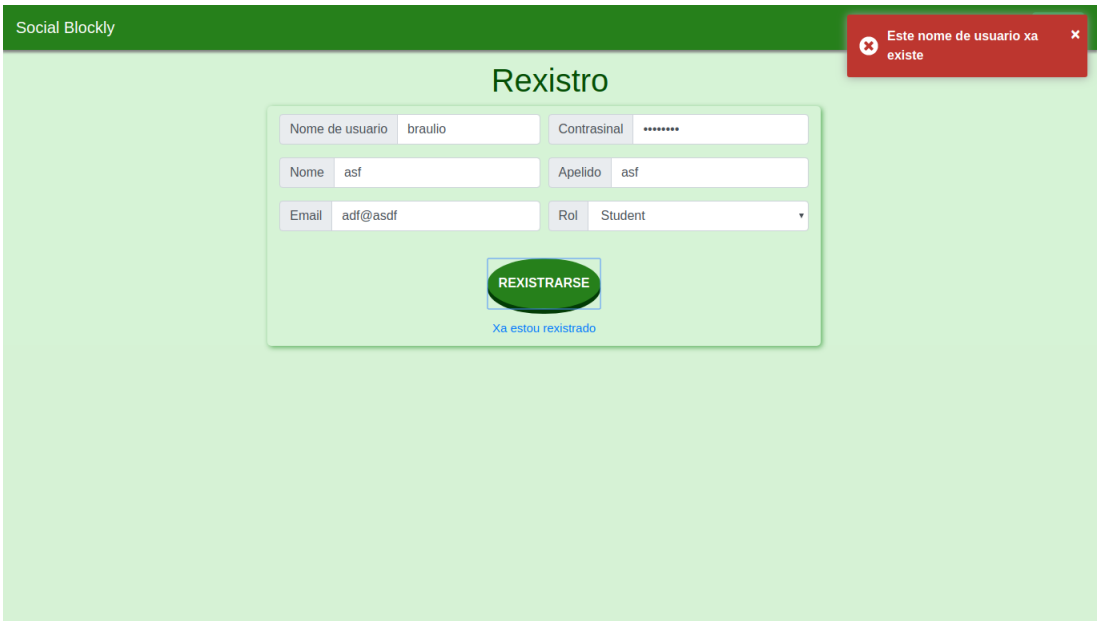


Figura C.4: Erro na páxina de rexistro.

No caso de que xa exista un usuario con ese *username* será notificado como se ve na

figura C.4. En xeral, calquera erro que se produza na aplicación, comunicaráselle ao usuario mediante estas notificación. As verdes para notificar casos de éxito e as vermellas nos casos de erro.

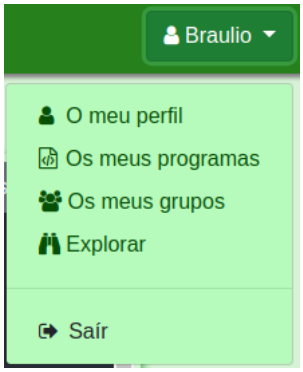


Figura C.5: Menú de usuario.

Unha vez o usuario está autenticado, verá o seu nome na esquina superior dereita e, pulsando nel, aparecerá o menú da figura C.5.

C.3.2 Xestión dos datos do usuario

Desde o menú da figura C.5 pode acceder ás distintas partes da aplicación: perfil persoal, programas, grupos e páxina de explorar. Pulsando no apartado "O meu perfil" accédese á pantalla da figura C.6 na que se ven os datos persoais.

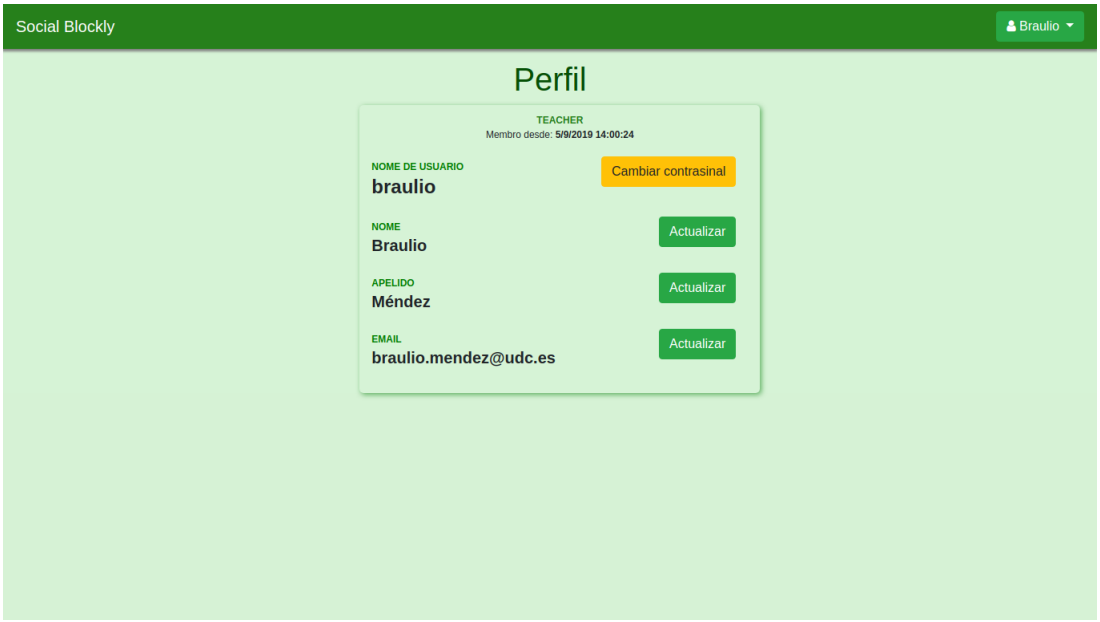


Figura C.6: Páxina do perfil persoal.

Neste apartado poden modificar todos os datos, excepto o *username*, como por exemplo o contrasinal, tal e como se aprecia na figura C.7.

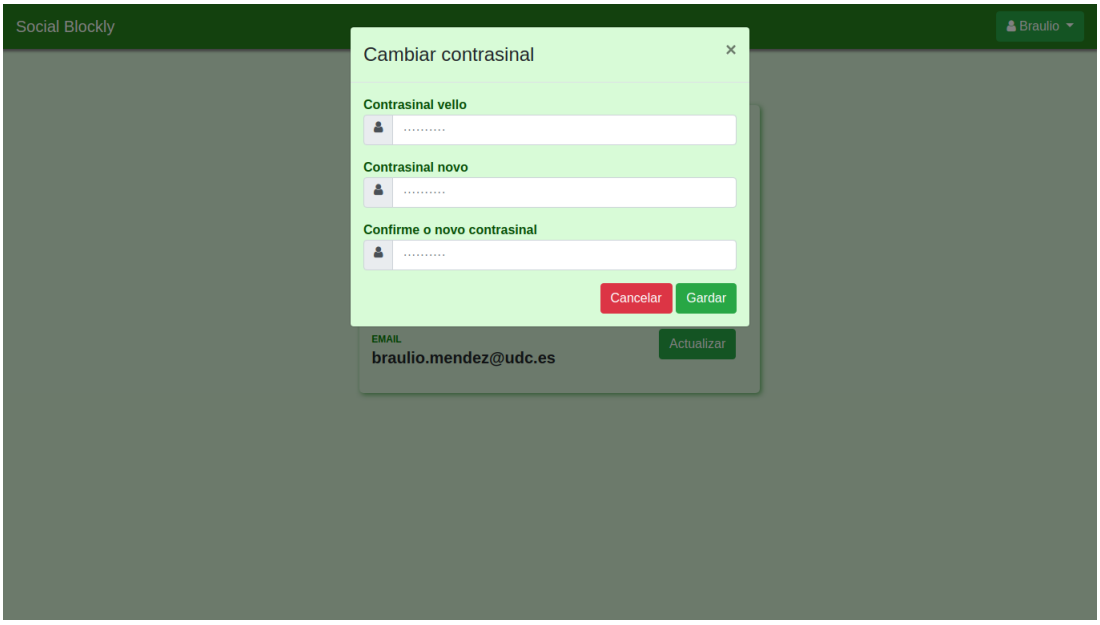


Figura C.7: Cambiar o contrasinal.

C.3.3 Xestión de programas

Figura C.8: Páxina dos meus programas.

Facendo click na sección "Os meus programas", éntrase na páxina da figura C.8. En cada programa, pódense editar os seus datos (figura C.9), eliminalo (figura C.10) ou xestionar os seus permisos, accedendo á páxina da figura C.11. No caso de querer que o programa sexa compartido, engadiranse usuarios a unha lista despois de procuralos por nome, apelido e email (C.12).

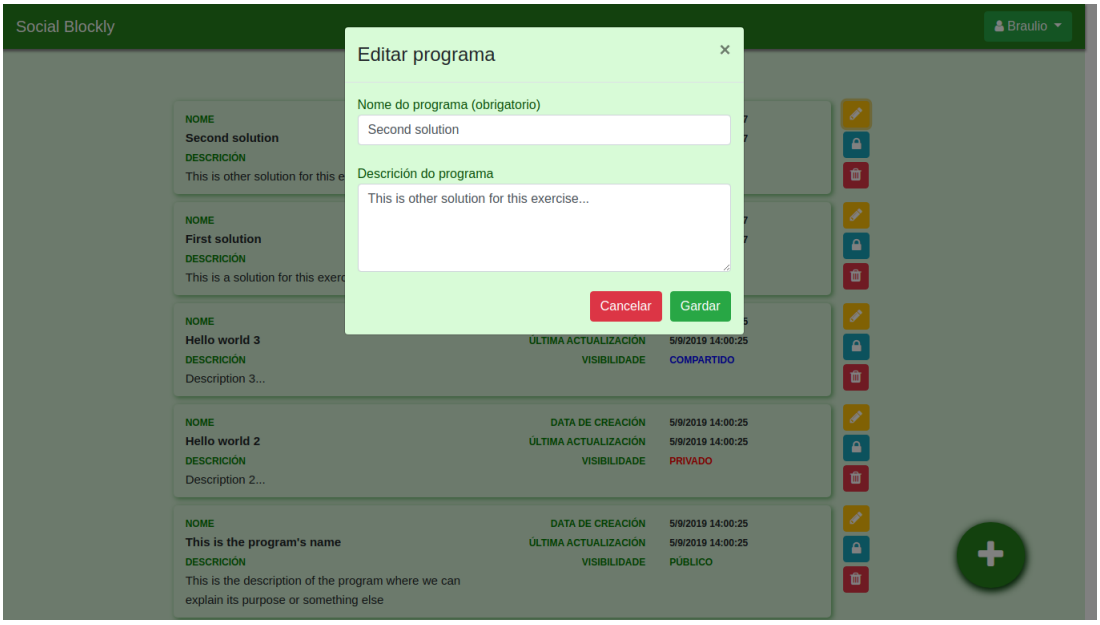


Figura C.9: Editar os datos dun programa.

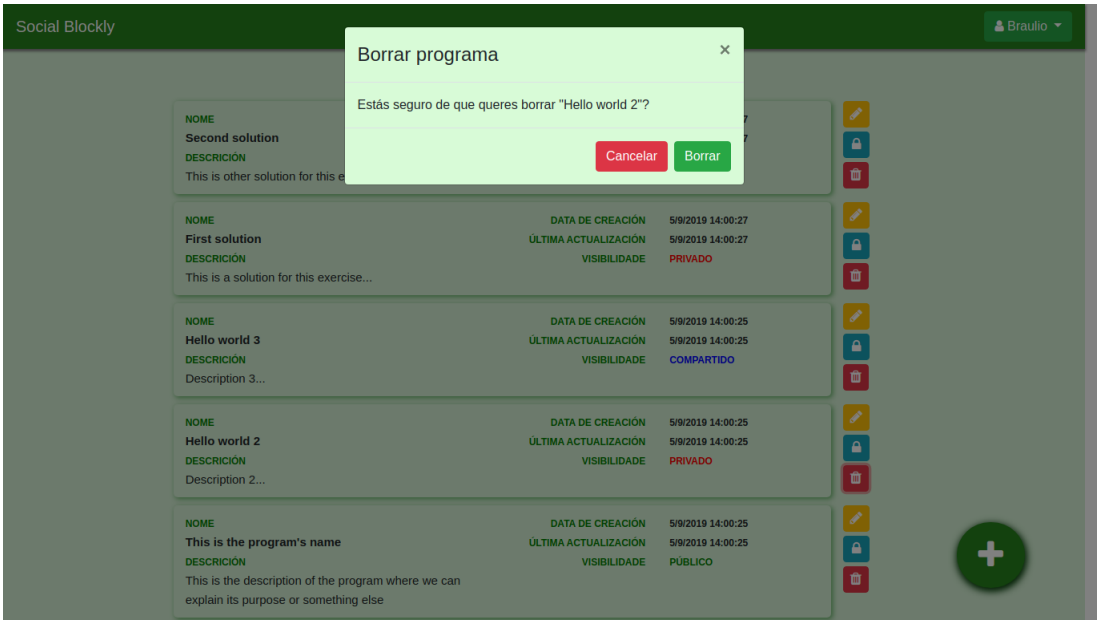


Figura C.10: Borrar un programa.

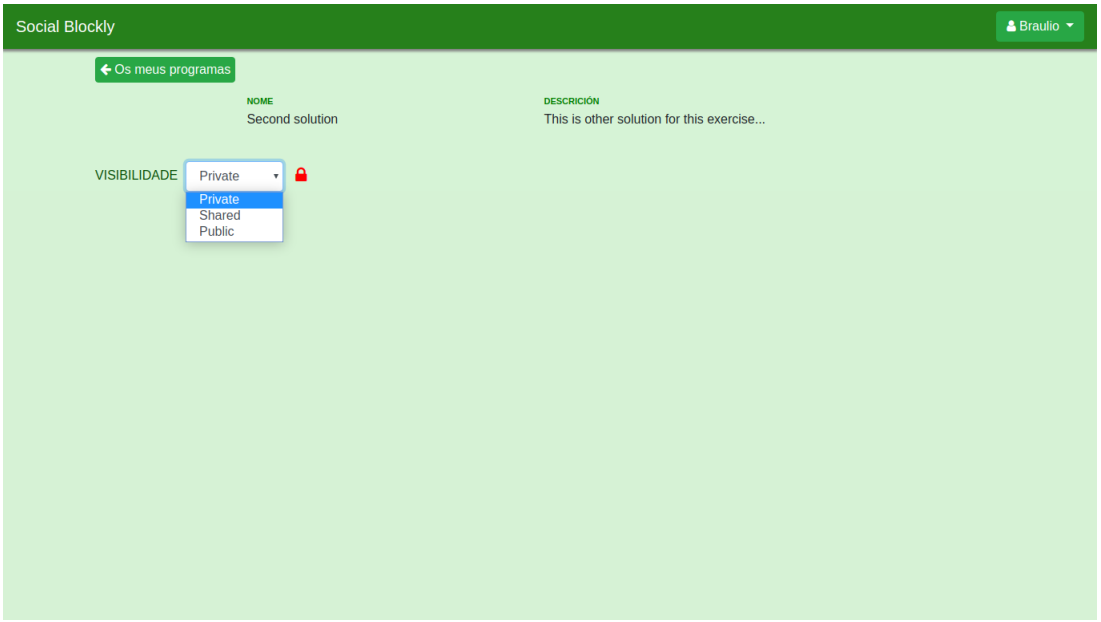


Figura C.11: Xestión dos permisos dun programa.

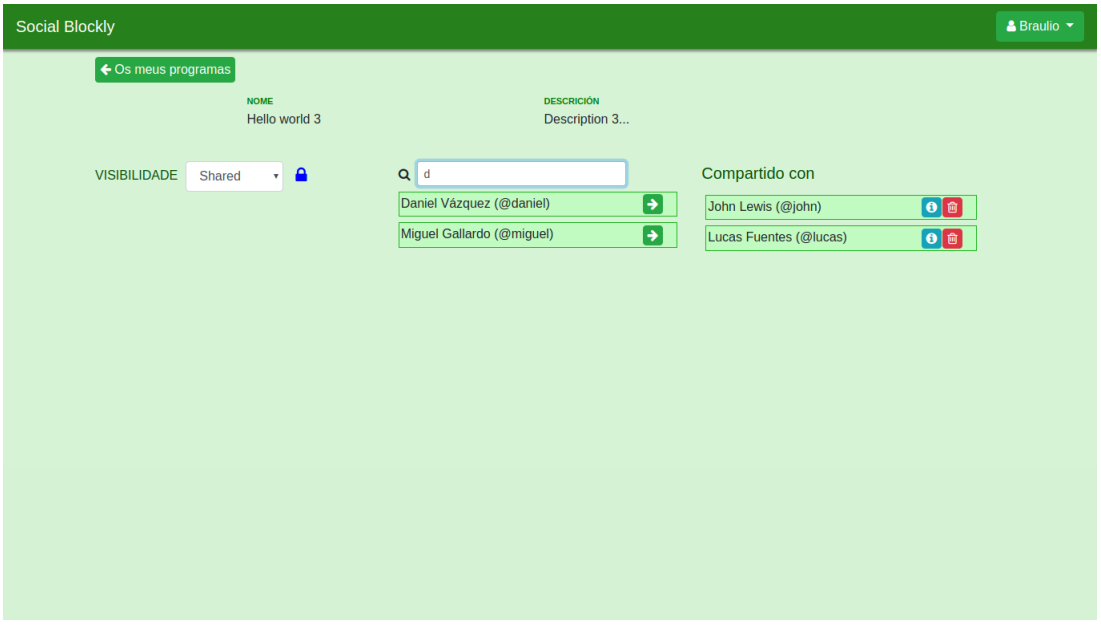


Figura C.12: Compartir un programa.

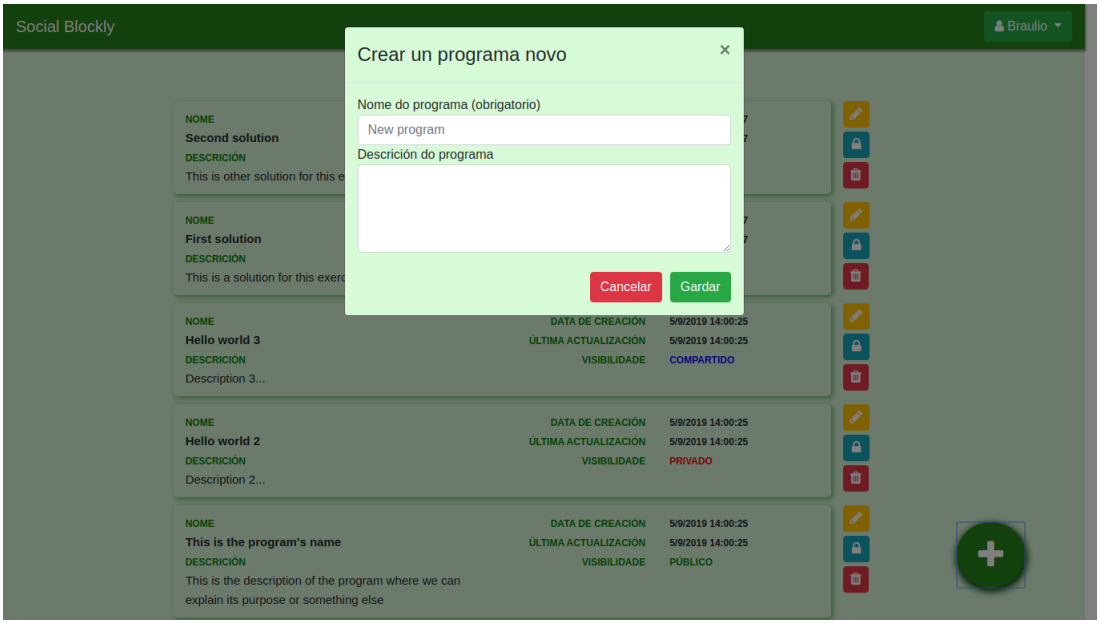


Figura C.13: Crear un programa.

Para crear un programa novo, desde a páxina dos meus programas (figura C.8), hai que pulsar no botón verde "+" aparecendo a ventá da figura C.13.

Ao acceder a un exercicio, móstrase a pantalla da figura C.14. Neste caso, o programa aínda non está gardado e por eso aparece un aviso en vermello xusto á dereita do botón de gardar. O botón á dereita desa mensaxe é para executar o programa. Os outros tres botóns, á esquerda do de gardar, son para *desfacer* un cambio, *refacer* un cambio e limpar o *workspace*, respectivamente de esquerda a dereita. Tamén se pode limpar a consola de saída pulsando no botón "Limpar saída".

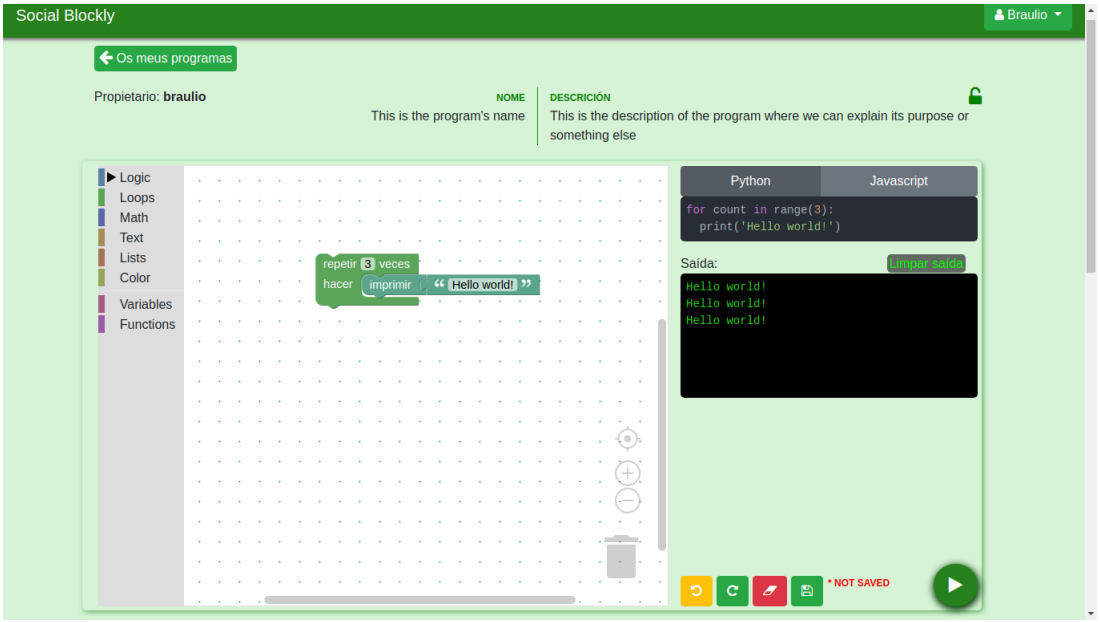


Figura C.14: Páxina dun programa.

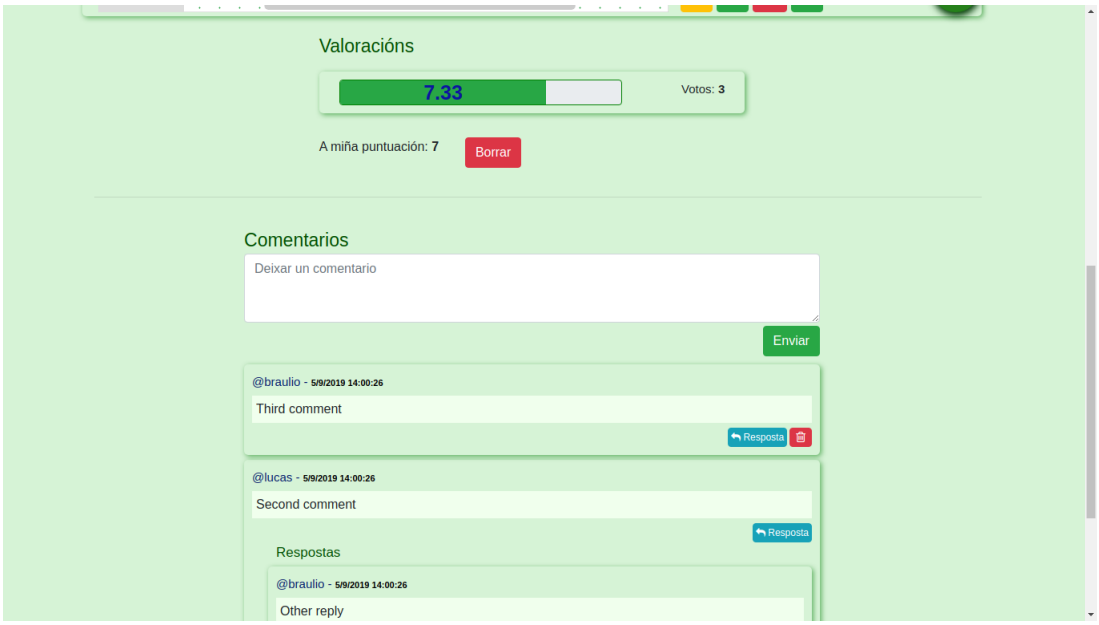


Figura C.15: Valoracións e comentarios dun programa.

Sobre calquera programa ao que se teña acceso, pódense realizar valoracións e comenta-

rios, como se ve na figura C.15.

C.3.4 Xestión de grupos

Pulsando no apartado de "Os meus grupos" do menú da figura C.5 accédese á pantalla da figura C.16 na que se mostran todos os grupos aos que pertence o usuario.

Figura C.16: Páxina dos grupos do usuario.

No caso de ser un *profesor*, o usuario pode crear un grupo pulsando no botón verde "+" e introducirá o seu nome nunha ventá coma a da figura C.17.

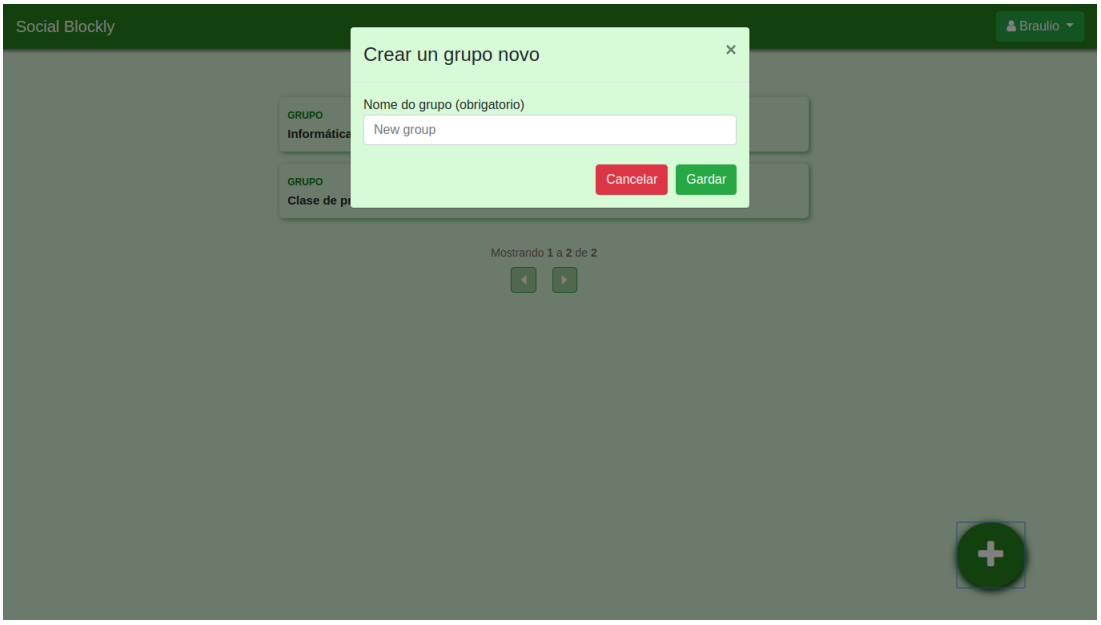


Figura C.17: Crear un grupo.

Ao acceder a un grupo vese unha páxina coma a da figura C.18, na que se aprecia a lista de exercicios, o chat no que poden falar todos os membros e a lista de membros.

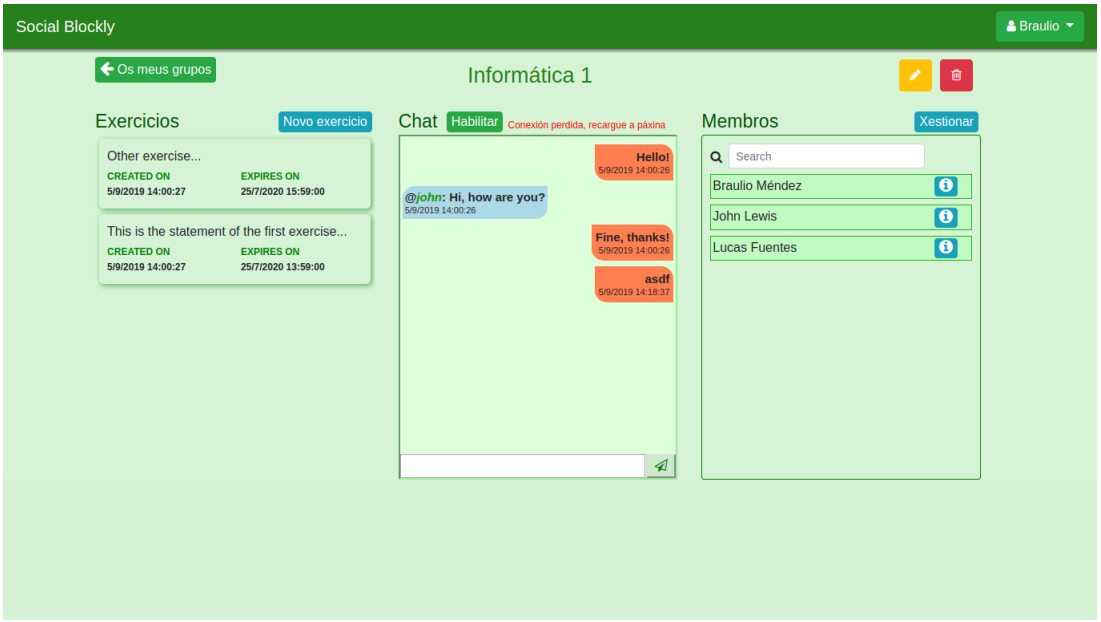


Figura C.18: Páxina dun grupo.

No caso de ser o *administrador* do grupo, o usuario pode editar ou eliminar o grupo (botóns á dereita do nome) e inhabilitar ou habilitar o chat grupo pulsando o botón que está enriba do chat. A mensaxe en vermello que aparece enriba do chat, só se mostra no caso de que se perda a conexión nalgún momento, xa que o chat actualízase automaticamente cando hai mensaxes novas. Se o usuario é o *administrador*, tamén pode acceder á páxina de xestión de membros (figura C.19), pulsando no botón "Xestionar". Nesa páxina, pódense procurar usuarios e, dos que aparecen na lista, pulsar na frecha á dereita do nome para engadilo (aparecerá unha modal de confirmación coma a da figura C.20).

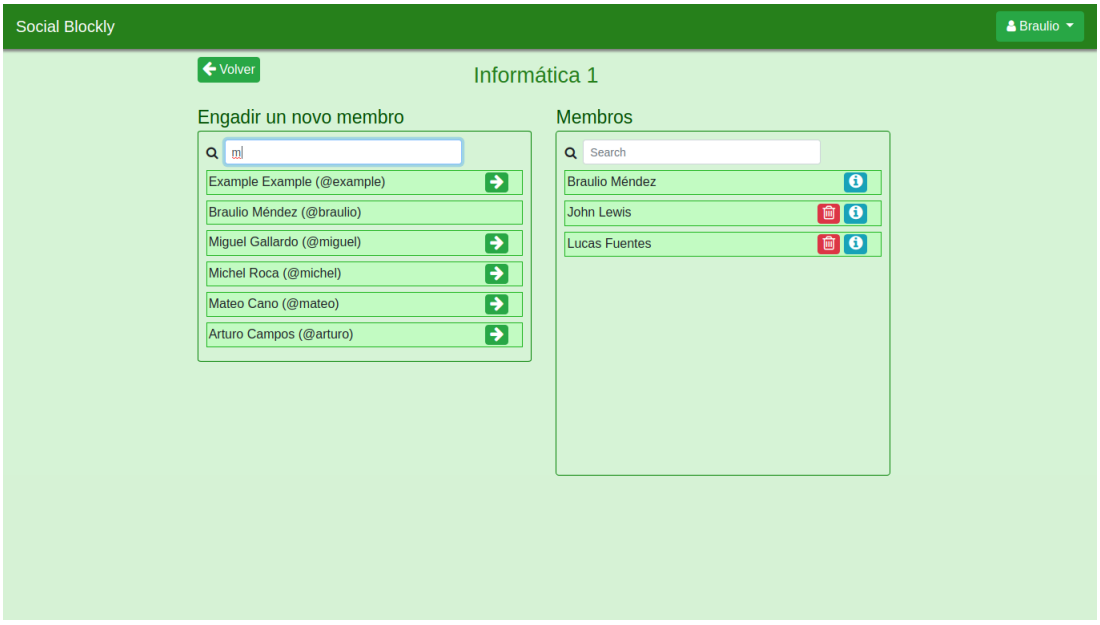


Figura C.19: Xestión de membros dun grupo.

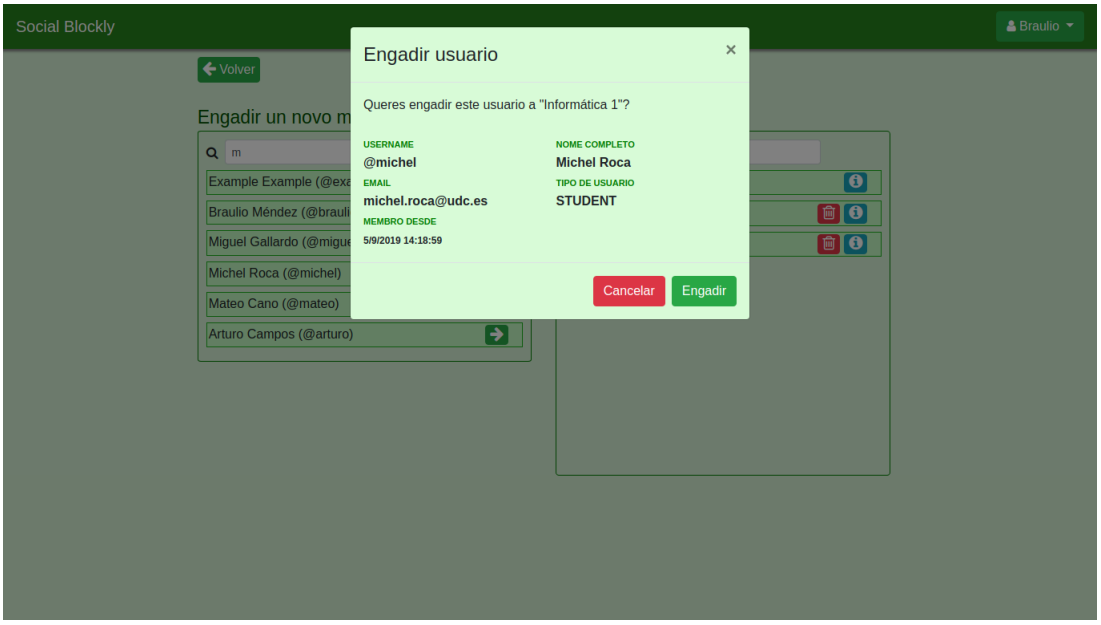


Figura C.20: Engadir un membro.

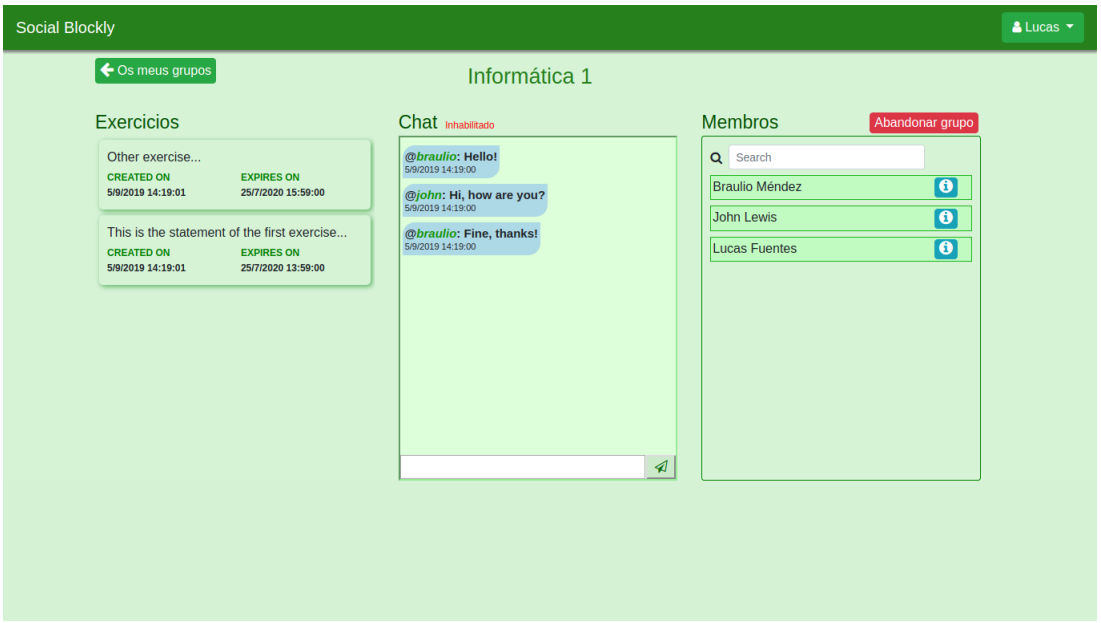


Figura C.21: Páxina dun grupo para un usuario non administrador.

No caso dun usuario, que non sexa o *administrador* do grupo, a pantalla que verá respecto

dun grupo ao que pertence será unha coma a da figura C.21. Nela, aparte de acceder a un exercio ou falar polo chat, poderá abandonar o grupo pulsando no botón que aparece enriba da lista de membros.

Desde a páxina dun grupo (C.18), un *profesor* tamén pode crear un exercicio como se ve na figura C.22, pulsando en "Novo exercicio".

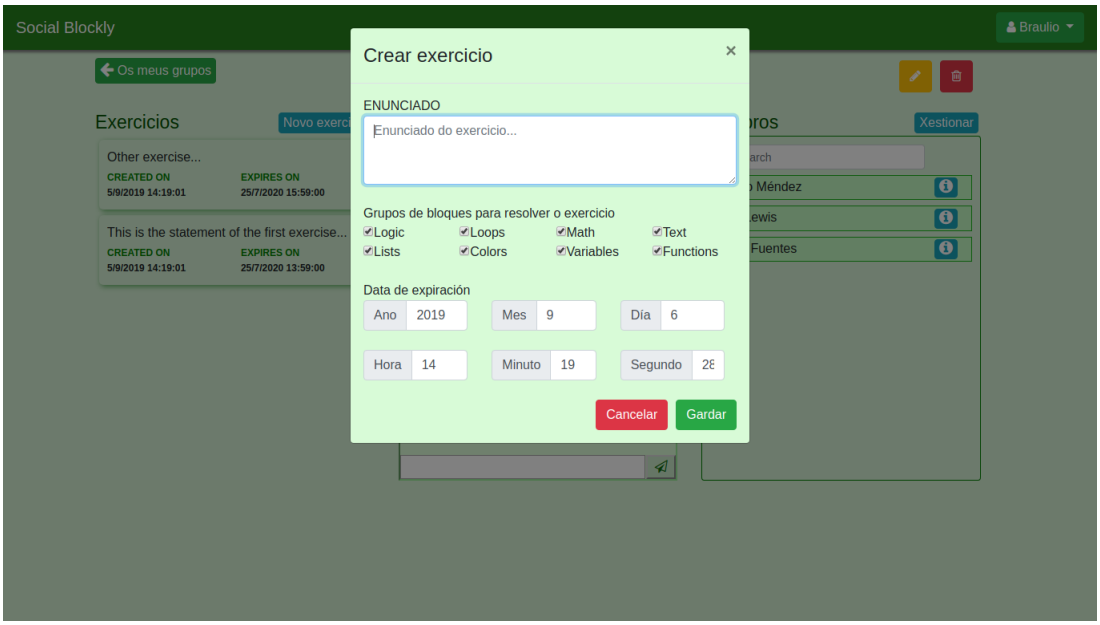


Figura C.22: Crear un exercicio.

Pulsando nun dos exercicios, desde a páxina do grupo (C.18), accédese a outra na que se mostra a información do exercicio e as solucións propostas, como se ve na figura C.23. Desde aí, o *administrador* do grupo tamén pode editar os seus datos ou eliminalo nos botóns situado á dereita.

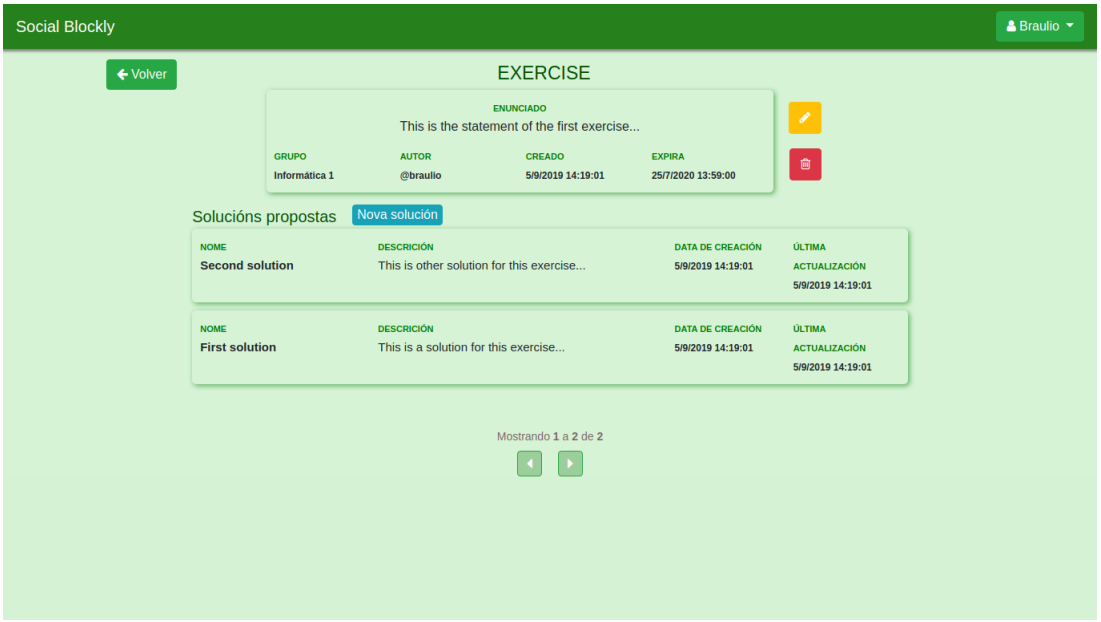


Figura C.23: Páxina dun exercicio.

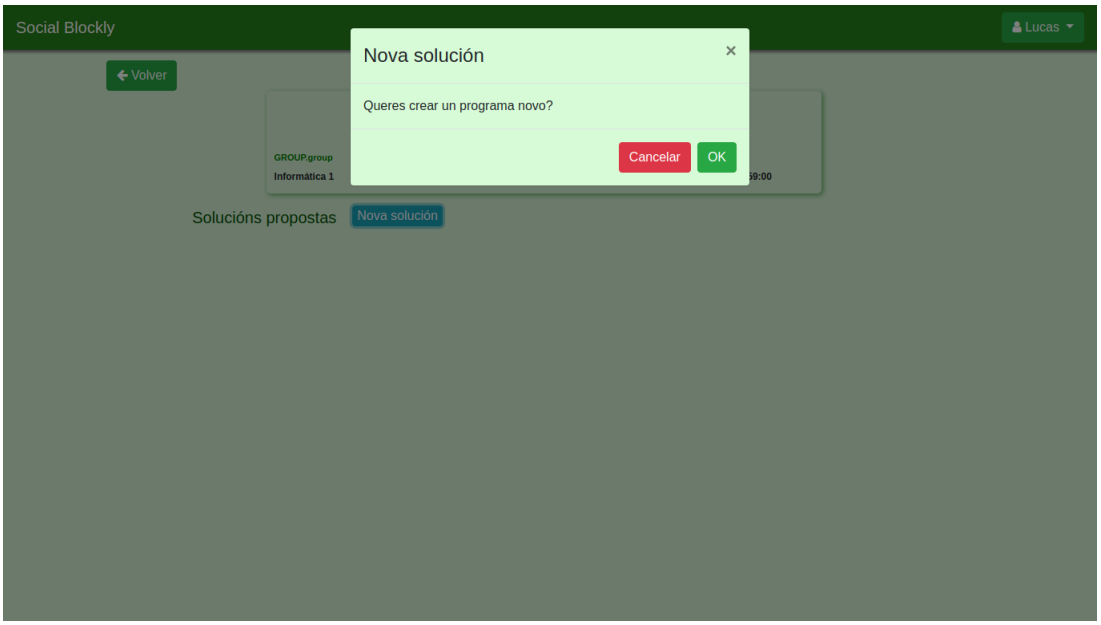


Figura C.24: Crear unha solución a un exercicio.

Para crear un novo programa que faga referencia a un exercicio, púlsase no botón Nova

solución e aparece unha ventá como a da figura C.24.

C.3.5 Explorar programas e usuarios

A última sección do menú da figura C.5 é a de Explorar. Pulsando aí, accédese á páxina da figura C.25. Nesa páxina aparecería a lista de programas compartidos por parte de calquera usuario, co usuario actual. Tamén unha barra de búsqueda para procurar programas doutros usuarios por título e descrición (só os que sexan públicos) e outra para buscar usuarios.

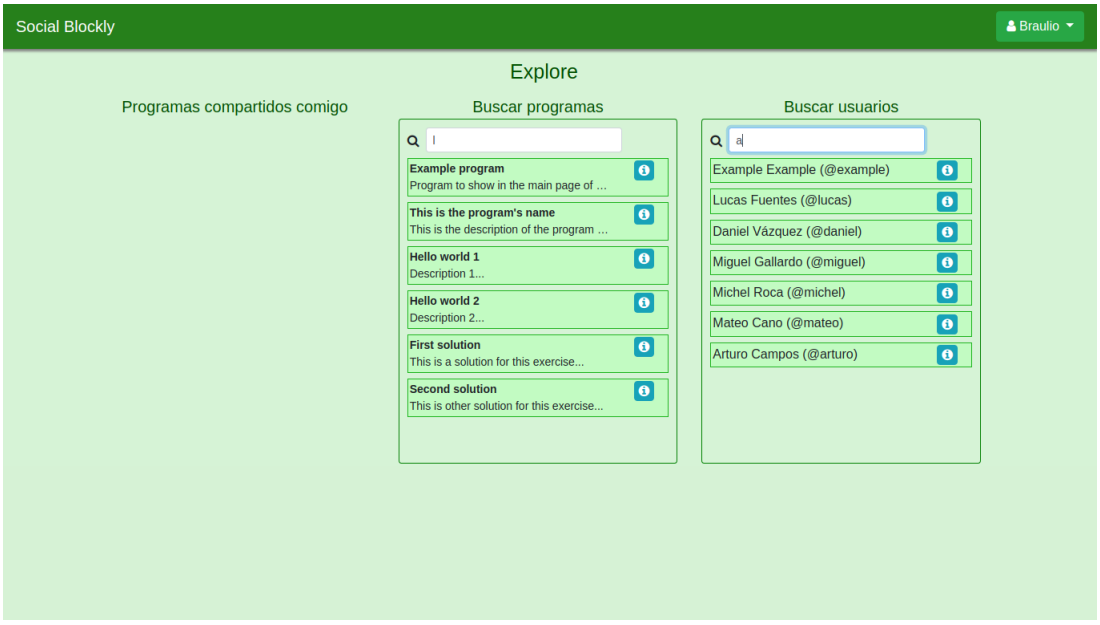


Figura C.25: Páxina de explorar.

No caso de querer ver a información dun usuario, accederase a unha pantalla coma a da figura C.26, na que se mostran os seus programas compartidos co usuario actual, os seus programa públicos e os grupos que ambos usuarios teñan en común.

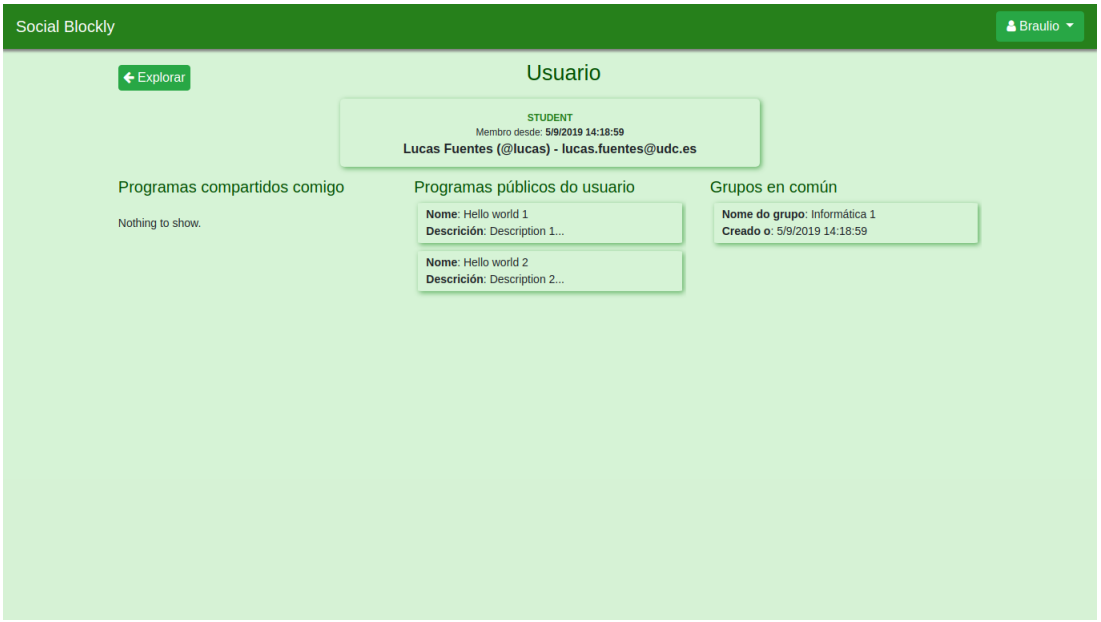


Figura C.26: Páxina do perfil doutro usuario.

Bibliografía

[1] Google. Introduction to blockly. [Online]. Available: https://developers.google.com/blockly/guides/overview

[2] Oracle. Introduction to java. [Online]. Available: https://docs.oracle.com/javase/7/docs/technotes/guides/language/

[3] MDN. Introduction to apache maven project. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[4] Typescript. [Online]. Available: https://www.typescriptlang.org/docs/home.html

[5] MDN. Introduction to html. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML

[6] ——. Introduction to css. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS

[7] Apache. Introduction to apache maven project. [Online]. Available: https://maven.apache.org/what-is-maven.html

[8] Pivotal. Spring boot. [Online]. Available: https://spring.io/projects/spring-boot

[9] ——. Spring. [Online]. Available: https://spring.io/

[10] Google. Getting started with angular. [Online]. Available: https://angular.io/start

[11] B. team. Introduction to bootstrap. [Online]. Available: https://getbootstrap.com/docs/4.3/getting-started/introduction/

[12] S. Cooper. ngx-toastr. [Online]. Available: https://www.npmjs.com/package/ngx-toastr

[13] Font awesome icons. [Online]. Available: https://fontawesome.com/v4.7.0/icons/

[14] Balsamiq. [Online]. Available: https://balsamiq.com/wireframes/

[15] Introduction to junit4. [Online]. Available: https://github.com/junit-team/junit4/wiki/getting-started

[16] Jacoco. [Online]. Available: https://www.eclemma.org/jacoco/

[17] Postman. [Online]. Available: https://www.getpostman.com/downloads/

[18] MDN. Http. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP

[19] What is jdbc. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSGU8G_12.1.0/com.ibm.jdbc_pg.doc/ids_jdbc_011.htm

[20] Eclipse. [Online]. Available: https://www.eclipse.org/downloads/

[21] Microsoft. Visual studio code. [Online]. Available: https://code.visualstudio.com/

[22] Git. [Online]. Available: https://git-scm.com/

[23] Angular cli. [Online]. Available: https://angular.io/cli

[24] Tomcat. [Online]. Available: http://tomcat.apache.org/

[25] Oracle. Introduction to mysql. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/introduction.html

[26] Scratct. [Online]. Available: https://scratch.mit.edu/

[27] Code.org. [Online]. Available: https://code.org

[28] Blockly games. [Online]. Available: https://blockly-games.appspot.com

[29] Micro bit. [Online]. Available: https://microbit.org

[30] Uml. [Online]. Available: https://www.uml.org/what-is-uml.htm

[31] Pagingandsortingrepository. [Online]. Available: https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html