Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Desenvolupament d'una aplicació web pel seguiment i avaluació de projectes de software en grups per docència

Document: Memòria

Alumne: Laura Puigmal Lladó

Tutor: Ignacio Martín Campos
Departament: Informàtica, Matemàtica Aplicada i Estadística
Àrea: Llenguatges i Sistemes Informàtics

Convocatòria (mes/any): **Setembre/2021**

ÍNDEX

1 INTRODUCCIÓ, MOTIVACIONS, PROPÒSIT I OBJECTIUS DEL PROJECTE

Dins la oferta d'assignatures del Grau en Enginyeria Informàtica de la Universitat de Girona s'imparteix al 2n semestre del 3er curs l'assignatura "Projecte de desenvolupament de software". Aquesta assignatura introdueix l'alumne a una eina de gestió de tasques i un sistema de control de versions mentre es desenvolupa un projecte de software en grup seguint una metodologia àgil. Aquest projecte, una aplicació Android que consumeix una API REST, és de temàtica lliure i s'expandeix en tot el semestre. En les activitats d'avaluació es valora principalment la qualitat del procés de desenvolupament del projecte, ja que l'objectiu principal de l'assignatura és ensenyar l'alumne a treballar seguint una metodologia de desenvolupament de software.

Dins l'aula s'ha utilitzat per a la gestió de tasques eines comercials: Jira [1] i YouTrack [2]. Tanmateix, aquestes eines no tenen funcionalitats integrades d'avaluació i d'aquí sorgeix la idea de fer una aplicació web per al control de projectes de software dins una aula amb metodologia àgil. Implementant una aplicació a mida es vol facilitar al professor el seguiment i avaluació de nombrosos alumnes de forma més lleugera i també oferir al alumne una experiència més senzilla, ja que les eines existents tenen moltes més funcionalitats que poden confondre.

A nivell personal, per al Projecte de Final de Grau tenia clar que volia desenvolupar una web en la qual s'hagués de fer anàlisi de requeriments, disseny i implementació. Se'm va proposar realitzar aquesta aplicació web i vaig trobar que encaixava les meves motivacions. Com a punt addicional, la part client del navegador web s'implementaria amb la llibreria React [3] que a nivell professional m'interessava ja que a l'empresa on treballo el podria adoptar en el futur.

El projecte parteix d'un punt de partida inicial: una API REST començada sobre l'entorn Spring Boot. La versió inicial de l'API REST de la qual parteix el projecte té l'esquelet per capes d'aplicació i un model inicial d'entitats. L'aportació d'aquest projecte és implementar l'API REST - fent les crides API, els serveis de la capa negoci i ampliant el model d'entitats – i implementar una Single Page Application que consumeix aquesta API REST.

Aquesta API REST forma part d'un projecte de codi obert anomenat TrackDev [4] iniciat pel professor Ignacio Martín Campos i allotjat dins l'organització Trackdevel a la plataforma GitHub [5].

2 ESTUDI DE VIABILITAT

El projecte desenvolupat, englobat dins del projecte anomenat TrackDev, és un projecte de codi obert. L'aplicació final tindrà moltes funcionalitats que no es poden implementar en un sol projecte de final de grau per la seva mida. L'objectiu és que es pugui continuar per part del professor o altres alumnes a posteriori.

Totes les llibreries i frameworks que s'utilitzaran són de codi obert i les hores requerides per implementar el projecte són hores aportades pels alumnes com a activitat acadèmica. Per tant a nivell econòmic el projecte és viable.

No hi ha cap dificultat tècnica per desenvolupar el projecte ja que es tracta d'una aplicació web de tipus de gestió. Les dificultats que es poden trobar són més encarades a definir els requeriments d'avaluació i seguiment dels alumnes i la usabilitat de l'aplicació, com també la organització del codi per assegurar que sigui de fàcil mantenir pels futurs contribuïdors.

3 METODOLOGIA

Per aquest projecte s'ha seguit un procés de desenvolupament de tipus agile, que bàsicament és iteratiu e incremental en el que els requeriments poden evolucionar entre iteracions. He escollit un procés iteratiu dividit en fases de funcionalitats per diversos motius.

Disposava de quatre mesos per a realitzar el projecte i tenia davant meu molts punts d'anàlisis i disseny: anàlisi de requeriments, disseny de classes i contracte d'API, disseny d'interfície, familiaritzar-me amb Java i Spring, aprendre a utilitzar React i organitzar una Single Page Application. El risc de dedicar molt més temps del necessari a una d'aquestes fases era molt gran per la qual vaig decantar-me per la metodologia que em permetés implementar parts de l'aplicació web quasi llestes, dividint el projecte en blocs de funcionalitats i fent a cada fase l'anàlisi de requeriments, disseny, codi i tests.

D'aquesta manera era més senzill conèixer la meva velocitat en realitzar cada activitat, trobar les dificultats amb les eines i si es donés el cas de desviar-me molt del pla inicial, simplificar alguna funcionalitat.

Els passos seguits són:

1. Analitzar a alt nivell els requeriments
2. Dividir el projecte en diferent conjunts de funcionalitats, i planejar-les en fases de 2 setmanes, iteracions.
3. Per a cada iteració:
    a. Fer anàlisis
    b. Fer disseny
    c. Implementar codi de l'API i SPA
    d. Testejar
4. Proves i ajustaments

4 PLANIFICACIÓ

4.1 ESTRATÈGIA

D'acord amb la metodologia escollida i descrita en l'apartat anterior, s'han planificat les següents tasques:

4.1.1 Anàlisi i disseny a alt nivell

Aquesta fase consisteix en un anàlisi a alt nivell dels requeriments, per ajudar a dividir el projecte en blocs de funcionalitats i començar-lo en la següent fase. D'aquesta fase apareixen les tasques definides a aquest apartat.

Recollida de requisits

Una primera recollida de requisits, molt semblant a l'apartat 6 d'aquesta memòria.

Familiarització amb el codi de l'API REST de punt de partida

Llegir el codi existent i fer el diagrama de model extret del codi.

4.1.2 Implementació i proves

Familiarització amb Spring, React i eines de treball

Conèixer l'entorn de Java, Java EE (ara Jakarta en la versió més nova), el framework Spring i Spring Boot. També instal·lar totes les eines de treball que s'utilitzarien i arrancar l'API REST per primer cop.

Gestió de cursos, grups i usuaris

Aquesta tasca engloba moltes funcionalitats: convidar usuaris al sistema o a cursos, gestió de cursos, grups d'alumnes i gestió de la sessió amb les accions d'inici de sessió i sortir-ne.

Gestió de backlogs i tasques

Creació de tasques i visualitzar el llistat en el backlog. Inclou també les accions típiques que es poden realitzar sobre les tasques: estimar, crear sub-tasques, canviar d'estat, assignar, ordenar i fer seguiment de hores de treball realitzades.

Creació d'iteracions i sprints

Configuració de les iteracions de part del professor i gestió dels sprints, permetent afegir i treure tasques.

Seguiment i avaluació d'alumnes

Aquest bloc de funcionalitats no estava clar com seria al principi del projecte. Una idea era començar per permetre posar comentaris a cada nivell d'objecte i canvi fet al backlog.

Detalls i funcionalitats extres

Bloc de funcionalitats no definit amb l'objectiu d'implementar funcionalitats necessàries no detectades en la primera recollida de requisits del sistema.

4.1.3 Implementació i resultats

Posada en marxa de l'aplicació web en un servidor per tal que estigui disponible al públic durant una setmana com a demo per al projecte.

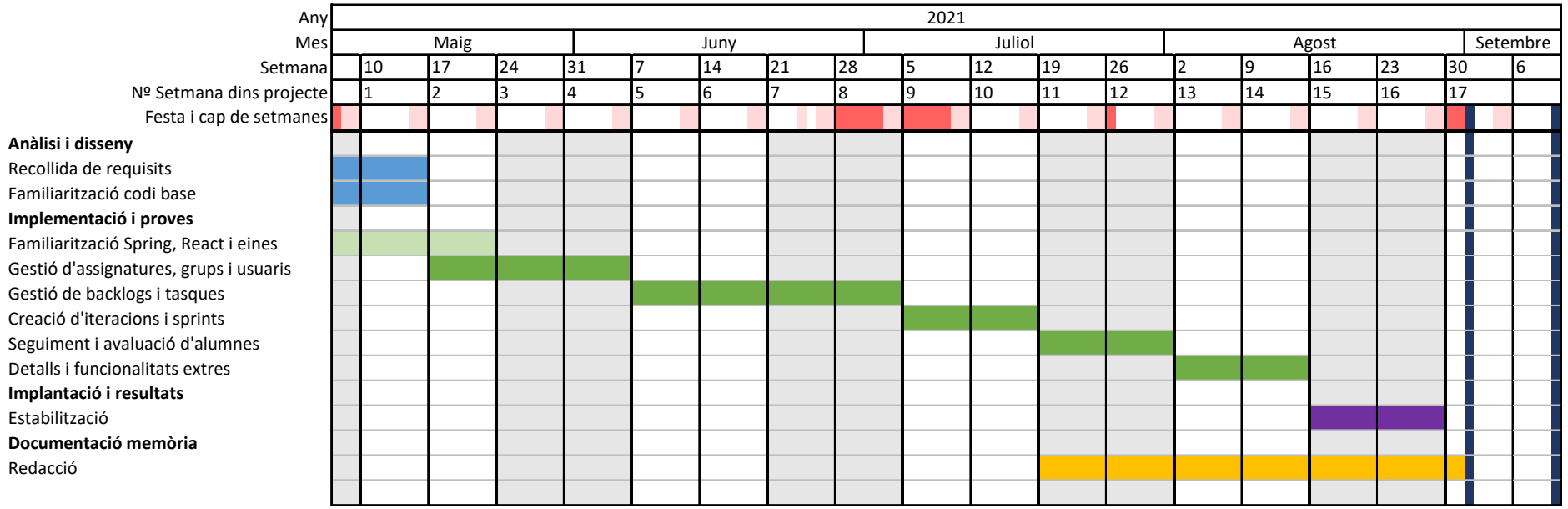
4.1.4 Documentació i memòria

Redacció

Redacció de la memòria.

4.2 PLANIFICACIÓ

Les fases i tasques descrites en l'apartat estratègia es van dividir al llarg dels quatre mesos del projecte, distribuint-les en iteracions de dues setmanes. En la següent figura podeu veure la planificació inicial que es va fer.



Il·lustració 1 Pla de treball inicial

4.3 COMENTARIS

Respecte al pla original de la figura anterior, hi ha hagut dos canvis importants. Per un costat, vaig fer unes setmanes de vacances entre en Juny i el Juliol que em va permetre avançar en el projecte treballant a jornada completa. L'altre punt és que degut a retards amb el pla inicial en la implementació i també que hi ha la possibilitat que aquest projecte el continuï un altre alumne dins un altre Projecte Final de Grau, es va acordar amb el professor deixar fora del projecte el bloc de "Seguiment i avaluació d'alumnes" i polir més el que és la gestió de tasques i iteracions.

També s'ha considerat al final que posar en marxa l'aplicació web a un servidor accessible al públic no tenia sentit ja que no hi hauria ara cap ús immediat d'aquesta.

5 MARC DE TREBALL I CONCEPTES PREVIS

5.1 METODOLOGIES DE DESENVOLUPAMENT

5.1.1 En cascada i iteratiu

Hi ha moltes metodologies de desenvolupament, tanmateix, es podria dir que hi ha dos tipus de processos: el model en cascada i els que segueixen un model iteratiu.

El model en cascada divideix el procés de desenvolupament basat en les diferents etapes o tipus d'activitat necessàries en tot projecte de software i les executa de forma seqüencial. Una etapa no comença fins que acaba l'anterior i no es torna a les etapes anteriors. Aquestes etapes són:

- Anàlisi de requeriments
- Disseny del software
- Implementació
- Proves
- Posar en marxa
- Manteniment

En canvi, els processos iteratius divideixen el desenvolupament en varis cicles que es repeteixen, en el quals es concentra en implementar un bloc de funcionalitats o fer prototips. Dins d'aquestes iteracions es poden fer vàries de les fases típiques de desenvolupament abans mencionades: anàlisi, disseny, implementació, proves i posada en marxa.

5.1.2 Agile

Les metodologies agile es basen en el manifesto Agile i engloba vàries metodologies del desenvolupament. Tenen en comú que són models iteratius e incrementals i que els requeriments de l'aplicació poden anar evolucionant al llarg de tot el desenvolupament. El manifesto declara una sèrie de principis en els quals destaquen la priorització de les persones i la interacció entre equips per sobre de les eines i documentació extensa.

Algunes metodologies sota del paraigües agile són: Extreme Programming (XP), Scrum i Kanban, entre altres.

5.1.3 Scrum

L'aplicació desenvolupada dins d'aquest projecte permetrà organitzar i fer control de tasques seguint una metologia agile inspirada amb Scrum. Scrum és un framework per gestionar projectes de forma agile.

Part del glossari de Scrum utilitzat dins el projecte és:

- **Sprint:** Un sprint és un període de temps fixe i curt en el qual l'equip de scrum completa una quantitat de feina acordada al inici d'aquest període. És una iteració.
- **Sprint Planning:** És un esdeveniment dins l'scrum en el qual es defineix la feina a dur a terme en el proper sprint.
- **Backlog:** El backlog és el document on es recull la llista de tasques que ha de fer l'equip. S'ordena per prioritat, amb el més important a fer properament a dalt de la llista i el menys prioritari a baix. La feina a fer, en format de tasques, poden ser funcionalitats noves en forma de user stories, bugs a arreglar o tasques tècniques de millora. Es

diferencia el backlog del producte i el backlog de l'sprint, en el qual hi ha les tasques de la iteració actual que l'equip ha agafat del backlog del producte.

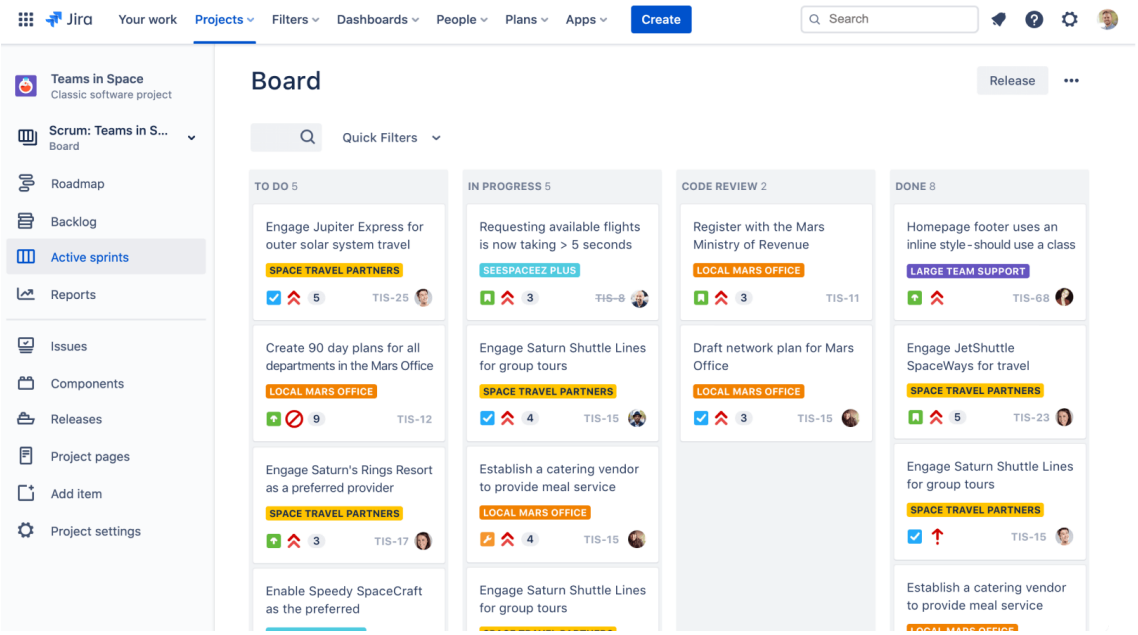
- **Estimació amb punts:** Les tasques s'estimen en punts en comptes d'hores de treball. Els punts donen una idea aproximada de l'esforç requerit per a realitzar la feina. Com que és una estimació aproximada, no s'utilitzen números exactes sinó que s'utilitzen els números d'una seqüència semblant a la de Fibonacci: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40 i 100.

5.1.4 Eines de control de projectes

Dues eines de control de projectes són Jira i YouTrack, ofertes per Atlassian i JetBrains respectivament. Aquestes eines, en format d'aplicació web, permeten crear projectes de software i organitzar-ne les tasques seguint una metodologia agile.

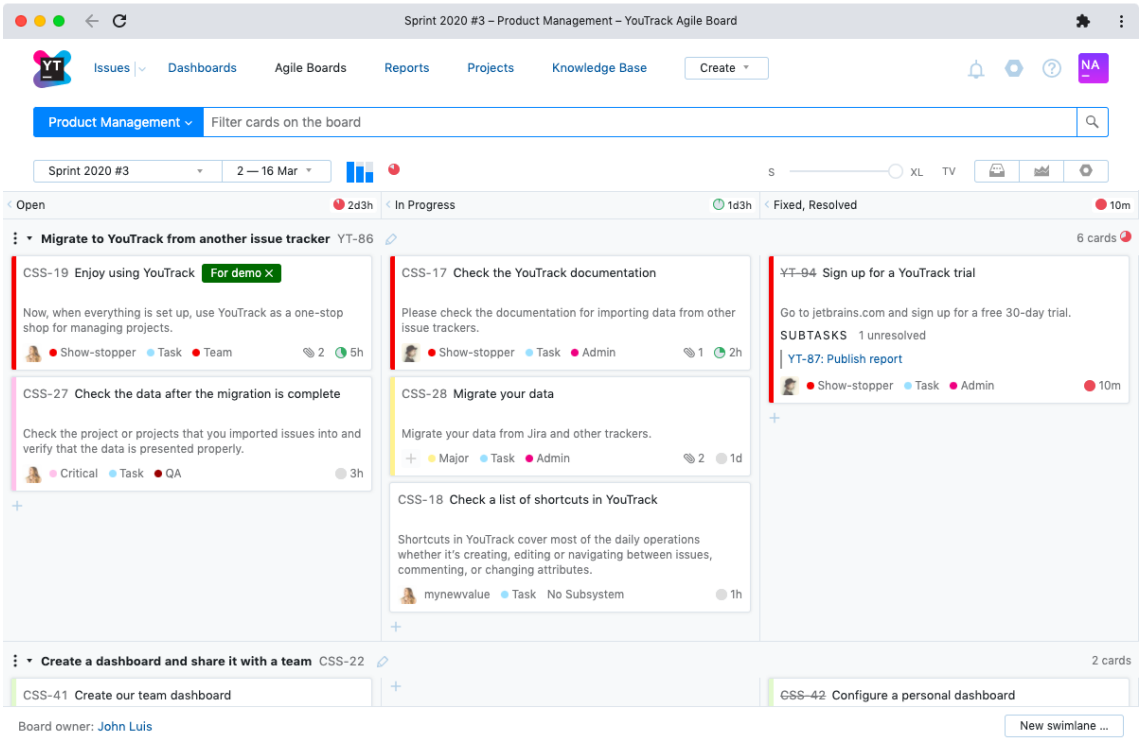
També ofereixen moltes més funcionalitats, com la configuració de panells per tenir una visió global d'un o varis projectes, integració automàtica amb sistemes de versió de control per tal de saber els canvis de codi relacionats amb cada tasca, definició d'èpiques i generació d'informes.

Ambdues eines s'han usat com a font d'inspiració per a implementar aquest projecte.



Il·lustració 2 Captura de pantalla que mostra un sprint actiu d'un projecte dins l'aplicació Jira [1]

[1] Imatge extreta de "Jira - Issue & Project Tracking Software", de Atlassian [1]



Il·lustració 3 Captura de pantalla que mostra un sprint actiu d'un projecte dins l'aplicació YouTrack[2]

5.2 Arquitectura Client-Servidor

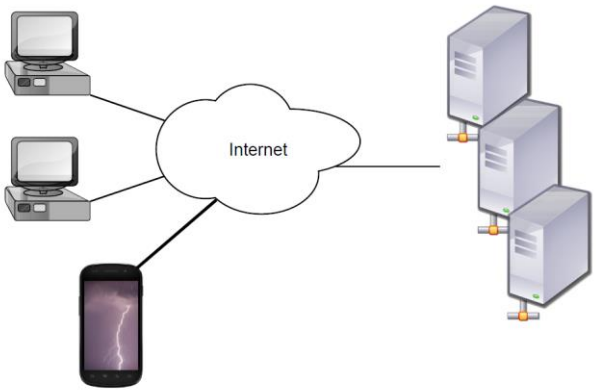
5.2.1 Serveis web

L'arquitectura client – servidor és una arquitectura d'aplicació distribuïda en el qual el client fa peticions per obtenir recursos o serveis a través d'una xarxa a un servidor on s'allotja l'aplicació que proveeix aquests recursos o serveis al client.

Quan el protocol utilitzat per a comunicar-se a través de la xarxa és el protocol HTTP es diu que l'aplicació del servidor és un servei web. API REST és una manera de dissenyar un servei web basat en la proposta d'arquitectura REST i utilitza el format XML o JSON per a transmetre les peticions.

Els clients poden ser navegadors web, programes d'ordinador, aplicacions de mòbil o altres serveis web.

[2] Imatge extreta de "YouTrack: The project management tool designed for agile teams", de JetBrains [2]



Il·lustració 4 Arquitectura Client-Servidor[3]

5.2.2 Aplicacions web

Una aplicació web és un programa que s'obté i s'utilitza a través d'un navegador web. Pot ser que sigui simplement un conjunt de pàgines web amb contingut estàtic o dinàmic, o pot oferir interactivitat i permetre gestionar tota mena de tasques, com ara fer compres per internet, consultar el correu electrònic, creació de documents i videotrucades.

A continuació s'explica el flux de documents i dades de vàries maneres d'implementar aplicacions web.

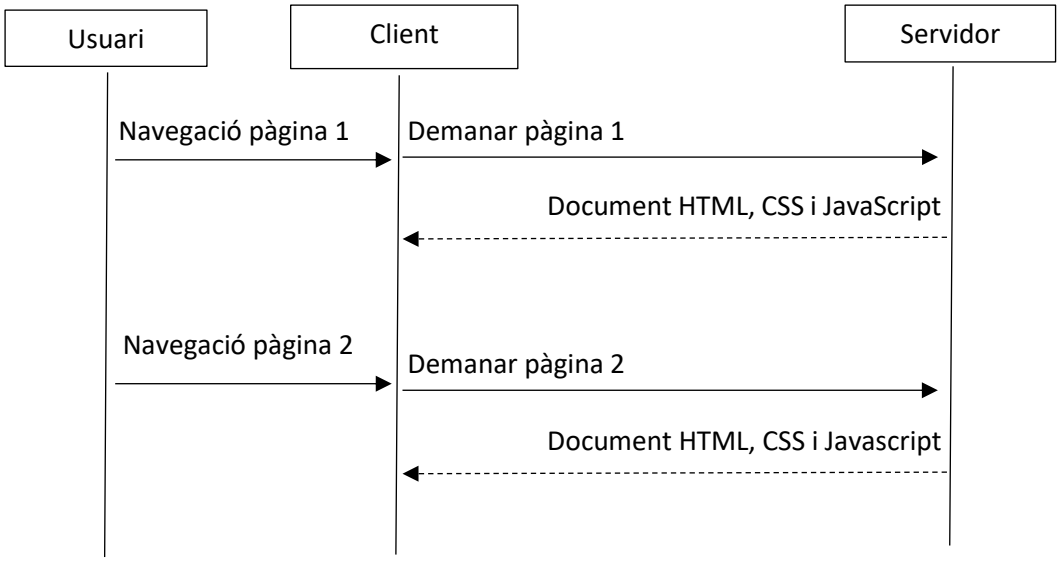
Pàgines estàtiques

En la forma més bàsica, les pàgines web són estàtiques amb el mateix contingut per a tots els usuaris i el servidor no genera l'HTML de la pàgina de forma dinàmica durant la petició feta pel client. El servidor guarda els documents HTML en un sistema de fitxers i només els serveix. Aquests documents es poden generar amb alguna eina que generi els documents HTML a partir de dades d'una base de dades o servei web, o podrien ser documents HTML generats a mà.

Aplicació web tradicional

El servidor genera el document HTML quan li arriba una petició del client, utilitzant una estructura HTML que conté espais reservats (placeholders) que se substitueixen amb dades d'una base de dades o de la sessió de l'usuari, entre altres fonts. Per escriure l'aplicació del servidor es poden fer servir molts llenguatges de programació, com ara PHP, Java, Python, Javascript o C#.

[3] Il·lustració extreta de "Multitier architectures: Spring", de I. Martin [43]

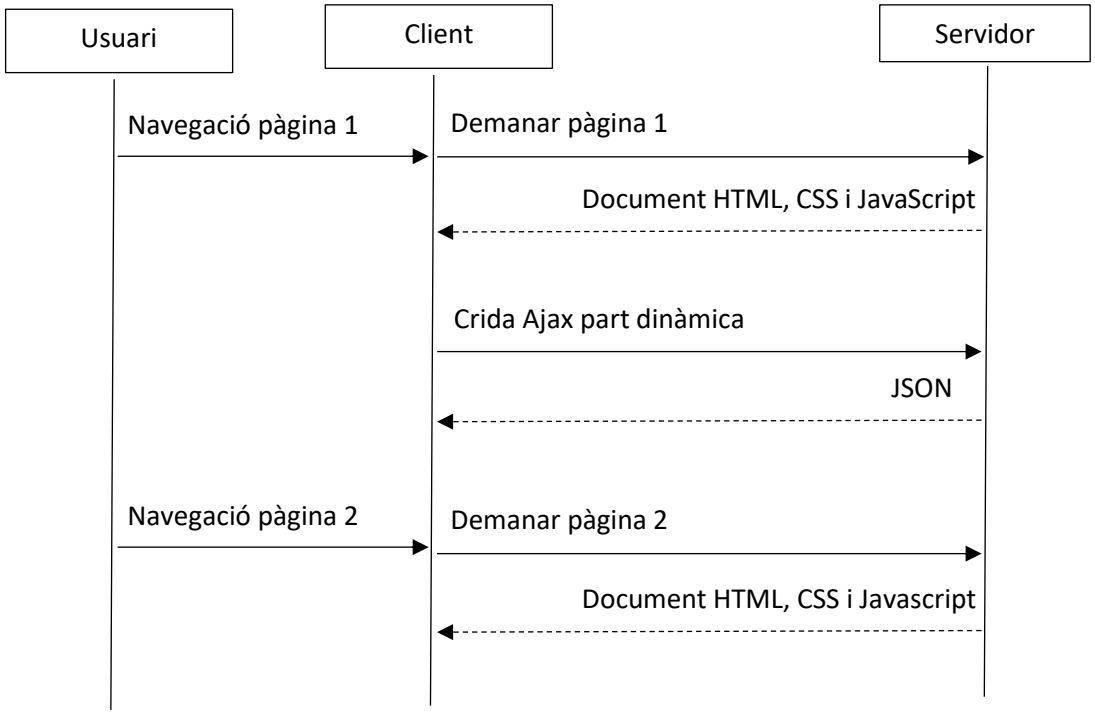


Il·lustració 5 Flux d'una aplicació web tradicional amb el renderitzat d'HTML a servidor

Aplicació web tradicional amb AJAX

Quan es va introduir als navegadors l'API `XMLHttpRequest` (XHR) es va popularitzar l'AJAX, el Asynchronous JavasScript and XML, la pràctica de portar interactivitat i dinamisme a les pàgines web usant com a tecnologia principal el XHR.

Un exemple seria una pàgina de cerca de productes d'un lloc web en el qual en la primera petició del client retorna el document HTML amb la capçalera, el menú principal de navegació, el peu de pàgina i els filtres de cerca. Quan l'usuari selecciona filtres, amb Javascript utilitzant l'API XHR es comunica amb el servidor per obtenir només les dades del resultat de la cerca i s'actualitza el Document Object Model (DOM) per mostrar a l'usuari els productes.

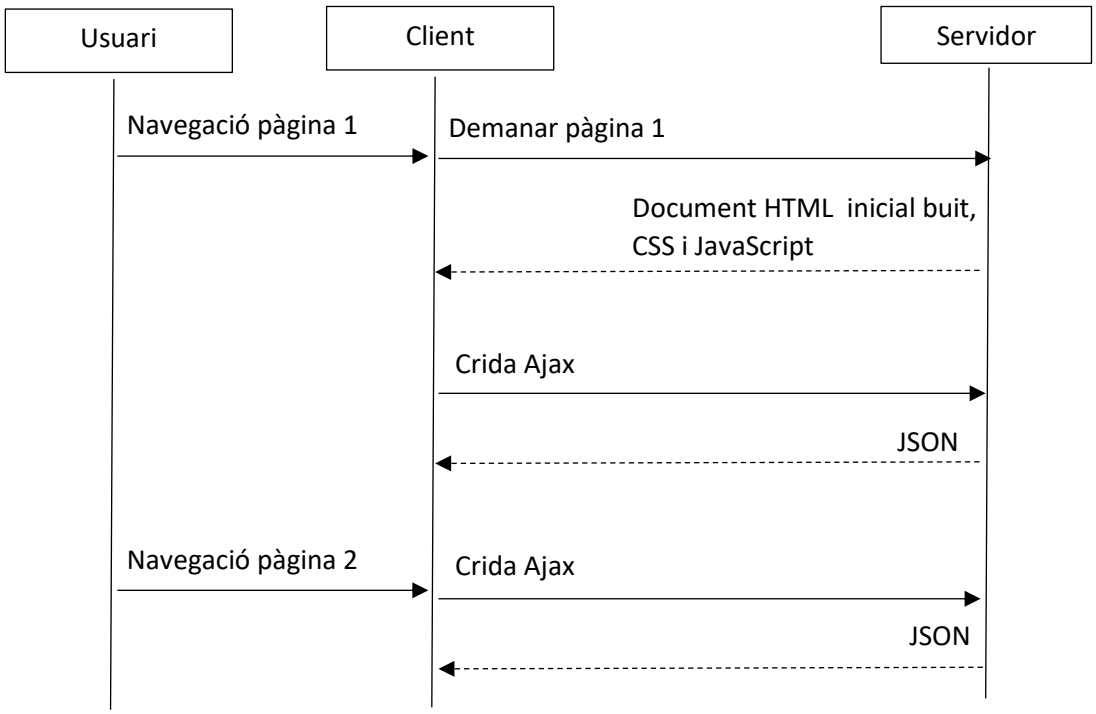


Il·lustració 6 Flux d'una aplicació web tradicional amb renderitzat a Servidor i parts dinàmiques amb crides AJAX

A partir d'aquí es va evolucionar cap a un model nou d'aplicació web, les Single Page Application.

Single Page Application

Les Single Page Application porten tota la generació del HTML de les diferents pàgines web al navegador. En una primera crida inicial del client al servidor es carrega un únic document HTML que és una carcassa buida a on viurà l'aplicació web implementada en Javascript. Totes les dades s'obtenen a través de crides AJAX. La navegació entre les diferents pàgines es fa amb Javascript de forma que no calgui una recàrrega de tota la pàgina web com sol passar amb el model d'aplicació web tradicional i només es canvia dinàmicament en el DOM els elements que canvien amb les noves dades. Per tal que per l'usuari hi hagi aparença d'aplicació web tradicional, es modifica la URL amb la nova ruta i es modifica la història de navegació per tal que elements com el botó d'anar enrere funcionin igual.



Il·lustració 7 Flux d'una aplicació Single Page Application amb tot el renderitzat HTML al client

6 REQUISITS DEL SISTEMA

En aquest apartat es descriuen els requisits funcionals i no funcionals que ha de complir l'aplicació per assolir els seus objectius. Aquests requisits són els recollits a l'inici del projecte i els més essencials.

6.1 FUNCIONALS

Els usuaris de l'aplicació poden tenir tres rols diferents: **Administrador**, **Professor** i/o **Alumne**. Per blocs de funcionalitats, aquests són els requisits funcionals que ha de permetre el sistema:

6.1.1 Gestió d'assignatures, usuaris i grups

Els administradors poden:

- Gestionar els professors. Donar d'alta.

Els professors poden:

- Crear assignatures i gestionar-les.
- Gestionar els alumnes d'una assignatura. Convidar i donar de baixa.
- Gestionar els grups de projecte. Crear i eliminar. Afegir-hi o treure-hi alumnes.

Els alumnes es poden registrar utilitzant les invitacions dels professors.

6.1.2 Gestió de backlogs, iterations i tasques

Els professors poden:

- Visualitzar les accions efectuades sobre una tasca pels alumnes
- Visualitzar la història dels sprints del backlog i veure l'evolució del progrés del projecte.
- També podrien fer les mateixes accions que els alumnes, en cas de necessitat.

Els alumnes poden:

- Crear tasques al backlog del seu grup.
- Crear sprints nous, obrir-los i tancar-los.
- Entrar o treure tasques a un sprint.
- Treballar amb les tasques: estimar-les, crear sub-tasques, canviar el seu estat i assignar-les.

El sistema ha d'enregistrar totes les accions efectuades sobre el backlog.

6.2 REQUISITS NO FUNCIONALS

L'aplicació web requereix iniciar sessió per poder efectuar qualsevol acció. Les úniques funcionalitats permeses anònimament és l'inici de sessió i el registre.

Pel que fa als recursos – cursos, grups i backlogs – hi ha els següents requeriments:

- Només el professor encarregat de l'assignatura pot visualitzar i gestionar els grups, alumnes i backlogs dins d'aquesta.
- L'alumne enregistrat a una assignatura només pot visualitzar el seu propi grup i gestionar els backlogs dins d'aquest grup. No pot veure per exemple quins altres alumnes hi ha dins l'assignatura i altres grups.

7 ESTUDIS I DECISIONS

En aquest apartat es vol donar a conèixer les llibreries principals utilitzades per implementar el projecte i el programari utilitzat durant el desenvolupament d'aquest. Com que el projecte parteix d'una API REST de partida, també s'explicarà l'estat d'aquesta al començar el projecte.

7.1 LLIBRERIES I FRAMEWORKS

7.1.1 Servidor: REST API

L'API REST de punt de partida està implementada amb el llenguatge Java sobre l'entorn Spring Boot.

Java

Java és un llenguatge de programació i una plataforma.

El llenguatge de programació Java és un llenguatge d'alt nivell i Orientat a Objectes que està pensat per ser independent de la plataforma, sigui Windows o Linux.

Les plataformes de Java consisteixen en una Màquina Virtual de Java (*JVM*) i una Interfície de Programació d'Aplicacions (*API*). Dins d'aquest projecte s'ha treballat sobre les següents plataformes:

- Java Standard Edition (Java SE). L'API conté els objectes i tipus més bàsics de Java, que bàsicament són part del cor del llenguatge Java.
- Java Enterprise Edition (Java EE). Estén Java SE i proveeix APIs per a desenvolupar aplicacions de xarxa multicapa. Algunes són:
    - Servlets
    - JAX-RS RESTful web service
    - Enterprise Java Beans
    - Java Persistance API (JPA)

Actualment Java EE ha evolucionat cap a Jakarte EE en les últimes versions, passant el desenvolupament d'Oracle a l'Eclipse Foundation.

Spring

Spring [6] és un framework de Java. El formen varis mòduls, dels quals s'ha utilitzat:

- **Framework Spring Web MVC**, implementat a sobre de l'API Servlet de Java EE, que permet construir una API REST.
- **Spring Boot**, que facilita la configuració d'una aplicació web amb unes llibreries de tercer configurades.
- **Sprint Data**, que conté un conjunt d'interfícies i objectes per obtenir entitats de la base de dades per sobre de Java Persistance API (JPA).

Hibernate

Hibernate [7] principalment és conegut per implementar un framework d'Object/Relational Mapping (ORM). Addicionalment a la seva pròpia API, també implementa la Java Persistance API (JPA).

MySQL

Personalment he utilitzat una base de dades MySQL per a persistència de dades, però l'API original persistia les dades a una base de dades MariaDb. Amb una configuració d'Hibernate es pot definir la base de dades relacional concreta i per sota s'aplicaran els canvis adients per comunicar-se amb la base de dades.

Jackson

Jackson [8] és una llibreria per serialitzar i deserialitzar JSON i es fa servir per a construir el cos de la resposta de l'API REST.

7.1.2 Client: SPA

React

React es defineix com una llibreria Javascript per a construir interfícies d'usuaris. Els seus principals punts són:

- **Declaració de la interfície d'usuari.** Amb React la filosofia és que es defineix com ha de ser la interfície i llavors amb aquesta descripció de la interfície en forma de components React actualitza els elements reals del document de la pàgina que han canviat respecte l'estat anterior.
- **Organització en components** per dividir una interfície d'usuari en parts petites, cadascuna amb el seu propi estat.
- **No defineix la resta de l'arquitectura.** Per exemple es pot utilitzar React per només una part petita d'una pàgina web, implementar una Single Page Application completa o utilitzar-se per renderitzar HTML a un servidor amb Node JS.

L'últim punt en referència a l'arquitectura és el que diferencia React amb altres llibreries com Angular, que són tot una plataforma de desenvolupament. Per exemple React no ofereix mecanismes per tractar diferents rutes de la web o llibreries per comunicar-se amb el servidor.

Aquesta llibreria és mantinguda per Facebook i es fa servir en les seves aplicacions.

Create React App

Create React App [9] és un conjunt de tasques i agrupament de dependències per a desenvolupar una aplicació web amb React al navegador. Vaig optar per fer servir aquesta llibreria ja que altrament hauria de configurar a mà les eines per compilar i ajuntar els fitxers Javascript de codi font, Babel [10] i Webpack [11].

React-Router

React Router [12] és una llibreria que ofereix components per definir rutes, de tal manera que es pot simular que l'aplicació en realitat està formada per vàries pàgines i l'usuari navega entre elles amb enllaços.

Bootstrap

Bootstrap [13] és un joc d'eines de codi obert per dissenyar webs responsives. Ofereix un conjunt de components d'interfície bàsics com botons, pestanyes o carrusels; i també un conjunt d'utilitats com el sistema de columnes per a distribuir elements. Bootstrap està implementat amb Sass – un llenguatge d'estils que es compila a CSS - i Javascript.

React-Bootstrap

Per facilitar la integració dins l'aplicació React vaig escollir afegir una llibreria que oferís els components de Bootstrap com a components de React. N'hi havia dos disponibles: React-Bootstrap [14] i Reactstrap [15].

Bootstrap va llançar la versió 5 de la llibreria el Maig d'aquest any 2021 i cap de les dues llibreries implementava aquesta versió quan les vaig analitzar al Juny. Vaig escollir React-Bootstrap en comptes de Reactstrap perquè semblava que React-Bootstrap tenia el desenvolupament de la migració cap a Bootstrap 5 més avançada amb una release beta. Tot i això, he utilitzat la versió de Bootstrap 4 per poder fer servir la versió estable de React-bootstrap.

React-beautiful-dnd

React-beautiful-dnd [16] és una llibreria que proveeix components de React per implementar llistes ordenables i accessibles amb la funció d'arrossegar i deixar anar (drag and drop).

Aquesta llibreria encaixava molt bé amb els requeriments que tenia de poder ordenar elements d'una llista i moure elements d'una llista a un altre. El fet de veure que tenia suport per gestos tàctils en dispositius mòbil i suport accessible a ordenar només amb el teclat va ser un punt addicional afegit per escollir-la.

7.2 EINES DE DESENVOLUPAMENT

7.2.1 Desenvolupament de codi

IntelliJ IDEA Community Edition

IntelliJ IDEA [17] és un IDE per Java ofert per JetBrains. Per al desenvolupament de l'API REST d'aquest projecte he utilitzat la versió gratuïta IntelliJ IDEA Community Edition.

Visual Studio Code

Visual Studio Code [18] és un editor per escriure codi amb funcionalitats addicionals com ara terminal integrada, IntelliSense senzill i accions de control de versions integrades. L'he utilitzat per desenvolupar la SPA en Javascript i React. L'he escollit perquè ja la utilitzo a diari, però és molt semblant a altres editors com Atom [19].

7.2.2 Altres

MySQL Workbench CE

MySQL Workbench Community Edition [20] ofereix un conjunt d'interfícies gràfiques per treballar més fàcilment amb bases de dades MySQL. L'he utilitzat per a escriure i executar consultes a la base de dades i també per a generar automàticament diagrames d'entitat relació a partir de les taules existents generades per Hibernate.

Git

S'ha utilitzat Git com a sistema de versió de control durant tot el projecte. El projecte està dividit en dos repositoris allotjats a GitHub.

GitHub Desktop

GitHub Desktop [21] l'he utilitzat bàsicament per pujar el codi cap a GitHub i també per visualitzar la història més fàcilment.

Postman

Postman [22] és una aplicació per ajudar al desenvolupament d'APIs i l'he usada per a testejar les crides d'API directament sense l'aplicació web real i també per guardar-les en carpetes.

Diagrams.net, Inkscape i Word

Per a fer els diagrames UML i les figures de la memòria he utilitzat vàries eines: Diagrams.net [23], Inkscape [24] i Microsoft Word.

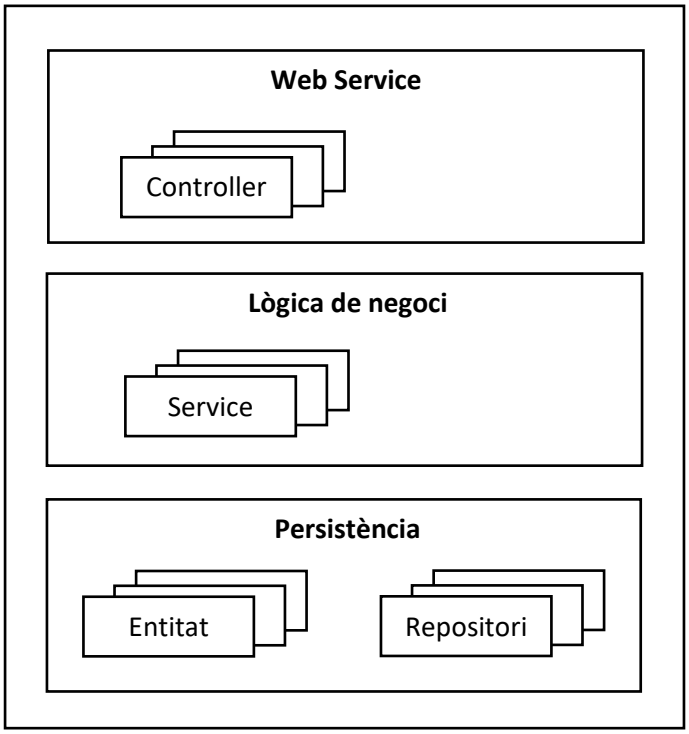
7.3 PUNT DE PARTIDA

Com que el projecte es basa en una aplicació de partida, seguidament procedeixo a explicar l'estat actual d'aquesta.

7.3.1 Estructura de l'API

L'API és una aplicació servidor organitzada en capes, cadascuna amb seva responsabilitat, que es correspondrien amb el patró Model-View-Presenter (MVP):

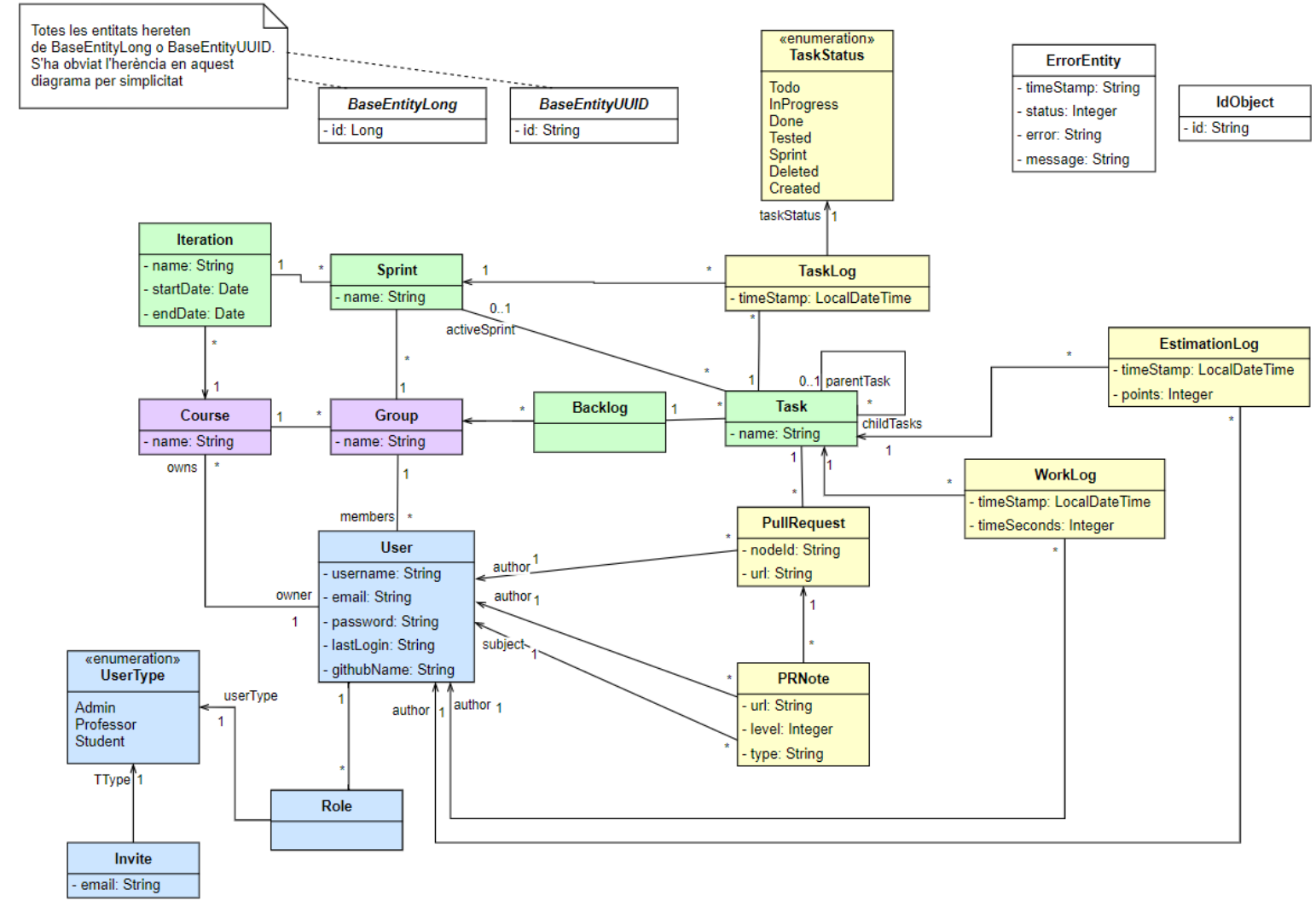
- **Web service**: La seva responsabilitat és processar les peticions HTTP i fer validacions simples. Formada en gran part per classes anomenades Controllers. Correspondria a la View.
- **Capa de negocis**: La seva responsabilitat és implementar la lògica de l'aplicació. Formada principalment per classes anomenades Services. Correspondria al Presenter.
- **Persistència**: Conté les Entitats i els Repositoris, responsables de la persistència i obtenció d'entitats de la base de dades. Les entitats són les classe que modelen i representen la realitat del domini de l'aplicació. Aquesta capa correspondria al Model.



Il·lustració 8 Esquema de les capes de l'API REST i les seves classes

7.3.2 Diagrama de classes

Una de les primers activitats que vaig fer va ser dibuixar un diagrama amb les classes de les entitats per visualitzar el model entitat-relació implementat en el aquell moment.



Il·lustració 9 Diagrama de classes de les entitats sense mètodes

7.3.3 URIs existents

A continuació es llisten les crides existents de l'API REST:

Mètode	Camí	Descripció
POST	/auth/login	Inicia sessió i crea un token JWT nou
POST	/auth/logout	Tancar sessió – expira les cookies
GET	/auth/self	Obtenir perfil d'usuari de la sessió
GET	/courses	Cercar cursos amb paràmetre `search`
GET	/tasks	Cercar tasques amb paràmetre `search`
POST	/hooks/github/pr	En progrés, la idea és guardar cada pull request fet a GitHub dels repositoris dels grups de treball automàticament
POST	/hooks/github/issue_comment	En progrés, la idea és guardar els comentaris d'un tiquet de GitHub

7.3.4 Serveis i repositoris

Per gairebé totes les entitats, hi ha el seu corresponent servei i la interfície d'accés a la base de dades utilitzant els repositoris base de Spring. Aquests components estan pràcticament buits i per implementar en la seva totalitat, exceptuant el codi necessari per les crides d'API definides al apartat anterior.

7.3.5 Utilitats per cercar de forma genèrica

L'API conté un paquet anomenat `query` amb classes que transformen una cerca escrita en un text pla en una consulta utilitzable en els repositoris utilitzant l'API de Specification de Spring Data, que per sota utilitza l'API de Criteria de JPA. Aquesta lògica s'utilitza per tractar el paràmetre URL `search` en els Controllers, de forma genèrica independentment de l'entitat concreta. El text de la cerca s'escriu de la següent forma:

- Una condició té el format: `{nom atribut}{operador de comparació}{valor}`
- Les condicions es poden ajuntar utilitzant les paraules '`AND`', '`OR`' i parèntesis obert '`(`' i tancat '`)`' per agrupar condicions.
- Els operadors de comparació són:

Operador	Significat
:	Igual. Pot contenir asteriscs al principi del text o al final de text, semblant a un Regex.
!	Diferent
>	Més gran que
<	Més petit que
~	Semblant

Exemples de cerca:

- `name:*projecte`
- `backlogId:13 AND ( sprintId:1 OR sprintId:2 )`

Aquest codi que tracta i transforma la cerca el vaig modificar un cop per poder cercar per valors nuls o que no fossin nuls.

8 ANÀLISI I DISSENY DEL SISTEMA

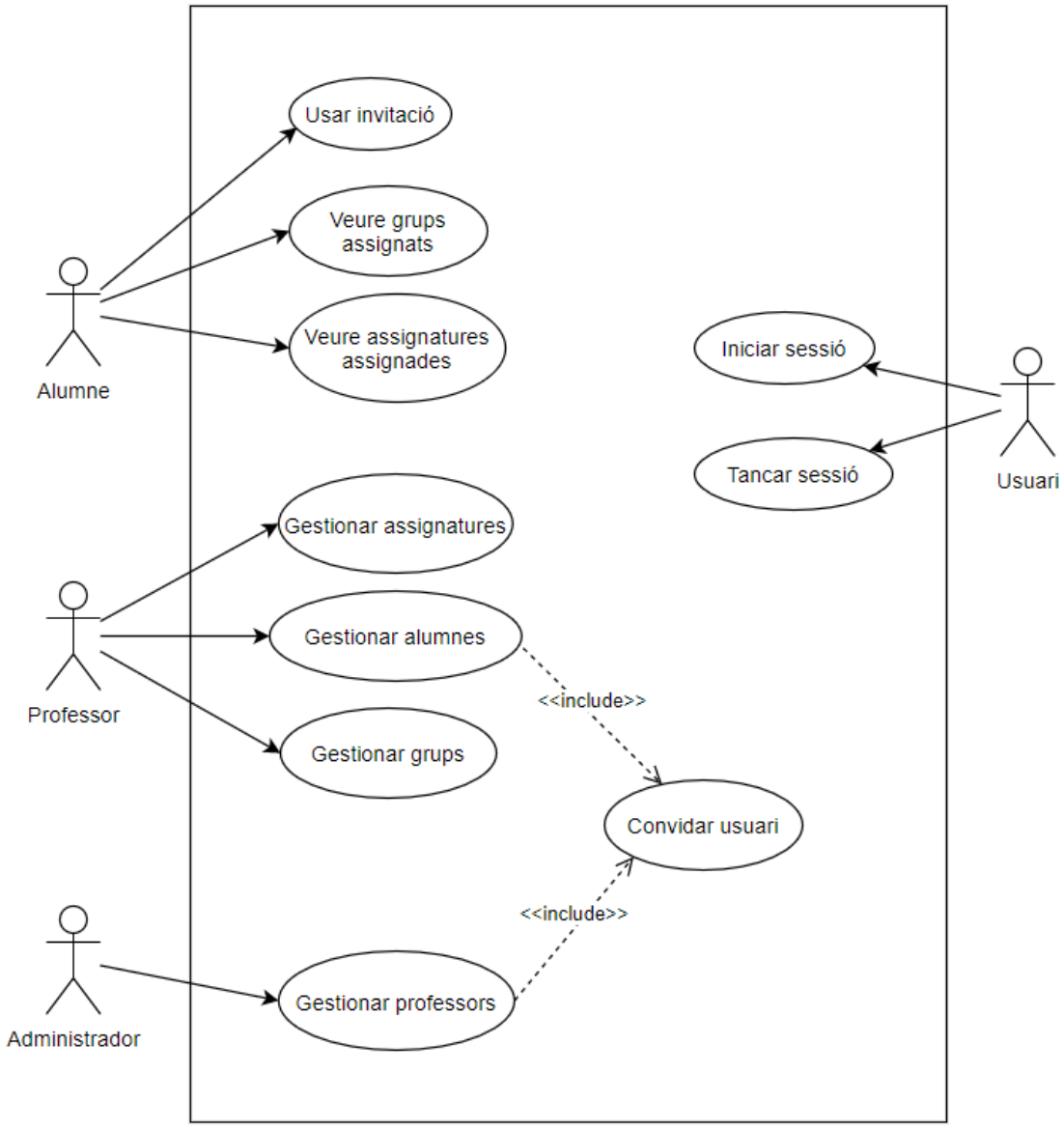
En aquest apartat es documenta l'anàlisi i el disseny de la solució.

8.1 ANÀLISI

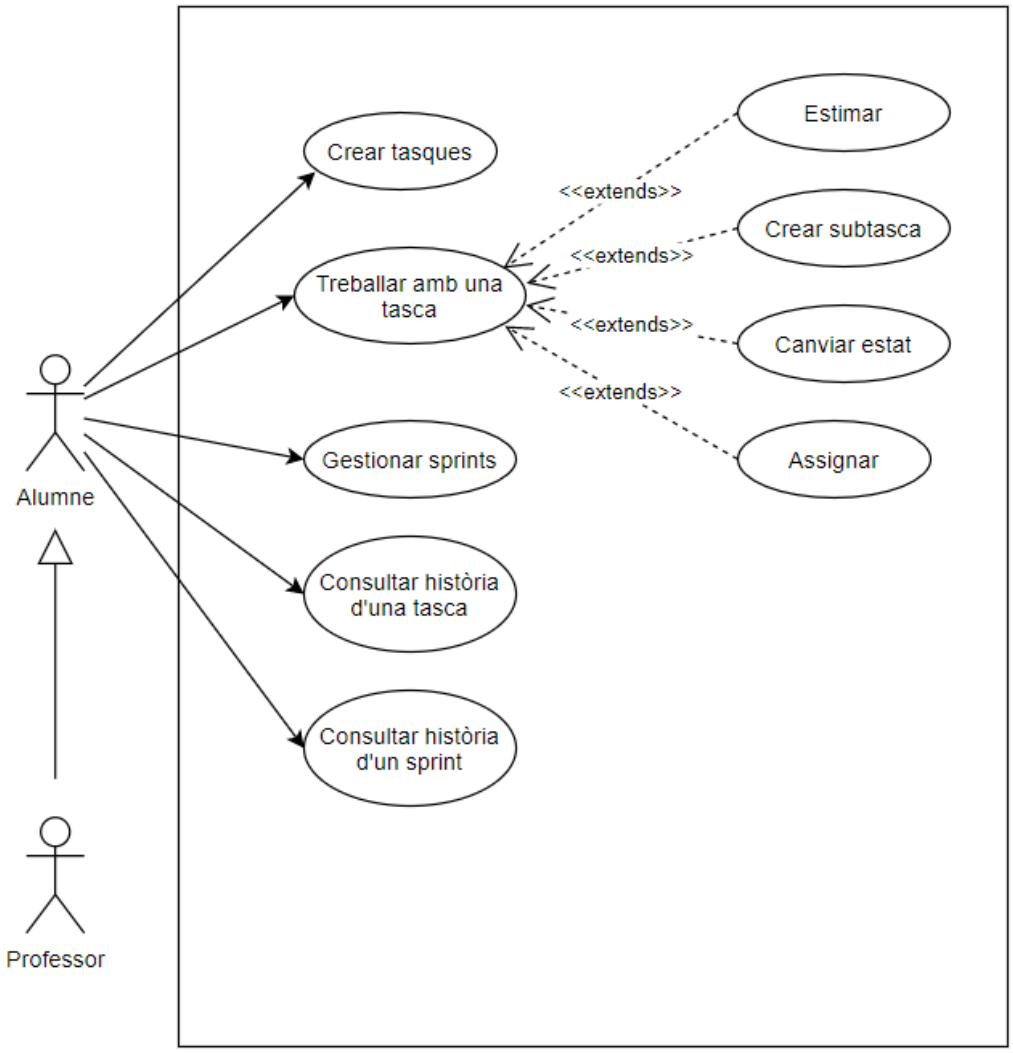
A continuació adjunto els diagrames de casos d'ús i algunes de les fitxes de casos d'ús fets durant la fase d'anàlisi per recollir les necessitats de l'aplicació.

Hi ha tres actors diferents segons el seu rol dins l'aplicació: l'**alumne**, el **professor** i l'**administrador**. L'actor **usuari** representa qualsevol usuari que interacciona amb l'aplicació independentment del seu rol.

8.1.1 Diagrama de casos d'ús



Il·lustració 10 Diagrama de casos d'ús per gestió d'assignatures, grups i usuaris



Il·lustració 11 Diagrama de casos d'ús de gestió de backlogs, tasques i sprints

8.1.2 Fitxes casos d'ús principals

A continuació s'inclouen les fitxes dels casos d'ús més representatius.

Convidar un usuari a l'aplicació
Actor: Administrador, Professor
Descripció: L'objectiu és convidar a un altre usuari a entrar a l'aplicació web, ja que el registre no és lliure.
Precondicions: Tenir sessió iniciada
Postcondicions: Invitació creada per uns rols

Flux principal:

1. Introduir dades de la invitació: adreça electrònica i rols
2. Crear invitació

Fluxos alternatius:

2. Si un usuari ja existeix amb aquesta adreça electrònica:
   2.1. Sistema mostra missatge d'error
2. Si ja hi ha una invitació creada pel mateix usuari per aquesta adreça electrònica

2.1. Sistema mostra missatge d'error

Gestionar assignatures

Actor: Professor

Descripció: L'objectiu és que el professor gestioni les assignatures: crear-ne, editar-ne les dades generals i eliminar-ne.

Precondicions: Tenir sessió iniciada

Postcondicions: Assignatura creada, modificada o eliminada

Flux principal:

1. Veure llista d'assignatures creades pel professor
2. Escollir operació a fer
3. Si crear assignatura
   2.1. Introduir dades: nom
   2.2. Crear assignatura
4. Si editar les dades
   3.1. Seleccionar assignatura
   3.2. Canviar les dades: nom
   3.2. Guardar modificacions
5. Si eliminar assignatura
   4.1. Seleccionar assignatura
   4.2. Escollir acció eliminar assignatura
   4.3. Sistema demana confirmació
   4.4. Confirmar i eliminar assignatura

Fluxos alternatius:

2.2. Si les dades són incorrectes:
   2.2.1. Sistema mostra missatge d'error
3.2. Si les dades són incorrectes:
   3.2.1. Sistema mostra missatge d'error

Convidar un alumne a un any de l'assignatura

Actor: Professor

Descripció: L'objectiu és convidar un alumne a afegir-se a un any concret. L'alumne pot haver-se registrat abans o no.

Precondicions: Tenir sessió iniciada i l'any de l'assignatura creat.

Postcondicions: Invitació creada per l'alumne per l'any de l'assignatura

Flux principal:

1. Seleccionar assignatura i any
2. Introduir adreça electrònica de l'alumne.
3. Crear invitació

Fluxos alternatius:

2. Si un alumne amb aquesta adreça electrònica ja està registrat a l'any de l'assignatura:
   2. 1. Sistema mostra missatge d'error
2. Si ja hi ha una invitació per l'any de l'assignatura per aquesta adreça electrònica:
   2.1. Sistema mostra missatge d'error

Crear grup
Actor: Professor
Descripció: L'objectiu és crear un grup de treball dins d'una assignatura i amb uns alumnes membres.
Precondicions: Tenir sessió iniciada i l'any de l'assignatura creat.
Postcondicions: Grup creat amb un backlog

Flux principal:
1. Seleccionar assignatura i any
2. Introduir nom del grup
3. Seleccionar alumnes com a membres
4. Crear grup

Fluxos alternatius:
4. Si les dades són incorrectes:
   4.1. Sistema mostra missatge d'error

Crear tasca
Actor: Alumne
Descripció: L'objectiu és crear una tasca dins un backlog del grup de treball
Precondicions: Tenir sessió iniciada i grup de treball assignat dins de l'assignatura
Postcondicions: Tasca creada

Flux principal:
1. Seleccionar grup de treball i backlog
2. Escollir opció crear tasca
3. Introduir dades generals de la tasca: nom
4. Crear

Fluxos alternatius:
4. Si les dades són incorrectes:
   4.1. Sistema mostra missatge d'error

Iniciar un sprint
Actor: Alumne
Descripció: L'objectiu de l'escenari és planificar les tasques a realitzar durant la següent iteració i iniciar l'sprint
Precondicions: Tenir sessió iniciada, grup de treball assignat dins de l'assignatura, tasques creades i cap sprint actiu
Postcondicions: Sprint creat i amb estat actiu, amb unes tasques assignades.

Flux principal:
1. Seleccionar grup de treball i backlog
2. Escollir opció crear sprint
3. Introduir dades generals de l'sprint: nom, data inici i data de fi.
4. Crear sprint
5. Pantalla mostra sprint en estat esbós
6. Planificar tasques
   6.1. Mentre es considera que hi ha espai:
      6.1.1. Moure tasques del backlog al sprint
   6.2. Si es considera que hi ha masses tasques o no són adients:

6.2.1. Moure tasques del sprint al backlog
7. Iniciar sprint

Fluxos alternatius:
4. Si les dades són incorrectes, per exemple data d'inici al passat:
4.1. Sistema mostra missatge d'error
6.1.1. Si no hi ha tasques al backlog o no les adients
6.1.1.1. Crear tasques

Editar una tasca
Actor: Alumne
Descripció: L'objectiu és crear una tasca dins un backlog del grup de treball
Precondicions: Tenir sessió iniciada i grup de treball assignat dins de l'assignatura
Postcondicions: Tasca modificada

Flux principal:
1. Seleccionar grup de treball i backlog
2. Seleccionar tasca de: backlog o sprint
3. Escollir camp a editar:
3.1. Nom
3.2. Estimació de punts
3.3. Estat
3.4. Alumne assignat
4. Modificar camp
5. Desar modificació

Fluxos alternatius:
5. Si les dades són incorrectes, per exemple data d'inici al passat:
5.1. Sistema mostra missatge d'error

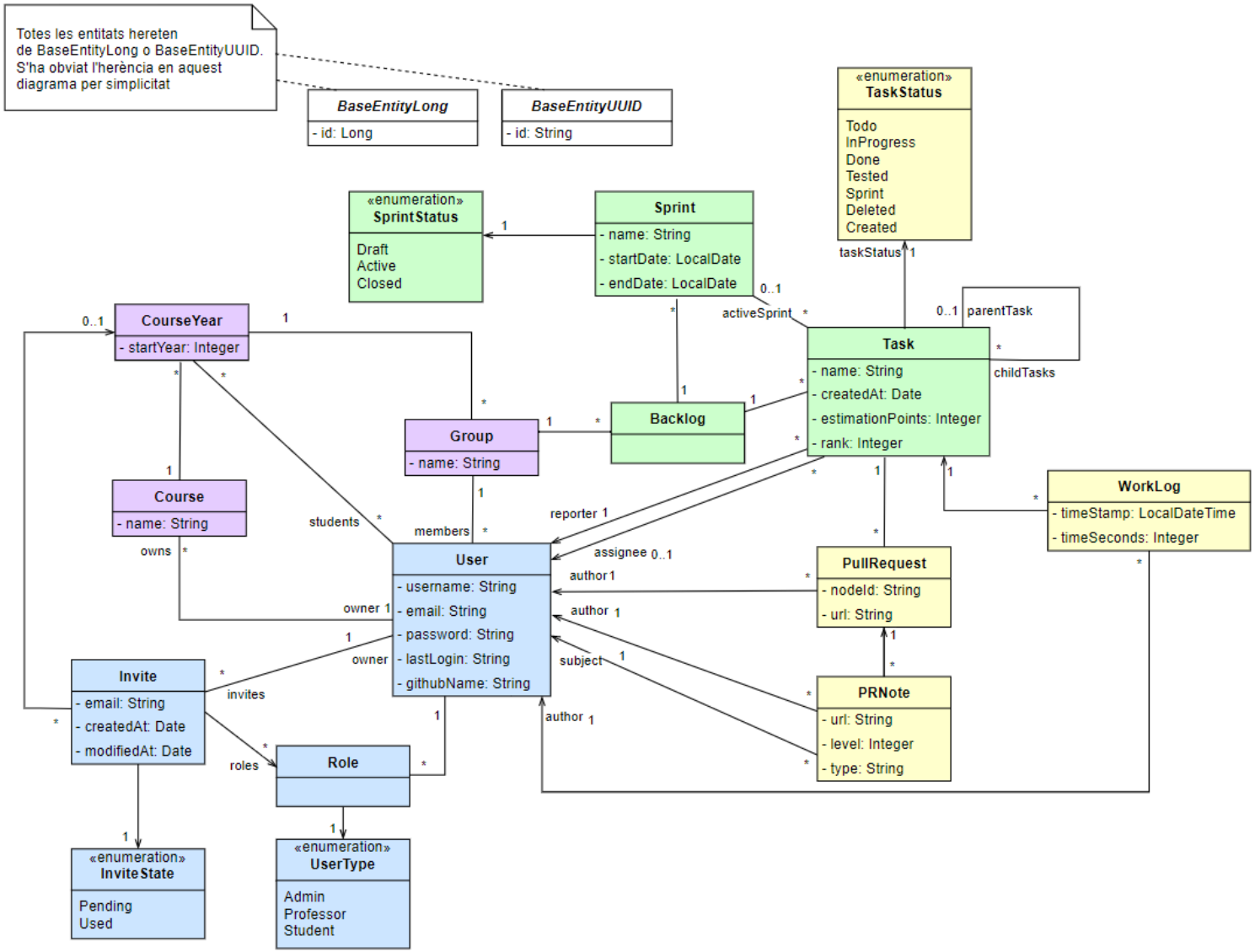
8.2 DISSENY

En aquest apartat mostro el model d'objectes, les crides de l'API REST i les interfícies d'usuaris.

8.2.1 Disseny d'objectes i dades

A continuació es mostra el diagrama de classes amb les entitats final. S'ha ampliat el model original amb el següent:

- Ajustat `Invitation` per poder convidar només a un any de l'assignatura, a part de poder convidar a un altre usuari a nivell de l'aplicació – per exemple, un altre professor o administrador. També s'han afegit atributs per poder deixar constància de quan es va convidar i acceptar la invitació.
- Afegir una entitat entre el `Course` i el `Group`, el `CourseYear`, per poder tenir englobades dins la mateixa assignatura els diferents anys.
- S'ha eliminat la `Iteration` perquè es vol deixar els alumnes crear els `Sprint`. Tot i que potser en el futur es vol restringir el rang de la data d'inici o data de fi, en realitat poden tenir diferents dates segons quin sigui el seu dia de grup de pràctiques.
- S'ha ampliat l'entitat `Task` amb més atributs.
- Les classes `ErrorEntity` i `IdObject` s'han mogut a un altre paquet anomenat `models` ja que només es feien servir en les respostes de l'API i no es persistien.



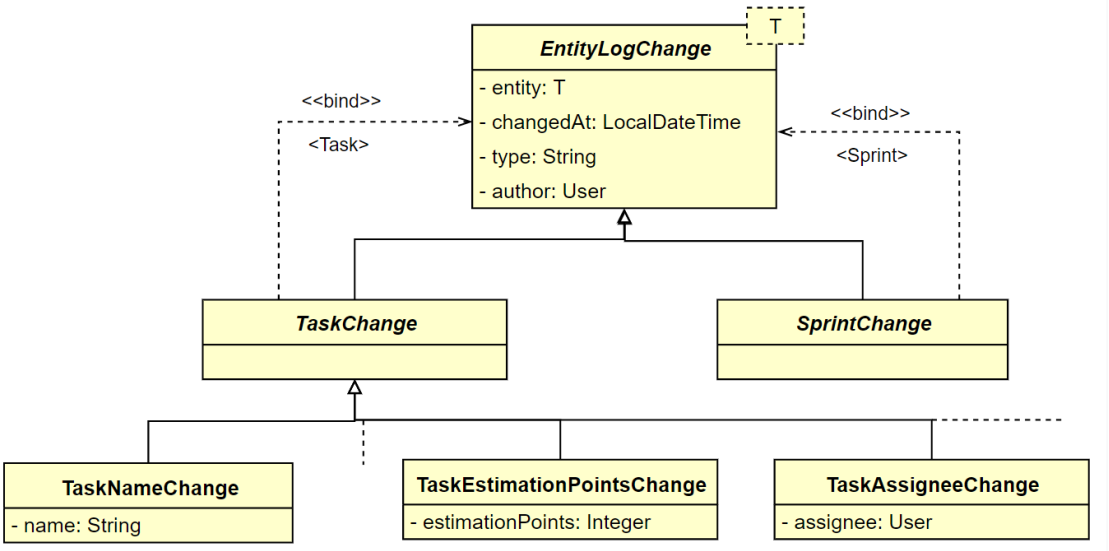
Il·lustració 12 Diagrama de classes sense mètodes i sense les entitats per enregistrar canvis

Per a facilitar la lectura del diagrama s'ha pintat en diferents colors la classe per classificar-los per àrees. Com es pot veure, hi ha les següents entitats:

- Entitats bases (en blanc):
    - **Classe `BaseEntityLong`**: Classe base amb el identificador que és de tipus `Long`. S'usa com a classe base de totes les entitats amb identificador d'aquest tipus i facilita la creació d'un repositori i service bases utilitzant classes parametritzades.
    - **Classe `BaseEntityUUID`**: Classe base amb el identificador que és de tipus `UUID`. S'usa amb la mateixa finalitat que `BaseEntityLong`.
- Entitats relacionades amb l'usuari i els seus rols (en blau clar):
    - **Classe `User`**: Representa un usuari registrat de l'aplicació i el seu perfil. Pot ser un alumne, professor o administració, depenent dels rols que tingui. Un usuari pot ser el propietari de varis `Course` i `Invite`. Pot estar registrat a varis anys d'una assignatura `CourseYear` i formar part de varis `Group` d'aquests.
    - **Classe `Role`**: Representa un tipus de rol al qual un usuari pertany. Només se'n persisteix un per cada tipus a la base de dades, el qual es relacionarà amb tots els usuaris amb aquell rol.
    - **Enumeració `UserType`**: Enumeració dels diferents tipus de rol de l'aplicació, hi ha tres: administrador, professor i alumne.
    - **Classe `Invite`**: Representa la invitació que un usuari fa a un correu electrònic per accedir a un recurs. Ara mateix es pot convidar a l'aplicació amb un rol o bé al curs d'una assignatura.
    - **Enumeració `InviteState`**: L'estat de la invitació, que pot estar pendent o acceptada. En el futur es podria afegir un nou estat declinat.
- Entitats relacionades amb la gestió de les assignatures i grups de treball (en lila):
    - **Classe `Course`**: Representa l'assignatura. Per exemple, una instància de `Course` podria representar l'assignatura "Projecte de Desenvolupament de Software" i una altra l'assignatura "Projecte Final de Grau". Com a detalls només té el seu nom. Un `Course` pot tenir varis `CourseYear`.
    - **Classe `CourseYear`**: Representa el curs d'una assignatura, o en altres paraules l'any acadèmic d'una assignatura. Mentre que `Course` és una entitat que persisteix al llarg dels anys, `CourseYear` representa l'assignatura dins un any acadèmic com seria "Projecte de Desenvolupament de Software 2020/2021" o "Projecte Final de Grau 2021/2022". És on els alumnes es matriculen i per tant on es registren amb invitacions. Per configurar-lo només cal definir l'any d'inici.
    - **Classe `Group`**: Representa el grup de treball format per varis alumnes de l'assignatura que desenvolupen un projecte junts dins de l'assignatura. Un grup normalment tindrà un sol `Backlog`, però en podria tenir més d'un si es fan varis projectes petits dins de l'assignatura.
- Entitats relacionades amb el backlog (en verd):
    - **Classe `Backlog`**: Representa la llista de tasques a desenvolupar per dur a terme un projecte o producte. Un `Backlog` té moltes `Task` i si es segueix Scrum tindrà varis `Sprint`.
    - **Classe `Task`**: Representa una tasca fer. Podria representar una prestació nova a implementar, un error a arreglar del programari o una tasca tècnica de millora. Té varis atributs: el nom, l'usuari que ha creat la tasca i la data de creació, l'usuari assignat per a realitzar-la, l'estimació en punts acordada amb tot el grup

de treball, la prioritat dins del `Backlog` i l'estat. També té una llista de `TaskChange` que representa la història de canvis fets al editar la tasca. Aquesta classe es pot veure al següent diagrama. Una tasca pot estar assignada a un `Sprint`.
    - **Classe `Sprint`**: Representa una iteració. És un període de temps indicat amb el `startDate` i el `endDate` en el qual el grup de treball treballa amb unes tasques planificades al principi d'aquest període. També té una llista de `SprintChange` amb la història de canvis fets.
    - **Classe `SprintStatus`**: L'estat de l'sprint, que pot ser en esbós – durant l'sprint planning -, actiu i tancat.
- Entitats que afegeixen funcionalitats addicionals a les entitats principals Task i Sprint (en groc):
    - **Enumeració `TaskStatus`**: L'estat de la tasca, que pot estar creada, oberta, per fer, en progrés, testejada, feta o tancada.
    - **Classe `WorkLog`**: Representa el registre d'hores treballades en una tasca un dia. S'indica el moment en què s'ha treballat i la quantitat de temps. La classe està dins del model d'entitats però encara no s'utilitza.
    - **Classe `PullRequest`**: Utilitzada per a guardar una referència al pull request creat dins l'eina on es guarda el codi. La classe està dins el model d'entitats però encara no s'utilitza. Hi ha un codi en progrés per guardar automàticament els pull requests d'un repositori de GitHub, que no s'ha tocat com a part d'aquest projecte.
    - **Classe `PRNote`**: Utilitzada per a guardar una referència a un tiquet de l'eina on es guarda el codi. Igual que els `PullRequest`, encara no s'utilitza.

També s'ha afegit una entitat nova `EntityLogChange` amb un paràmetre genèric, que té com subtipus `TaskChange` i `SprintChange`. La seva funció és enregistrar els canvis fets al backlog i als seus elements. Per exemple, `TaskEstimationPointsChange` representa una modificació de la tasca per afegir l'estimació nova `estimationPoints`. Ara per ara s'enregistra cada canvi durant una modificació de la tasca o sprint guardant el valor nou del camp modificat.



Il·lustració 13 Diagrama de classes parcial amb EntityLogChange, TaskChange, SprintChange i alguns dels subtipus de TaskChange

8.2.2 Disseny API REST

REST és una proposta d'arquitectura per dissenyar serveis webs proposada per Roy Fielding l'any 2000. Les seves sigles signifiquen Representational State Transfer (REST) i les seves principals idees són:

- El servei web està dissenyat al voltant de **recursos**. Els recursos no són sols entitats a la base dades, sinó que també poden ser imatges, fitxers, o un dispositiu físic com un sensor.
- Cada recurs té una **URI** que l'identifica de forma única.
- Els clients i servidors intercanvien **representacions** del recurs en un punt del temps. Aquesta representació està definida amb metadades.
- El servei web hauria de poder ser explorat mitjançat enllaços sense coneixements previs.
- S'hauria de tenir una interfície uniforme per treballar amb els diferents recursos.

API REST i HTTP

REST no està lligat a cap protocol específic, però tot hi així la majoria d'implementacions utilitzen HTTP com a protocol. Se solen seguir les següents convencions:

- L'API organitza les URIs en una jerarquia en forma d'arbre, en el qual el primer nivell és el nom d'una col·lecció de recursos i el segon l'identificador d'un recurs únic. Exemples de URIs:

URI	Recurs identificat
/imatges	La col·lecció de totes les imatges del sistema
/imatges/1	La imatge amb identificador 1
/usuaris/34	L'usuari amb identificador 34
/usuaris/34/imatges	Les imatges de l'usuari amb identificador 34

- S'utilitzen els verbs estàndard HTTP per fer operacions sobre els recursos de forma uniforme. Se sol utilitzar:

Mètode HTTP	Operació
GET	Obtenir un recurs o llista de recursos
POST	Crear un recurs nou
PUT	Modificar un recurs existent, substituint tot el recurs amb la nova representació enviada.
PATCH	Modificar parcialment un recurs existent.
DELETE	Eliminar un recurs.

- Les URIs haurien de promocionar els noms de les entitats per sobre de les operacions sobre aquestes. Per exemple, es recomana una URI com `DELETE /imatges/1` en comptes de `POST /modificar-imatge`.
- S'utilitzen els codis de resposta estàndard HTTP per indicar el resultat de la petició al servidor. Els més usats:

Codi de resposta HTTP	Què indica
200 OK	La petició ha anat bé i inclou al cos de la resposta una representació del recurs o el resultat de l'acció.
201 Created	La petició per crear un recurs ha anat bé.

204 No Content	La petició ha anat bé, però sense incloure cap contingut a la resposta.
404 Not Found	El recurs demanat no existeix.
400 Bad Request	El servidor no pot processar la petició ja que es considera que la petició té un format invàlid i amb errors per part del client. Se sol utilitzar com a resposta d'errors de validació de format o quan una acció no és vàlida per l'estat actual del recurs.
401 Unauthorized	La petició necessita autenticació.
403 Forbidden	La petició no s'accepta ja que la identitat autenticada no té accés al recurs.
500 Internal Server Error	Hi ha hagut un error inesperat al servidor.

- Els recursos es podrien representar en qualsevol format. Els més utilitzats són JSON i XML, ja que són entenedors tant per programes com per humans. S'utilitza el Header HTTP `Content-Type` com a metadata per indicar el format de la representació. Pel JSON i XML serien `application/json` i `application/xml` respectivament.
- Es poden utilitzar paràmetres URL (*query strings*) per a filtrar els recursos d'una col·lecció en funció del valor d'un dels seus atributs o per demanar una pàgina específica en cas de suportar paginació.

URIs i operacions dissenyades

S'han seguit les convencions anteriors per a definir les URIs de l'API i els mètodes HTTP utilitzats per fer operacions sobre les entitats de l'aplicació.

Per les modificacions d'entitats he escollit permetre operacions parcials amb el mètode PATCH. Pel que fa a les col·leccions relacionades a una entitat, en alguns casos tenen URI pròpia tant per obtenir com modificar-les i altres no. La regla que he seguit és per la mida d'aquesta col·lecció, per exemple els membres d'un grup són pocs i ja són inclosos dins la representació del grup en la petició `GET /groups/1` . En canvi la llista de tasques d'un backlog pot créixer molt i en un futur necessitar paginació per la qual tenen URI pròpia amb `GET /backlogs/19/tasks`.

A continuació es llisten les URIs definides.

Autenticació

Mètodes per gestionar la sessió.

Mètode	Camí	Descripció
POST	/auth/login	Iniciar sessió. Crea un JWT i afegeix la cookie necessària
POST	/auth/logout	Expirar la cookie
GET	/auth/check	Comprova si la sessió és vàlida
GET	/auth/self	Perfil de l'usuari autenticat

Invitacions

Mètode	Camí	Descripció
POST	/invites	Crea una invitació per registrar-se a l'aplicació amb certs rols
GET	/invites	Veure llistat d'invitacions creades per l'usuari. Suporta els paràmetres `type` i `courseYearId` per filtrar.

DELETE	/invites/{inviteId}	Elimina l'invitació
GET	/users/self/invites	Veure les invitacions obertes, tant de rols com d'assignatures.
PATCH	/users/self/invites/{inviteId}	Modificar invitació acceptant-la

Usuaris

Mètode	**Camí**	**Descripció**
GET	/users/{id}	Veure el perfil públic d'un usuari
POST	/register	Donar-se d'alta a l'aplicació fent servir una invitació

Assignatures

Mètode	**Camí**	**Descripció**
GET	/courses	Veure cursos creats pel propi usuari i cercar amb el paràmetre `search`
GET	/courses/{id}	Veure una assignatura
POST	/courses	Crear una assignatura
PUT	/courses/{id}	Modificar les dades generals d'una assignatura
DELETE	/courses/{id}	Eliminar assignatura

Any assignatura

Mètode	**Camí**	**Descripció**
GET	/courses/years	View enrolled course years as Student
GET	/courses/years/{yearId}	View one course year
POST	/courses/{courseId}/years	Crear un any acadèmic nou per una assignatura
DELETE	/courses/years/{yearId}	Eliminar un any acadèmic d'una assignatura
POST	/courses/years/{yearId}/invites	Convidar un usuari o email a una any de l'assignatura
GET	/courses/years/{yearId}/students	Veure els estudiants d'un any de l'assignatura
DELETE	/courses/years/{yearId}/students/{username}	Treure un estudiant d'un any de l'assignatura

Grups de treball

Mètode	**Camí**	**Descripció**
GET	/courses/years/{yearId}/groups	Veure els grups d'un any de l'assignatura
POST	/courses/years/{yearId}/groups	Crear un grup nou
GET	/groups/{groupId}	Veure un grup concret
PATCH	/groups/{groupId}	Modificar les dades generals d'un grup concret
DELETE	/groups/{groupId}	Eliminar un grup

Backlogs

Mètode	**Camí**	**Descripció**
GET	/backlogs/{id}/tasks	Veure les tasques d'un backlog. Suporta un paràmetre opcional `search` per filtrar
POST	/backlogs/{id}/tasks	Crea una tasca nova al backlog
GET	/backlogs/{id}/sprints	Veure els sprints d'un backlog
POST	/backlogs/{id}/sprints	Crea un sprint nou al backlog

Sprints

Mètode	Camí	Descripció
GET	/sprints/{id}	Veure un sprint. Suporta un paràmetre opcional `search` per filtrar
PATCH	/sprints/{id}	Modificar un sprint
GET	/sprints/{id}/history	Veure la història de canvis d'un sprint com ara canvis de nom o afegir i treure tasques. Suporta un paràmetre opcional `search` per filtrar

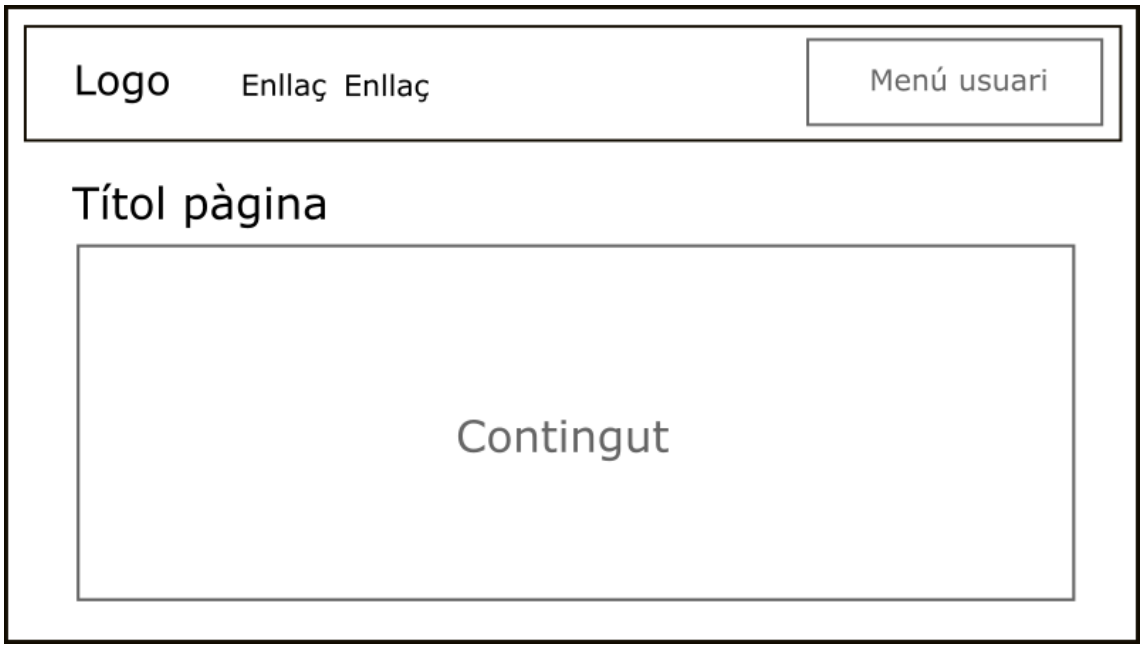
Tasques

Mètode	Camí	Descripció
GET	/tasks	Veure i buscar tasques a tota l'aplicació. Suporta un paràmetre `search` per filtrar en funció dels atributs de la tasca i el paràmetre `backlogId` per reduir la cerca i controlar l'accés.
GET	/tasks/{id}	Veure una tasca
GET	/tasks/{id}/history	Veure la història de canvis d'una tasca com ara canvis d'assignació o d'estat. Suporta un paràmetre opcional `search` per filtrar
PATCH	/tasks/{id}	Modificar una tasca
POST	/tasks/{id}/subtasks	Crear una subtasca sota una tasca pare

8.2.3 Disseny d'interfícies d'usuaris

En aquesta fase calia identificar les interfícies d'usuari de l'aplicació.

Totes les pàgines de l'aplicació web tindran les següents seccions: capçalera i contingut.



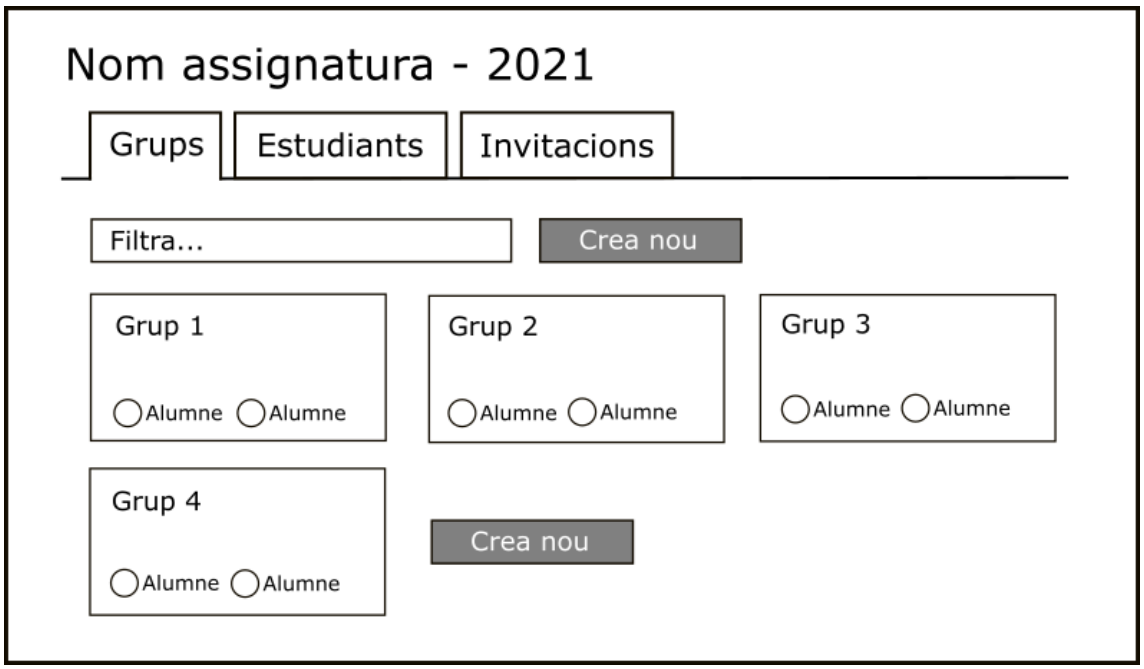
Il·lustració 14 Esquema de pàgina general de totes les pàgines

El menú de navegació principal tindrà dos estats: el menú visible per usuaris anònims que no han iniciat sessió i el menú visible pels que l'han iniciat:

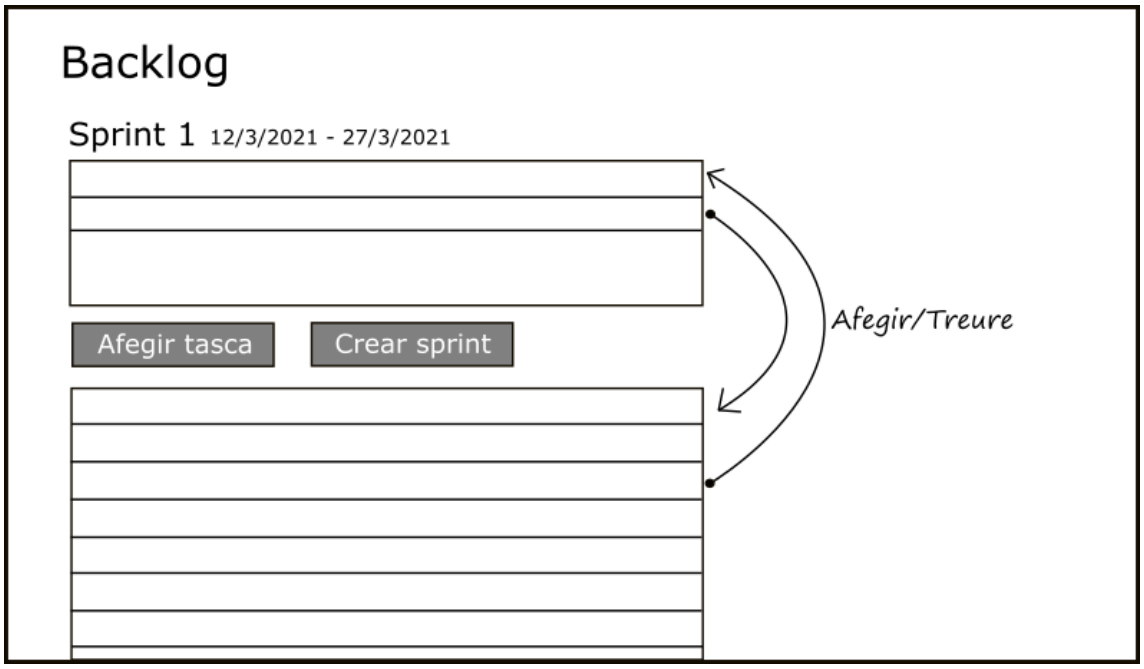
Il·lustració 15 Menú usuari anònim

Il·lustració 16 Menú usuari amb sessió iniciada

A continuació es mostren l'esquema de pàgina de l'any d'una assignatura i la vista principal del backlog. En aquests s'obvien la capçalera ja que és un element persistent en totes les pàgines.



Il·lustració Esquema de pàgina de l'any d'una assignatura

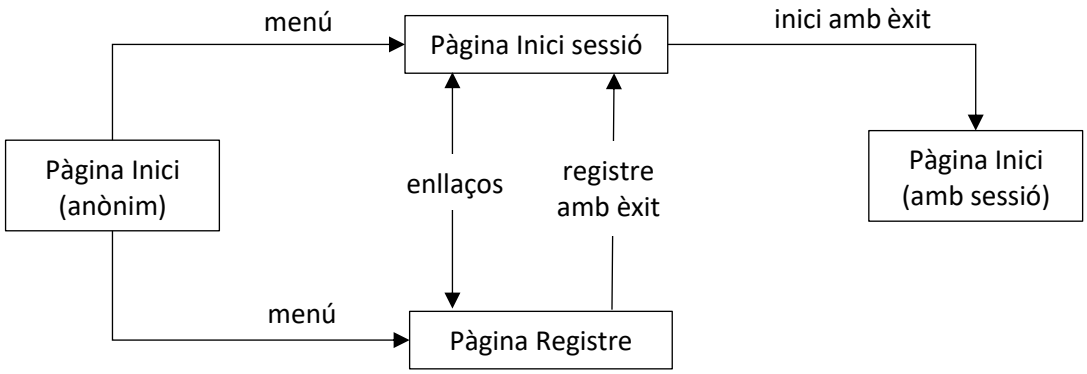


Il·lustració 17 Esquema de pàgina de la vista principal del backlog amb un sprint en mode esbós

El camí que segueix l'usuari navegant per les pàgines de l'aplicació seria semblant al que es mostra a continuació en figures.

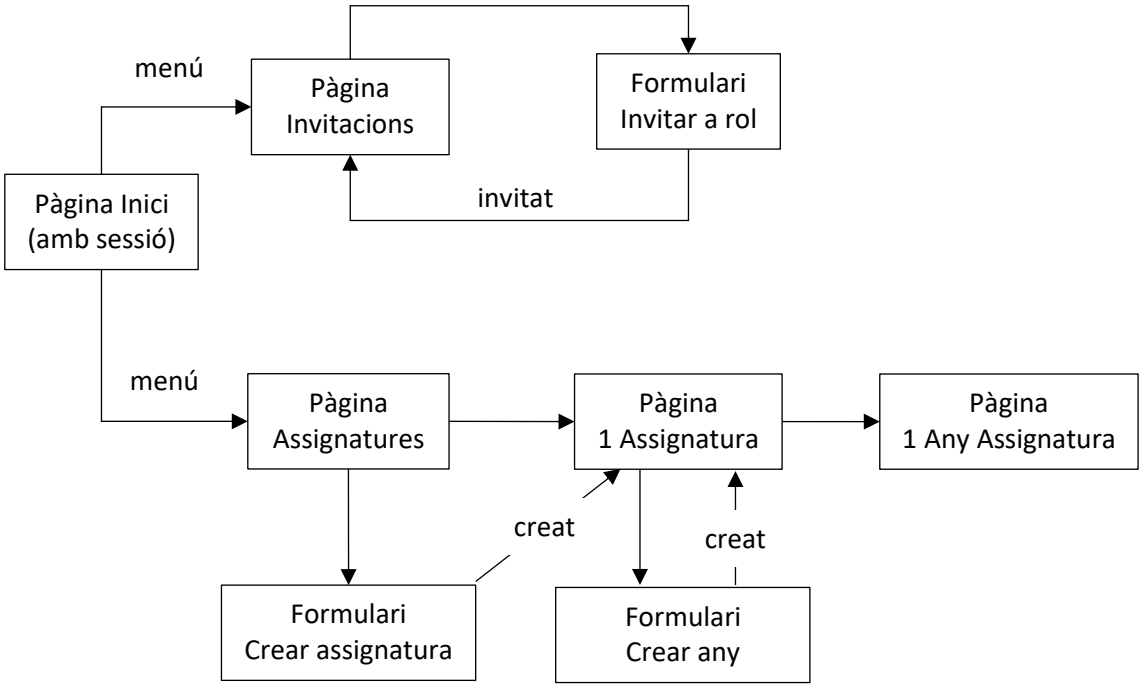
Les pàgines tenen una ruta individual. Les seccions en una primera fase no estava clar si serien pàgines, pestanyes o quelcom semblant a pestanyes. La majoria han acabat sent pestanyes. Els formularis la majoria són pàgines individuals, però altres són formularis incrustats o modals.

En qualsevol moment l'usuari pot tornar a la pàgina d'inici mitjançat el menú de navegador i idealment les pàgines han de tenir un element indicant la ruta de navegació de la pàgina actual (breadcrums).



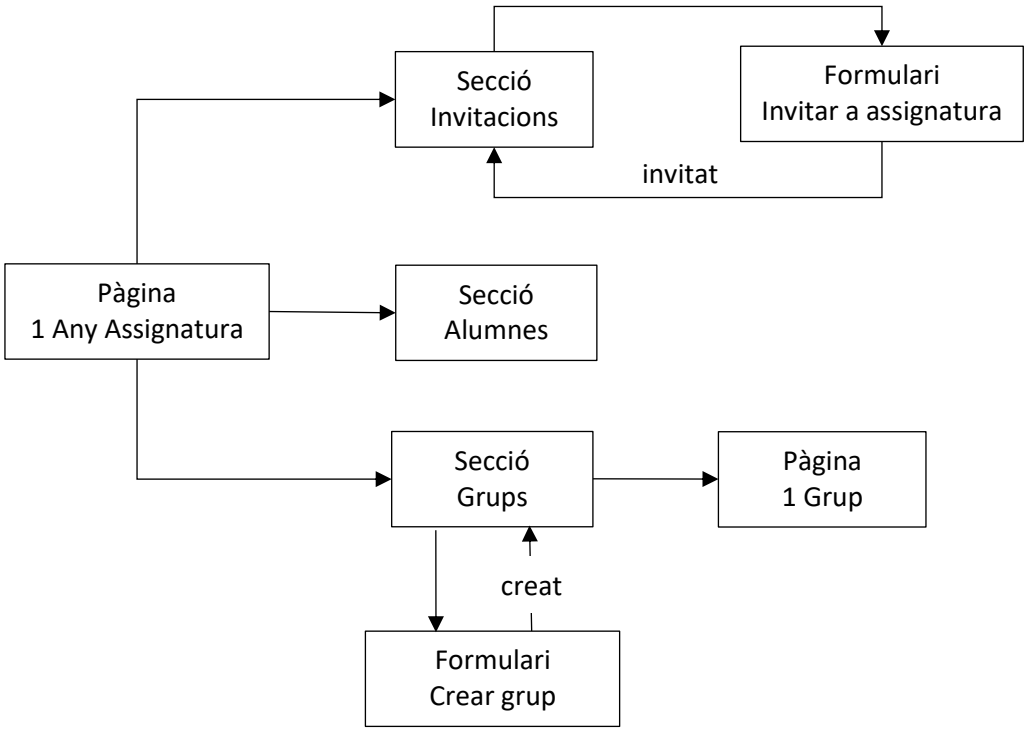
Il·lustració 18 Flux de navegació entre pàgines per la part d'iniciar sessió i registre

La figura següent mostra el flux de navegació entre pàgines per un professor. Per l'alumne seria més senzill ja que no veu la pàgina d'assignatures (només s'enregistra a any concret) i tampoc pot convidar ni crear assignatures.



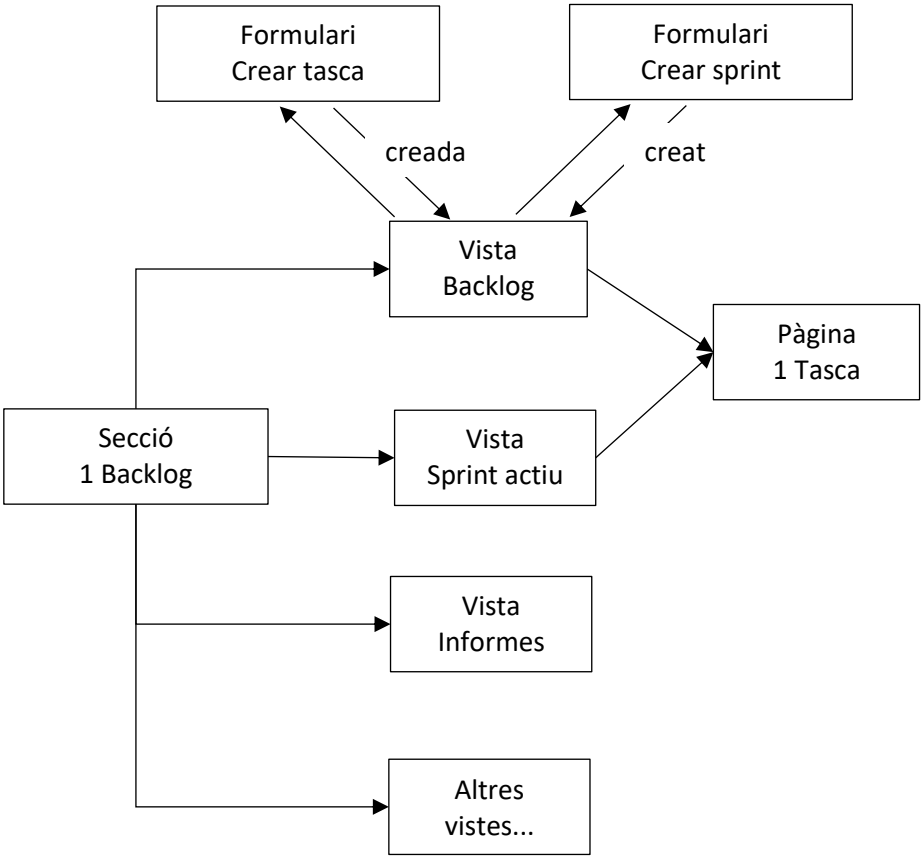
Il·lustració 19 Flux de navegació entre pàgines per la part de gestionar invitacions i assignatures

El següent també correspondria al professor. L'alumne només veuria la secció de grups.



Il·lustració 20 Flux de navegació entre pàgines per la part de gestionar un any acadèmic de l'assignatura

Les pàgines d'una assignatura, l'any d'un assignatura i el grup haurien de permetre mitjançant un menú o opcions l'edició de les dades i eliminar-los.



Il·lustració 21 Flux de navegació entre pàgines i vistes per la part de gestionar un backlog i les seves tasques

9 IMPLEMENTACIÓ I PROVES

9.1 SERVIDOR: REST API

9.1.1 Les capes

Per implementar una ruta de l'API hi ha implicades vàries classes: un Controller, un Service, un Repository i una Entitat. Totes es marquen amb anotacions, respectivament: `@RestController`, `@Service`, `@Component` i `@Entity`. Com a regla general, cada entitat té el seu Controller, Service i Repository.

Web Service: Controller

El Controller processa les peticions HTTP, es comunica amb el Service i retorna la resposta. Cada ruta es correspon a un mètode i està definit amb una de les següents anotacions del framework de Spring: `@GetMapping`, `@PostMapping`, `@PatchMapping` i `@DeleteMapping`.

```
// Imports

@RestController
@RequestMapping(path = "/tasks")
public class TaskController extends CrudController<Task, TaskService> {
    @Autowired
    BacklogService backlogService;

    @Autowired
    TaskChangeService taskChangeService;

    @Autowired
    AccessChecker accessChecker;

    // Altres mètodes

    @GetMapping(path = "/{id}")
    @JsonView(EntityLevelViews.Basic.class)
    public Task getTask(Principal principal, @PathVariable("id") Long id) {
        String userId = super.getUserId(principal);
        Task task = service.get(id);
        accessChecker.checkCanViewBacklog(task.getBacklog(), userId);
        return task;
    }

    @PatchMapping(path = "/{id}")
    @JsonView(EntityLevelViews.Basic.class)
    public Task editTask(Principal principal,
                         @PathVariable(name = "id") Long id,
                         @Valid @RequestBody MergePatchTask taskRequest) {
        String userId = super.getUserId(principal);
        Task modifiedTask = service.editTask(id, taskRequest, userId);
        return modifiedTask;
```

```
    }

    // Altres mètodes
}
```

Taula 1 Fragments de TaskController mostrant els mètodes per processar les rutes GET /tasks/{id} i PATCH /tasks/{id}

Lògica de negoci: Service

El Service obté les entitats utilitzant els repositoris i gestiona la lògica de la petició, creant o modificant les entitats.

```
// Imports

@Service
public class SprintService extends BaseServiceLong<Sprint, SprintRepository> {

    // Injected dependencies

    @Transactional
    public Sprint create(Long backlogId, String name, LocalDate startDate, Loca
lDate endDate, String userId) {
        Backlog backlog = backlogService.get(backlogId);
        accessChecker.checkCanManageBacklog(backlog, userId);

        Sprint sprint = new Sprint(name);
        sprint.setStartDate(startDate);
        sprint.setEndDate(endDate);
        sprint.setBacklog(backlog);
        backlog.addSprint(sprint);
        this.repo().save(sprint);

        return sprint;
    }

    // Altres mètodes
}
```

Taula 2 Fragment de SpringService mostrant el mètode createSprint

Aquí en aquesta capa s'ha creat una classe `AccessChecker` que centralitza les comprovacions de permisos per llegir o modificar un recurs en concret en funció de la l'usuari de la petició. Es valora si l'usuari és el propietari del recurs o hi està relacionat de manera lícita, com per exemple que el recurs pertany al grup de treball on s'és membre. Es crida des dels Services per comprovar si l'usuari que fa una modificació té permisos d'escriptura i des del Controller per comprovar si té permisos de lectura.

Persistència: Repository i Entity

L'Entity representa un model del domini de l'aplicació, com ara una tasca o un usuari. Alguns dels seus atributs s'anoten amb anotacions de l'API de persistència de Java EE, JPA, per tal d'indicar com s'han de reflectir a la base de dades. També es poden anotar les classes o mètodes

getters amb anotacions de la llibreria de serialització Jackson per a controlar la serialització en la resposta de la crida HTTP.

```
// Imports

@Entity
@Table(name = "tasks")
@JsonIdentityInfo(
    generator = ObjectIdGenerators.PropertyGenerator.class,
    property = "id"
)
public class Task extends BaseEntityLong {

    public static final int NAME_LENGTH = 100;

    // Constructors

    @NonNull
    @Column(length = NAME_LENGTH)
    private String name;

    @ManyToOne
    @JoinColumn(name = "backlogId")
    private Backlog backlog;

    @Column(name = "backlogId", insertable = false, updatable = false)
    private Long backlogId;

    @ManyToOne
    private User reporter;

    private Date createdAt;

    // Altres atributs

    @OneToMany(mappedBy = "parentTask")
    private Collection<Task> childTasks;

    // Altres atributs

    @NonNull
    @JsonView(EntityLevelViews.Basic.class)
    public String getName() {
        return name;
    }

    public void setName(@NonNull String name) {
        this.name = name;
    }
```

```
    @JsonView(EntityLevelViews.Basic.class)
    @JsonSerialize(using = JsonDateSerializer.class)
    public Date getCreatedAt() { return createdAt; }

    @JsonView(EntityLevelViews.Basic.class)
    public User getReporter() { return reporter; }

    // Altres mètodes: getters, setters i altres
}
```

Taula 3 Fragment de l'entitat Task mostrant alguns dels seus atributs i mètodes

El Repository és molt petit i senzill gràcies al framework de Spring que proveeix les operacions més bàsiques de gestió d'entitats – obtenir, guardar actualitzacions i eliminar – de forma senzilla declarant una interfície que hereti del repository base de Spring `JpaRepositoryRepository<T,ID>`.

```
package org.udg.trackdev.spring.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.udg.trackdev.spring.entity.BaseEntityLong;

public interface BaseRepositoryLong<T extends BaseEntityLong> extends JpaReposi
tory<T, Long>, JpaSpecificationExecutor<T> {
}
```

Taula 4 Codi de la interfície base BaseRepositoryLong

```
package org.udg.trackdev.spring.repository;

import org.springframework.stereotype.Component;
import org.udg.trackdev.spring.entity.Group;

@Component
public interface GroupRepository extends BaseRepositoryLong<Group> {
}
```

Taula 5 Codi de la interfície del repository GroupRepository

Si algun repositori requereix operacions específiques, es poden complementar de vàries formes:

- Declarant mètodes a la interfície amb una nomenclatura acordada. Spring parseja el nom del mètode i genera la consulta.

  ```
  List<Role> findByUserType(@Param("userType") UserType userType);
  ```

  Taula 6 Exemple de mètode que es generarà a una consulta

- Declarant mètodes a la interfície i escrivint a mà amb el Java Persistence Query Language (JPQL) la consulta. També és possible escriure-la amb el llenguatge natiu, SQL, però no es recomana.

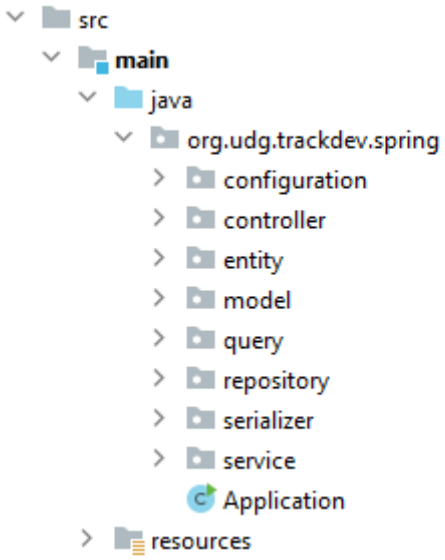
```
@Query("SELECT t FROM Task t JOIN t.backlog b WHERE b.id = :backlogId AN
D ( t.rank BETWEEN :fromRank AND :untilRank ) ORDER BY t.rank ASC")
List<Task> findAllBetweenRanksOfBacklog(
                @Param("backlogId") Long backlogId,
                @Param("fromRank") Integer fromRank,
                @Param("untilRank") Integer untilRank);
```

Taula 7 Exemple de mètode amb consulta escrita en JPQL

- Heretant de `JpaSpecificationExecutor<T>` que permet crear consultes programàticament sense generar el text de la consulta amb el JPQL utilitzant les Specification de Spring.

9.1.2 Organització del codi

Aquí mostro en una captura de pantalla l'organització del codi en paquets.



Il·lustració 22 Captura de pantalla de l'estructura del codi en paquets

Les classes s'han organitzat en diferents paquets segons la capa on pertanyen o per responsabilitat.

- **configuration**: Conté les configuracions generals de l'API utilitzant Spring. Es defineixen comportaments globals com és la validació dels tokens JWT en cada petició que arriba a l'API o la configuració del CORS.
- **controller**: Agrupa els controllers, que s'han descrit anteriorment. Equival a la capa de presentació.
- **entity**: Agrupa les entitats, amb el model de domini de l'aplicació.
- **model**: Conté classes simples que s'utilitzen només com a resposta de l'API o com a model d'una petició de modificació amb PATCH.
- **query**: Conté el codi necessari per transformar un text d'un paràmetre de cerca a una cerca amb les classes de l'API Specification de Spring Data.
- **repository:** Conté les interfícies usades per a recuperar entitats de la base de dades i guardar-ne les modificacions.
- **serializer:** Conté classes utilitzades per a personalitzar la serialització de determinats tipus a la resposta JSON de l'API.
- **service:** Agrupa els services. Equival a la capa de negoci.

9.2 CLIENT: SPA

9.2.1 React en detall

React es basa en components. Cada component pot rebre unes propietats de fora i contenir el seu propi estat intern dels qual en genera una descripció de la interfície. Aquesta descripció seria l'HTML a renderitzar o actualitzar en el DOM de la pàgina. Cada cop que hi ha un canvi d'estat es programa una actualització de la interfície. Un exemple de component senzill seria el següent:

```
const TemperatureInput = (props) => {
    const [temperature, setTemperature] = useState('')

    function handleChange(event) {
        setTemperature(event.target.value)
    }

    return (
        <fieldset>
            <legend>Enter temperature in {props.scale}:</legend>
            <input value={temperature} onChange={handleChange} />
        </fieldset>
    )
}
```

El que s'ha de tornar per descriure la interfície és un objecte anomenat element de React creat amb `React.createElement`. Tanmateix, el més estàndard és utilitzar la sintaxi estesa de Javascript JSX, que l'eina Babel compila a `React.createElement`. Tot seguit es mostra un exemple d'aquesta transformació:

```
const element = (<h1 className="greeting">Hello, world!</h1>)
```

Taula 8 Exemple utilitzant JSX, que necessita compilació amb Babel

```
const element = React.createElement(
    'h1',
    {className: 'greeting'},
    'Hello, world!'
)
```

Taula 9 Exemple utilitzant React.createElement, que seria el Javascript real a utilitzar

Els components poden tenir altres components, creant una jerarquia:

```
const UserMention = ( { user }) => {
  return (
    <span key={user.username} className="user-mention">
      {user.username}
    </span>
    )
}

const UsersList = ( { users }) => {
```

```
  return (
    <span className="user-list">
      {
        users?.map(user => (<UserMention key={user.username} user={user} />))
      }
    </span>
  )
}
```

Taula 10 Exemple de component utilitzant un altre component fent una llista

Dins el JSX es diferencia un component de React d'una etiqueta HTML normal per tenir la primera lletra en majúscula.

Camí de dades: de dalt cap a baix

La idea principal de React és que cada component hauria de ser encapsulat, el seu estat ha de ser intern i les propietats no es poden modificar. El camí de les dades sempre és d'un component pare cap a baix en forma de propietats. Els components fills no poden modificar-les ja que són propietats de només lectura per ells. Només hi ha d'haver un sol responsable d'una dada, no hauria d'estar duplicada en l'estat de dos components.

Aixecar l'estat cap amunt

Una tècnica que explica React per a gestionar estat que varis components necessiten és el que anomenen aixecar l'estat cap amunt ("*lifting state up*"). L'avantpassat comú més proper és qui gestiona l'estat, i els fills comuniquen de canvis amb funcions de callback.

L'API de Context

React també ofereix l'API de Context per poder tenir un estat global per tot un arbre de components, normalment tota l'aplicació, i així haver de passar propietats avall en molts nivells manualment per compartir dades. Com a exemples d'estat que es poden implementar amb el Context es proposa l'usuari autenticat, el tema de la pàgina – com fosc o clar – i la configuració de llenguatge.

Class Components i Function Components

Els components de React es poden escriure de dues maneres:

- **Class Component:** escrivint una classe que hereti de React.Component i fent override de mètodes que s'executen en determinats moments del seu cicle de vida. Com a mínim s'ha d'implementar el mètode render, amb el JSX a renderitzar.
- **Function Component:** escrivint una funció de Javascript que rep com a paràmetre les propietats i retorna el JSX a renderitzar. A dins es pot utilitzar l'API nova de Hooks de React per tenir estat o accedir a funcionalitats de React, sense haver d'escriure una classe.

En una sola aplicació o arbre de components es poden combinar components escrits en classes o en funcions, són totalment compatibles. El que no és compatible és utilitzar l'API de Hooks dins d'un Class Component. Alguna funcionalitat no és disponible en els Function Component, però bàsicament tot és possible.

En un principi em va semblar més entenedor les classes per a escriure components que tenen estat, i Function Component per components sense estat que simplement tornaven HTML. Però a mesura que anava implementant, vaig acabar utilitzant Function Component per tots.

Des de part de l'equip de React es diu que mai deixaran de suportar Class Components i dins de la seva secció de preguntes següents recomanen provar la nova API de Hooks per components nous. Aquesta API va sortir amb la release de React 16.8 el febrer de 2019.

Hooks

Els Hooks són funcions que es diferencien per començar per la paraula "use" i les més utilitzades són:

- **useState:** retorna una valor amb estat i una funció per actualitzar-lo. Al actualitzar es programa una renderització.
- **useEffect:** usat per poder executar una funció després d'un canvi del component amb efectes colateral. Per exemple, anar buscar un resultat de l'API si canvia el valor d'un filtre. Per defecte la funció s'executarà després de cada renderització, però es pot definir quines variables externes provoquen l'efecte, o cap si només es vol que s'executi el primer cop.
- **useContext:** utilitzat per obtenir un context definit en pares superiors.

9.2.2 Gestió de l'usuari autenticat

En aquest apartat explicaré com he implementat a la part del client la gestió de la sessió.

Les crides API relacionades són:

Mètode HTTP	Ruta	Descripció
GET	/auth/self	Torna el perfil de l'usuari actual en format JSON, amb dades com el nom de l'usuari i rols al qual pertany. Si no hi ha usuari autenticat torna un 403.
POST	/auth/login	Si el nom d'usuari i contrasenya enviats són correctes, inicia sessió. Més concretament: • Posa en una galeta "trackdev_JWT" amb un access token. • Retorna en el cos de la resposta el perfil de l'usuari i l'access token, per ser consumit per altres clients com una aplicació mòbil.
POST	/auth/logout	Tanca la sessió. Més concretament, esborra la galeta "trackdev_JWT".

Per a guardar l'estat de la sessió i també deixar-la a disposició de qualsevol component de l'aplicació de React vaig crear un Context anomenat UserContext. Aquest UserContext contindrà un objecte amb l'estat de la sessió i una funció per actualitzar-lo.

El Context de UserContext embolcalla tot l'arbre de components.

Al renderitzar per primer cop l'App, bàsicament al carregar la pàgina o en un recàrrega – tipus refresh de la pàgina amb la tecla F5 – es fa una crida a l'API a la ruta "/auth/self" per saber si l'usuari està autenticat o no. Això es fa amb el hook useEffect. Segons la resposta d'aquesta crida, es canvia el valor del UserContext amb la sessió iniciada o anònima.

Els components creats que utilitzen UserContext són:

- **Home**: Mostra la salutació a l'usuari amb el seu nom, al més pur estil de Hello World.
- **HeaderLinks**: Mostra el menú de navegació si l'usuari té sessió iniciada, altrament mostra els enllaços per iniciar sessió i registre.
- **HeaderUserLinks**: Mostra el menú de l'usuari amb el seu nom i enllaç per tancar sessió. Apart de llegir l'usuari del UserContext, pot modificar-ne l'estat al tancar sessió.
- **Login**: Un cop la petició de login ha anat bé de part de l'API REST, modifica l'estat de UserContext amb el perfil de l'usuari.
- **EnsureLoggedIn**: Per restringir l'accés a components i redirigir a la pàgina de login.
- **Restricted:** Per restringir l'accés a components segons els rols de l'usuari.
- **CourseYear:** Per amagar l'accés a components. En aquest cas per particularitats de l'interfície no es podia re-utilitzar l'anterior i necessita aquesta dependència amb UserContext.

Els més interessants són `EnsureLoggedIn` i `Restricted`. La seva funció és bàsicament assegurar que l'usuari està autenticat per certs components o que té uns rols específics.

El component `EnsureLoggedIn` mira si l'usuari està autenticat i mostra els components fills, altrament si no ho està redirigeix l'usuari anònim a la pàgina d'iniciar sessió. El faig servir per embolcallar components Route de react-router que necessiten l'usuari autenticat. Bàsicament totes les pàgines requereixen sessió iniciada, exceptuant l'arrel i les pàgines d'iniciar sessió i registre. Exemple d'utilització:

```
<Route exact path="/tasks/:taskId">
  <EnsureLoggedIn>
    <Task />
  </EnsureLoggedIn>
</Route>
```

Taula 11 Exemple d'utilització del component EnsureLoggedIn a la declaració de rutes dins de /pages/MainRoutes.js. La pàgina de la tasca sempre requereix un usuari amb sessió

El component `Restricted` comprova si l'usuari té certs rols. Si els té mostra els components fills, altrament mostra un component alternatiu passat o res. Per exemple, es fa servir per permetre l'accés a la pàgina de crear curs a només els professors:

```
return (
  <Restricted allowed={["PROFESSOR"]} fallback={(<p>You don't have access to he
re.</p>)}>
    <div>
      <CourseHeaderEditable
        course={course}
        onCourseChange={this.handleCourseChange} />
      <h3>Course years</h3>
      <CreateCourseYear courseId={course.id} onCourseTouched={this.handleCourse
Touched} />
      <CourseYearsList courseId={course.id} courseYears={course.courseYears} />
    </div>
  </Restricted>
)
```

Taula 12 Exemple d'utilització del component Restricted a la pàgina de Courses - l'alumne només pot visualitzar un Course Year concret

9.2.3 Rutes

Per a implementar les diferents pàgines amb diferents urls s'ha utilitzat bàsicament els següents components de la llibreria addicional React-Router:

- **Router:** Element necessari com a avantpassat al nivell més alt on es vulguin utilitzar la resta de
- **Switch:** Contenidor de rutes, en el quals només s'aplica una, la primera que encaixa.
- **Route:** Defineix una ruta.
- **Link:** Es comporta com un enllaç i fa canviar la ruta però no provoca carregar la pàgina com passaria amb una etiqueta <a> normal, amb el salt visual que pot causar per la inicialització de l'aplicació de React altre cop i la comprovació de la sessió de l'usuari.
- **NavLink:** El mateix que l'anterior, però aplica un estil si l'enllaç coincideix amb la ruta activa actualment.

La seva utilització és molt entenedora, declarant les rutes com un component més.

Com que la capçalera és un element persistent en totes les pàgines, s'ha deixat fora del Switch. Només la part del contingut principal canviarà en funció de la ruta.

Exemple de rutes:

```
<Switch>
  <Route exact path="/login">
    <Login />
  </Route>
  <Route exact path="/register">
    <Register />
  </Route>
  <Route path="/groups">
    <EnsureLoggedIn>
      <GroupsRoutes />
    </EnsureLoggedIn>
  </Route>
  { /* Altres rutes definides aquí */ }
  <Route path="*">
    <NotFoundPage />
  </Route>
</Switch>
```

Taula 13 Fragment del fitxer /pages/MainRoutes.js

Pel que fa a la declaració de les rutes, s'han anat agrupant segons el nivell de profunditat d'aquestes en diferents fitxers, posant per exemple les de primer nivell dins de /pages/MainRoutes.js i les de sota de "/courses" al fitxer de codi /pages/CourseRoutes.

Després dins de la ruta en sí React-Router permet recuperar dades sobre aquesta, per exemple, l'identificador concret dins una ruta de l'estil "/group/:groupId". Es pot fer amb hooks o higher-order components (HOCs) – un concepte de React que seria com un decorador de components.

9.2.4 Gestió de l'estat i comunicació amb l'API REST

Per a la comunicació amb l'API REST i la gestió de l'estat de l'aplicació no he fet servir cap llibreria addicional com Redux [25] o RxJS [26]. Volia primer aprendre com seria la implementació fent servir purament React i al final vaig continuar treballant d'aquesta manera, sense trobar el moment oportú per revisar l'estructura global de l'aplicació del client.

La comunicació amb l'API REST l'he fet utilitzant el mètode natiu del navegador `window.fetch`, dins d'uns mètodes per facilitar les crides de GET, POST, PATCH i altres:

```
function requestCourseYear() {
  Api.get(`/courses/years/${courseYearId}`)
    .then(data => setCourseYear(data))
    .catch(error => setHasError(true))
    .finally(() => setIsLoading(false))
}
```

Taula 14 Exemple de crida a l'API REST per obtenir les dades del Course Year

Les dades s'obtenen amb un GET anant a l'API REST utilitzant el hook `useEffect` i es guarden dins l'estat del component de la pàgina que mostra i treballa amb aquestes. Si els seus components fills modifiquen les entitats relacionades amb crides a l'API REST, notifiquen el pare cridant un mètode de callback. En resum, aquesta implementació es basa en la tècnica d'aixecar l'estat cap amunt per tal que l'avantpassat comú el gestioni. El pare llavors torna a demanar les dades amb un GET a l'API REST per mostrar la nova versió de l'entitat.

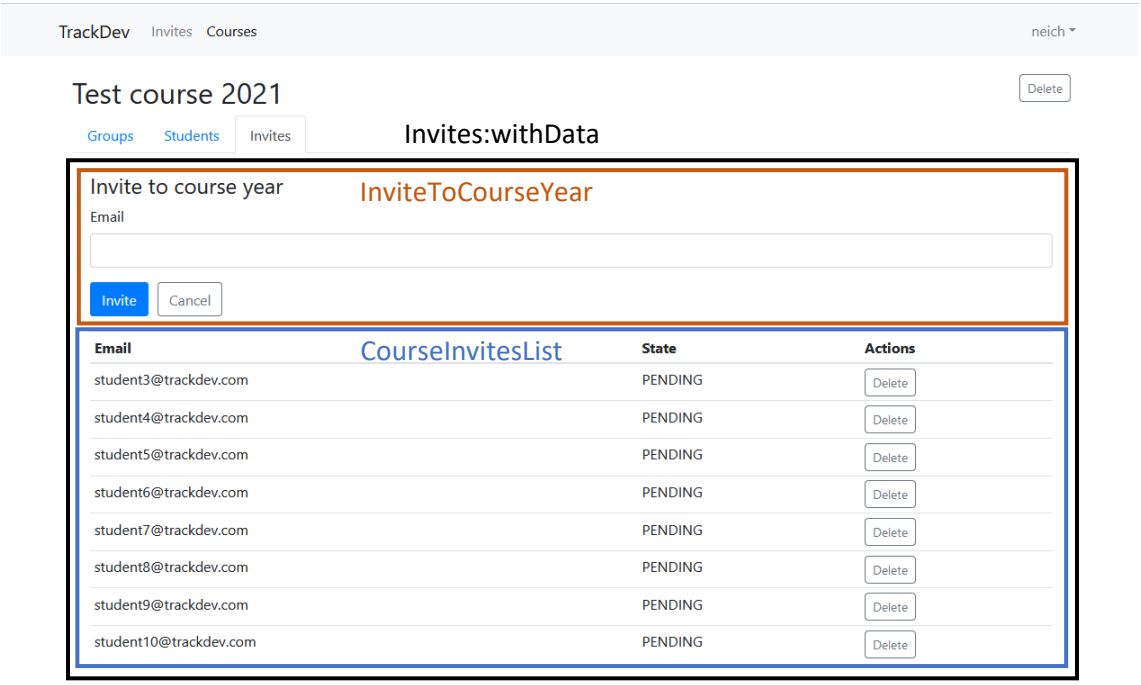
En alguns casos, per components que només visualitzen les dades d'una sola crida API i no necessiten gestionar les dades de diferents crides de l'API, he implementat un component d'ordre superior de React (HOCs) per a compartir la lògica de obtenir dades a l'API REST i guardar-ne l'estat, sigui les dades retornades, l'error en cas d'error i l'estat de carregant. A continuació es mostra un exemple on s'utilitza:

```
const TaskTimelineWithData = withData(
  TaskTimeline,
  'changes',
  (props) => Api.get(`/tasks/${props.taskId}/history`)
)
```

Taula 15 Exemple d'utilització del HOC withData

La nomenclatura està inspirada en els exemples de la documentació de React oficial i també en un HOC de la llibreria Redux.

A continuació explico en detall aquest flux amb els components implementats per mostrar i gestionar la llista d'invitacions d'un any de l'assignatura. La relació entre els components de React i la interfície seria la següent:



La seqüència de passos que passen quan s'obre la pestanya d'invitacions, es clica el botó de convidar i s'envia el formulari serien:

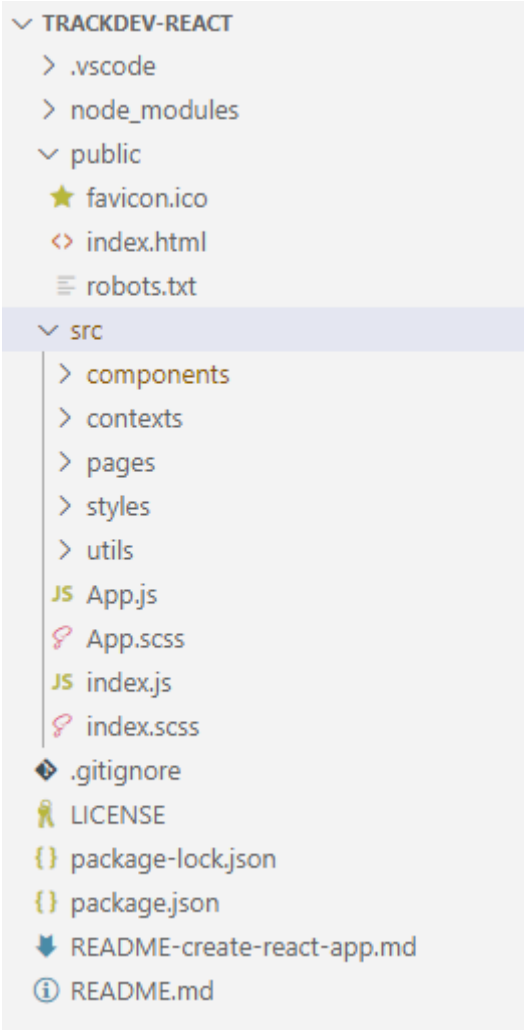
- React crida el mètode `componentDidMount` de `Invites:withData` just després de la primera renderització. El component `withData` està implementat com a Class Component, altrament React cridaria la funció dins del `useEffect`. Dins d'aquest mètode `componentDidMount` es fa la crida GET a l'API cap a la ruta `'/invites?type=courseYearId&courseYearId={courseYearId}'`.
- Quan l'API torna la resposta amb la llista d'invitacions, el component `Invites:withData` modifica el seu estat per guardar la resposta. Com que es modifica l'estat, React re-renderitzarà el component.
- React crida el mètode `render` del component `Invites:withData`.
- React crida els mètodes `render` dels components fills `InviteToCourseYear` i `CourseInvitesList` amb l'identificador de l'any de l'assignatura i la llista d'invitacions com a props.
- React actualitza el DOM amb la descripció de la interfície d'usuari tornada pels mètodes render.
- L'usuari clica el botó de convidar i entra el correu electrònic del nou membre de l'any de l'assignatura. `InviteToCourseYear` modifica el seu estat per gestionar l'estat local del formulari – obert un cop clicat el primer botó i el valor del camp de text quan s'entra. React haurà renderitzat el component després de cada canvi amb les actualitzacions del DOM corresponents.
- L'usuari envia el formulari per crear la invitació.
- React crida la funció de `handleSubmit` de `InviteToCourseYear`
- Dins de la funció `handleSubmit` del component `InviteToCourseYear`, es fa una crida POST a l'API amb el nou correu electrònic.
- La crida a l'API pot fer-se amb èxit o pot ocórrer un error:
    - Si ha tingut èxit:

        - `InviteToCourseYear` crida la funció `onDataChanged` que el component pare `Invites:withData` li ha passat per a que executi si les dades canvien.
        - Dins la funció `onDataChanged`, el component `Invites:withData` torna a fer el GET a l'API per a actualitzar la llista d'invitacions. Un cop arriba, modifica l'estat local amb les noves dades. Com que es modifica l'estat, React re-renderitzarà el component.
        - React crida el mètode `render` del component `Invites:withData`.
        - React crida els mètodes `render` dels components fills `InviteToCourseYear` i `CourseInvitesList` amb la nova llista com a props.
        - React actualitza el DOM amb els nous canvis.
    - Si no ha tingut èxit:
        - `InviteToCourseYear` guarda en el seu estat local l'error que ha ocorregut. Com que es modifica l'estat, React re-renderitzarà el component.
        - React crida el mètode `render` del component `InviteToCourseYear`, el qual dirà en la descripció de la interfície d'usuari que es mostra l'error de l'API.
        - React actualitza el DOM amb els nous canvis

Per explicar millor els passos, dic conceptualment que React crida el mètode `render` dels components fills `InviteToCourseYear` i `CourseInvitesList`, però com que en realitat són Function Components seria més correcte dir que executa els components en sí, que tornen la descripció de la interfície d'usuari utilitzant JSX.

9.2.5 Organització del codi

React no força ni recomana cap estructura específica de carpetes per organitzar els fitxers de codi. Com que he fet servir Create React App, sí que hi ha alguna restricció. Create React App et crea dues carpetes: `public` i `src`. A la carpeta `public` hi ha els fitxers estàtics que se serveixen amb el servidor: el fitxer index.html més les imatges pels logos. A la carpeta `src` hi ha els fitxers de codi font que després es processaran amb les tasques de compilació. Ja hi ha alguns fitxers per defecte: `App.js`, `App.css`, `index.js`, `index.css` i alguns altres de seguiment del rendiment i testos que he tret. Els únics fitxers obligatoris per Create React App són `public/index.html` i `src/index.js`.



Il·lustració 23 Captura de pantalla de l'estructura de carpetes del codi

Els fitxers de codi de Javascript els he organitzat per tipus en diferents carpetes:

- **components:** La majoria de components de React viuen aquí
- **contexts:** Definició de Contexts de Sitecore. Només hi ha el UserContext.
- **pages:** Components que equivalen a una pàgina / ruta. També conté les declaracions de les rutes. Alguns exemples són: GroupRoutes.js, `MainRoutes.js`, `Task.js` i `Group.js`. Els components que només defineixen rutes acaben en `Routes`, les altres són pàgines.
- **styles:** De moment amb un sol fitxer buit `_custom-boostrap.scss`, pensat per si es vol personalitzar en el futur les variables de Bootstrap. Els estils que s'han implementat viuen juntament amb el component que estilen, escrivint un fitxer `.css` o `.scss` per Component. Les classes de CSS tenen el mateix nom que el component que estilen.
- **utils:** Conté mètodes i classes amb utilitats independents de la interfície. Ara hi ha un sol fitxer `api.js` que conté els mètodes que faciliten les crides API utilitzant `window.fetch`.

Els fitxers de punt d'entrada a l'aplicació són App.js i index.js:

- **App.js:** conté el component pare de tots els components, és el component arrel de tota l'aplicació.
- **index.js:** S'encarrega d'inicialitzar l'aplicació renderitzant el component `App` a un element `<div>` buit de la pàgina `index.html` cridant:

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

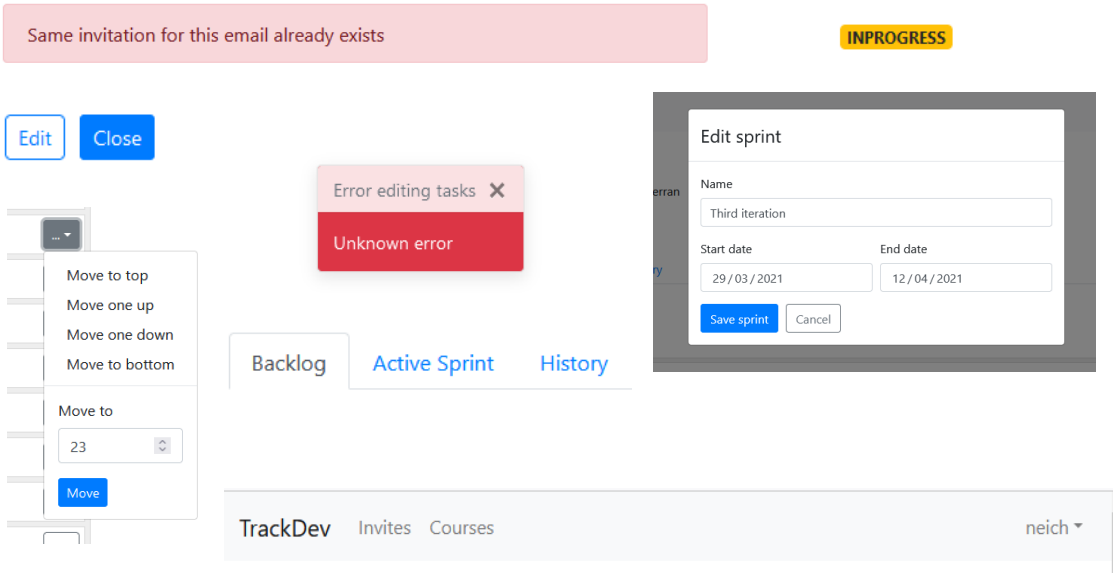
Taula 16 Inicialització de l'aplicació de React amb ReactDOM.render fent servir el mode Strict

El mode Strict evita utilitzar mètodes obsolets de React que s'eliminaran en properes versions.

9.2.6 Interfície usuari

Utilitzant Bootstrap

He utilitzat Bootstrap com a framework CSS per estilar l'aplicació web. Alguns dels components utilitzats són els següents:



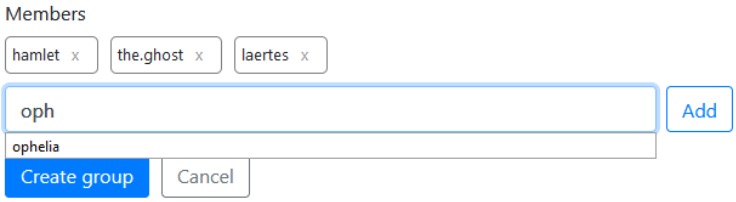
Il·lustració 24 Exemples d'utilitzacions de components de Bootstrap: Alert, Badget, Button, Dropdown, NavBar, Toast, Modal i Tabs

La integració ha sigut senzilla, l'única particularitat ha sigut haver d'instal·lar la llibreria react-router-bootstrap, implementada pel mateix grup de codi obert, per poder continuar utilitzant el component `Link` de react-router i tenir transicions suaus entre diferents pàgines.

Components genèrics implementats

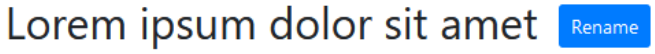
Tot i utilitzar Bootstrap, vaig haver d'implementar algun component purament d'interfície, independent de les crides API que cridaven o si eren de la pàgina de tasca o assignatura.

- **MultiListInput:**



Il·lustració 25 MultiListInput utilitzat com a camp dins el formulari de crear grup

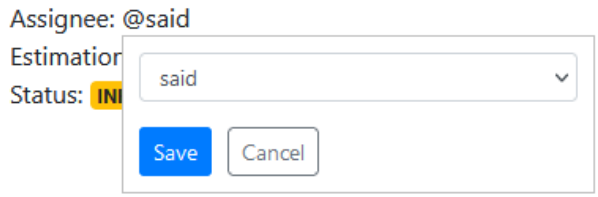
- **EditableHeader:**



Il·lustració 26 Títol de pàgina d'una tasca amb el ratolí a sobre

Il·lustració 27 Títol de pàgina d'una tasca en mode edició

- **EditableField:**



Il·lustració 28 Edició de la persona al qual està assignada la tasca amb el component EditableField

Utilitzant React-beautiful-dnd

React-beautiful-dnd ofereix els components `DragDropContext`, `Droppable` i `Draggable` per a implementar llistes ordenables amb efecte Drag and Drop. Els elements d'una llista es poden moure d'un a l'altre. La integració ha sigut fàcil seguint una documentació ben escrita i un vídeo tutorial [27] del autor principal de la llibreria.

La particularitat aquí ha sigut l'estratègia per actualitzar l'estat local. Seguint la recomanació de l'exemple del tutorial, s'ha actualitzat primer l'estat local del canvi d'ordenació o d'una llista a l'altra, sent optimistes, i després s'ha fet la petició a l'API REST per guardar el canvi de forma persistent. Si no es fa així, guardant de forma optimista el canvi, un cop s'ha deixat anar l'element arrossegat al seu nou lloc es mostra la llista original previ al arrossegament, ja que l'estat del component no ha canviat.

9.3 Altres

9.3.1 CORS

Al principi del projecte el primer problema que em vaig trobar va ser haver de configurar l'API per poder treballar en local amb diferents dominis per l'API REST i l'aplicació SPA. Les crides a l'API utilitzant el mètode `window.fetch` no anaven bé a causa del Cross-Origin Resource Sharing (CORS) i de la cookie SameSite.

El CORS és un mecanisme basat amb headers HTTP que pot fer servir un servidor per indicar al navegador quins dominis tenen accés als seus recursos, a part d'ell mateix.

Si el servidor no respon amb cap header HTTP de CORS permetent l'accés, el navegador bloqueja les peticions GET fets amb `XMLHttpRequest` o `window.fetch` d'un domini A cap a un domini B. Per les peticions d'escriptura es pregunta abans si es té accés amb una petició de pre-flight amb el mètode OPTIONS. Hi ha alguns tipus de peticions que no requereixen CORS o que no fan pre-flight. Alguns són els elements incrustrats (embedded) com imatges o vídeos, o enviaments de formularis amb POST amb els tipus de Content-Type `application/x-www-form-urlencoded`, `multipart/form-data` o `text/plain`.

En la següent il·lustració es mostra la comunicació entre el navegador i el servidor. Es pot veure que el client envia el header HTTP `Origin` amb el domini del lloc i com el servidor li indica que aquest domini té accés al recurs amb el header HTTP `Access-Control-Allow-Origin`. Addicionalment, el servidor indica que es poden utilitzar les cookies amb el header `Access-Control-Allow-Credentials`.

Il·lustració 29 Exemple d'intercanvi de crides d'una petició GET entre dominis i amb cookies[4]

En el cas d'aquesta aplicació necessitava accés des del domini de l'aplicació client 'http://localhost:3000' cap al domini de l'API REST 'https://localhost:8080' enviant cookies, ja que el token JWT que es fa servir per identificar l'usuari a l'API es guarda en una cookie de sessió.

A nivell d'API Spring permet configurar el CORS. El domini al qual es permet accés amb CORS està configurat dins el fitxer `application.properties` com a configuració:

```
trackdev.cors.allowed-origin=http://localhost:3000
```

Taula 17 Configuració del domini amb accés entre dominis al fitxer de configuracions

Llavors aquesta configuració s'utilitza per configurar el CORS amb Spring, a través de la classe `CorsConfigurations` creada:

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {
```

[4] Il·lustració extreta de "Cross-Origin Resource Sharing (CORS) - HTTP", de MDN contributors [31]

```
    @Autowired
    CorsConfiguration corsConfiguration;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        if(corsConfiguration.isEnabled()) {
            registry.addMapping("/**")
                .allowedOrigins(corsConfiguration.getAllowedOrigin())
                .allowedMethods("GET", "HEAD", "POST", "PUT", "PATCH", "DELETE")

                .allowCredentials(true).maxAge(3600);
        }
    }
}
```

Taula 18 Codi de la classe WebConfig que configura el CORS utilitzant Spring MVC

A nivell de l'aplicació client al navegador s'ha d'indicar al `window.fetch` que enviï les cookies amb la crida utilitzant la opció `credentials`:

```
/**
 *  Does a request with a body to the API.
 *
 * @param {String} method
 * @param {String} relativePath
 * @param {Object} requestBody
 * @returns {Promise} The data response from the API
 */
Api.send = async function(method, relativePath, requestBody) {
  let options = {}
  if(requiresCors) {
    options.credentials = 'include'
  }
  options.method = method
  if(requestBody) {
    options.headers = {
        'Content-Type': 'application/json'
    }
    options.body = JSON.stringify(requestBody)
  }
  const response = await fetch(apiBaseUrl + relativePath, options)
  return getData(response)
}
```

Taula 19 Codi del mètode Api.send que s'utiltiza en tota l'aplicació a través d'Api.get, Api.post i similars.

Després també s'havia de configurar la política de cookies de tercers utilitzant l'atribut SameSite de les cookies enviades pel servidor. La cookie necessària per guardar el token JWT no es

guardava al domini de l'aplicació client amb React perquè el valor per defecte del SameSite si no s'indica és Lax[5]. Els valors possibles per aquest atribut són:

- **Lax:** les cookies no s'envien en peticions entre dominis normals, excepte quan l'usuari navega cap a la web amb un enllaç. És el valor per defecte si no s'especifica.
- **Strict:** les cookies només s'envien en peticions del propi domini, i no en peticions iniciades per altres dominis.
- **None:** les galetes s'enviaran en tots els contexts, tant per peticions del propi domini com de peticions iniciades per webs de tercers.

Per solucionar aquest cas, es configura l'atribut SameSite quan es genera la cookie durant l'inici de sessió. Si l'origen indicat amb el header HTTP Origin coincideix amb el domini habilitat pel CORS, es configura la cookie amb valor SameSite=None, altrament amb SameSite=Lax.

[5] Abans els navegadors enviaven les cookies per totes les peticions. En canvi al llarg del 2020 els navegadors van canviar que el comportament per defecte si no s'indicava el SameSite fos el Lax per tant de protegir millor els usuaris contra alguns tipus d'atacs Cross Site Request Forgery (CSFR)

10 IMPLANTACIÓ I RESULTATS

El projecte no està acabat atès que el seu abast és molt gran. Per aquest fet, no s'ha implantat l'aplicació web a un servidor en mode de producció per ser accessible pel públic general. En futurs projectes de final de grau es pot continuar el desenvolupament de l'aplicació i implantar-la.

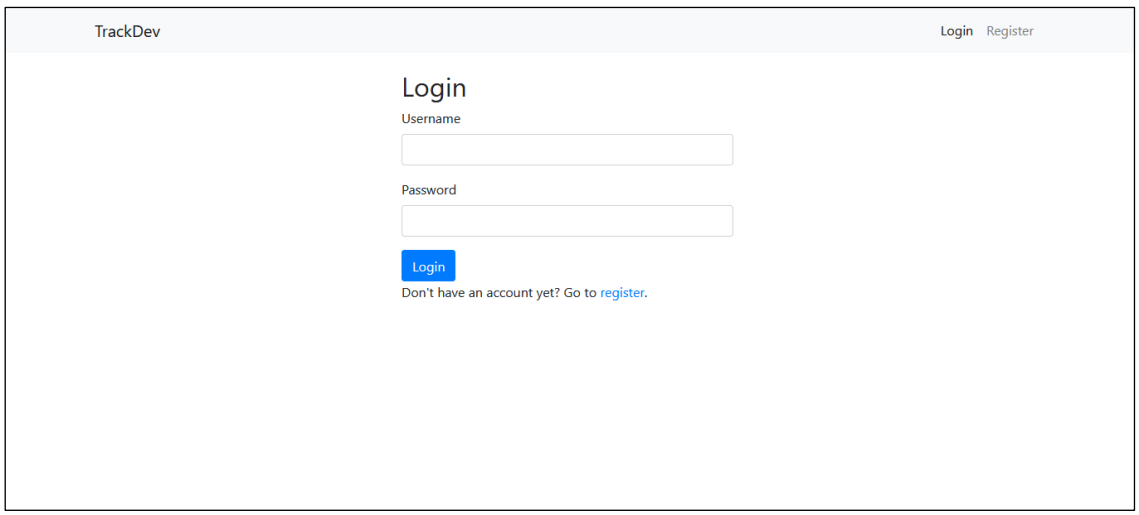
En aquest apartat mostro els resultats del projecte utilitzant captures de pantalla de l'aplicació executada en l'ordinador local.

10.1 GESTIÓ D'ASSIGNATURES, USUARIS I GRUPS

10.1.1 Inici de sessió, registre i invitacions a l'aplicació

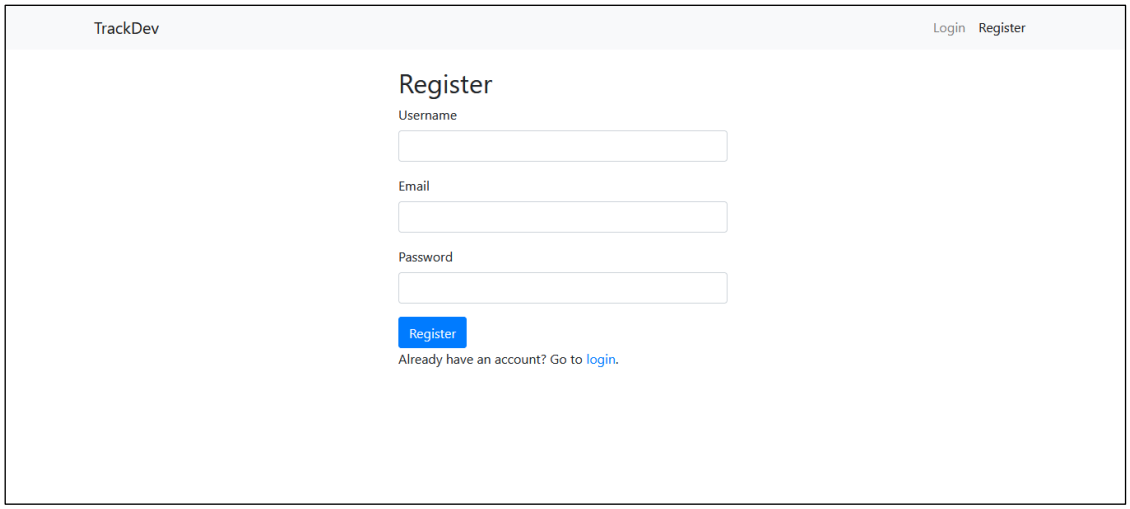
A continuació mostro les pàgines d'inici de sessió i registre. També la pàgina d'inici en els dos casos diferents, per l'usuari anònim i per l'usuari amb la sessió iniciada on se'l saluda amb el clàssic Hello World personalitzat.

Il·lustració 30 Captura de pantalla de la pàgina d'inici per un usuari anònim



Il·lustració 31 Captura de pantalla de la pàgina d'inici de sessió

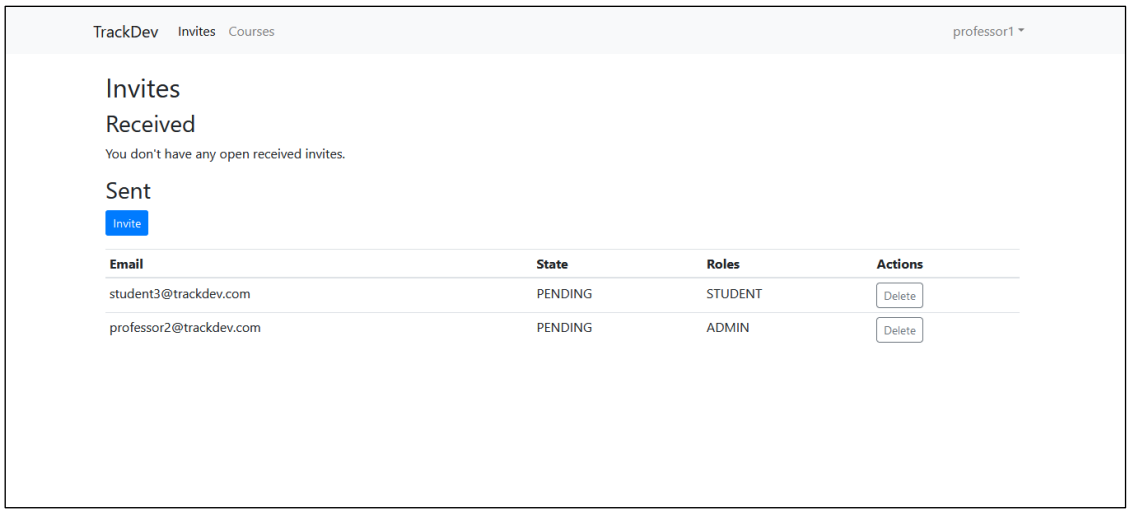
En la següent captura es veu la pàgina de registre en la qual s'ha d'entrar l'usuari, correu electrònic i contrasenya. El registre no és lliure i cal que el correu electrònic tingui una invitació a nivell d'aplicació per un rol – com ara professor o administrador – o a un curs d'una assignatura per un alumne.



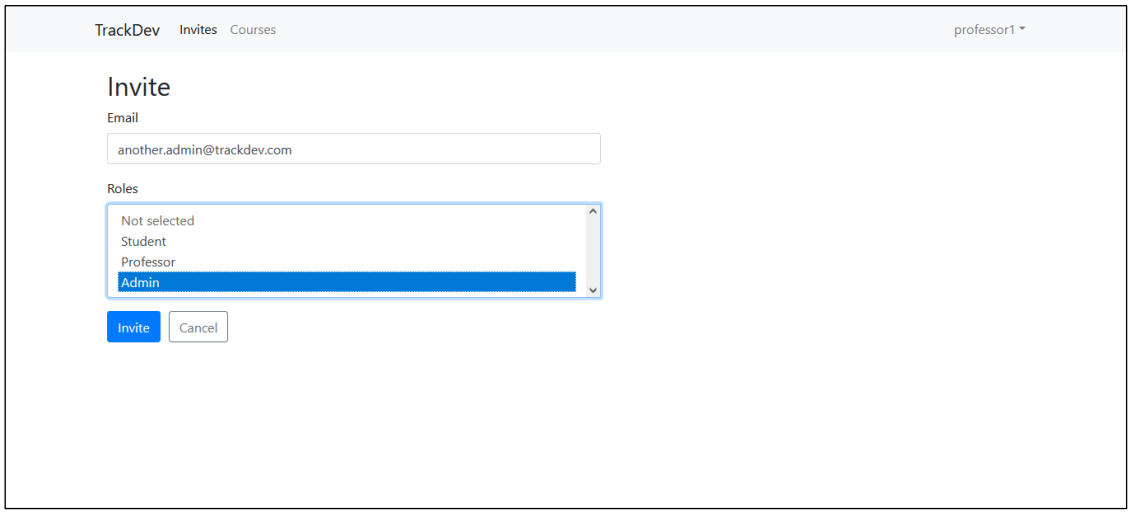
Il·lustració 32 Captura de pantalla de la pàgina de registre. Cal que el correu electrònic tingui una invitació

Il·lustració 33 Captura de pantalla de la pàgina d'inici per un usuari amb sessió iniciada

Dins la pàgina per gestionar les invitacions es poden veure les invitacions rebudes per accedir a recursos i a les invitacions enviades, que es poden cancel·lar mentre no s'hagin acceptat. Des d'aquesta plana es pot convidar a altres usuaris a l'aplicació amb uns rols.



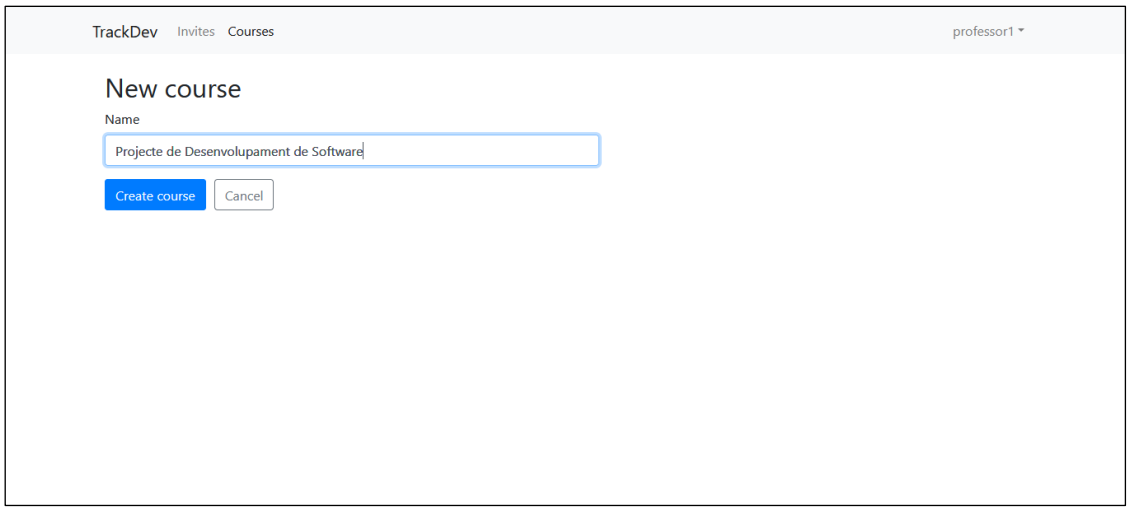
Il·lustració 34 Captura de pantalla de la pàgina per gestionar les invitacions



Il·lustració 35 Captura de pantalla del formulari per convidar un altre usuari a l'aplicació amb uns rols

10.1.2 Gestió d'assignatures, els seus cursos i alumnes

Il·lustració 36 Captura de pantalla de la pàgina amb les assignatures creades pel professor

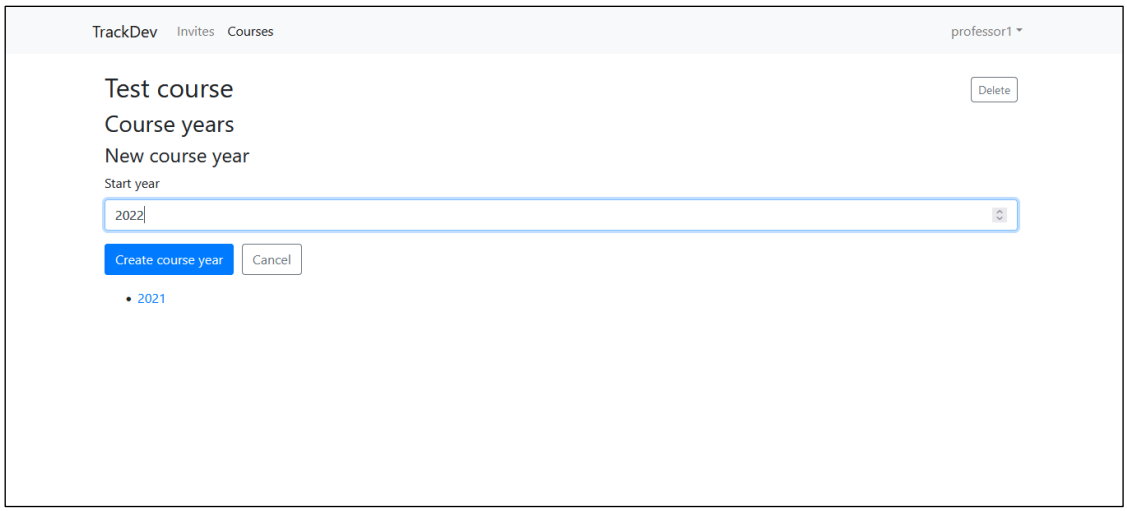


Il·lustració 37 Captura de pantalla de la pàgina per a crear una assignatura nova

En la següent captura es veu la pàgina d'una assignatura. El títol es pot editar dins la mateixa pàgina clicant a sobre i hi ha un botó per eliminar-la. Es llisten els diferents anys de les assignatures per si es guarden per consultar la història de cursos anteriors.

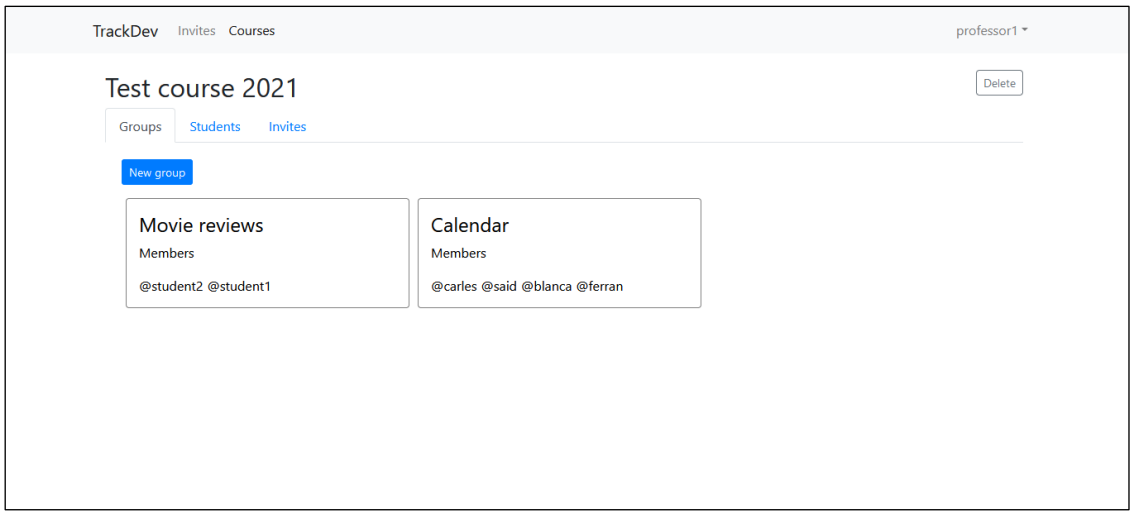
Il·lustració 38 Captura de pantalla de la pàgina d'una assignatura concreta, on es veuen els cursos de cada any

En alguns casos, per formularis molt simples que s'usen per afegir elements nous a una llista, he usat un formulari incrustat sense anar a obrir una pàgina nova. Aquest formulari es deplega quan es clica el botó de crear.

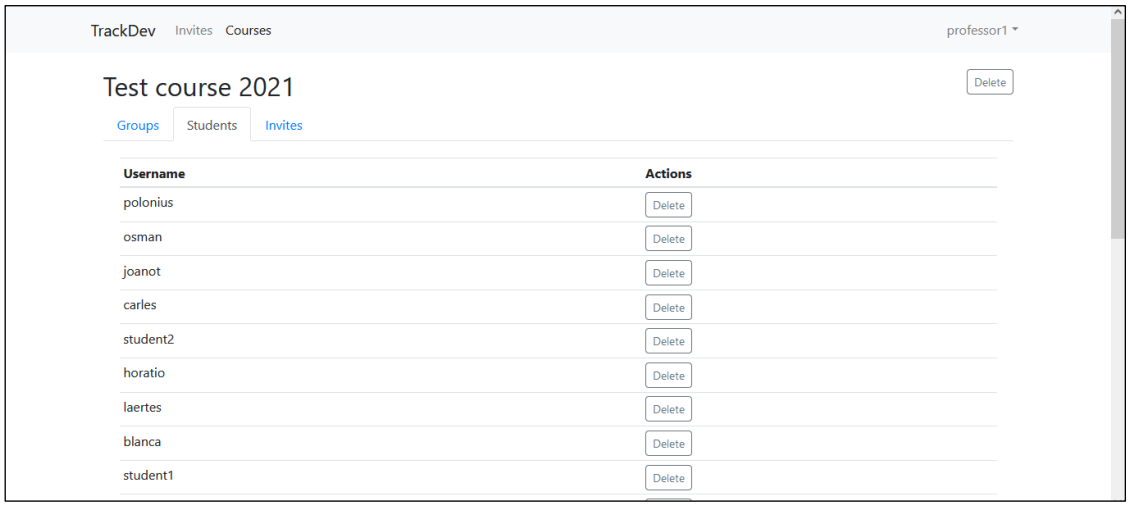


Il·lustració 39 Captura de pantalla de la pàgina d'una assignatura amb el formulari per crear nous cursos desplegat

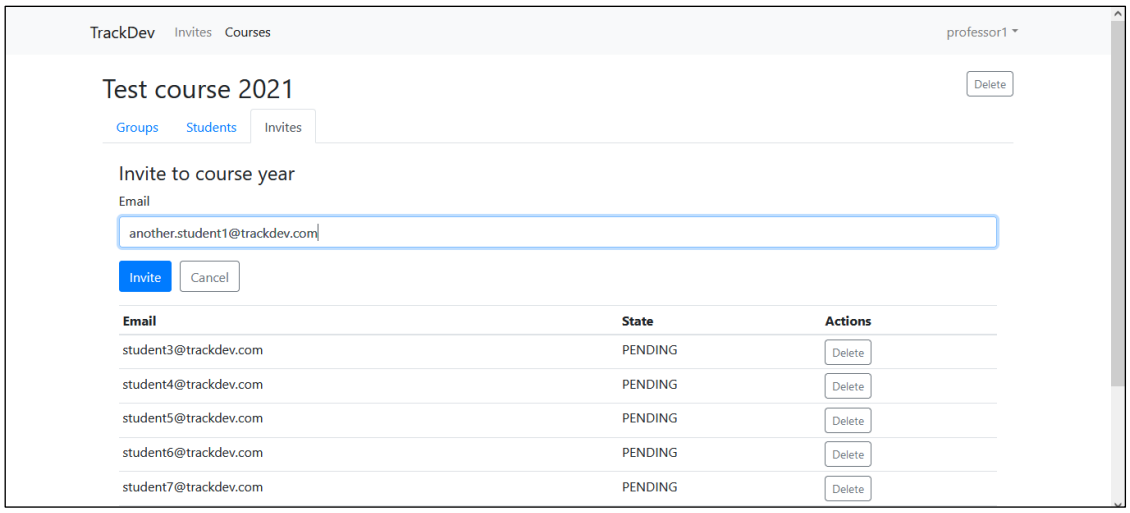
La resta de captures mostren les pàgines més principals, que serien el curs d'una assignatura, amb la gestió dels alumnes registrats, els grups de treball i les invitacions.



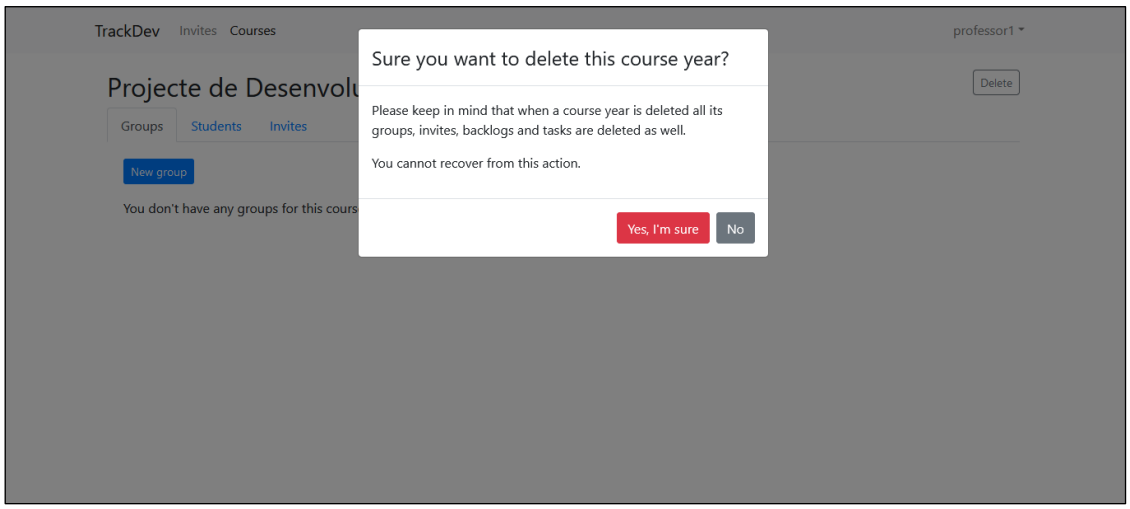
Il·lustració 40 Captura de pantalla d'un curs d'una assignatura amb la pestanya de grups seleccionada per defecte



Il·lustració 41 Captura de pantalla dels alumnes registrats a un curs d'una assignatura dins la pestanya d'alumnes



Il·lustració 42 Captura de pantalla de les invitacions a un curs de l'assignatura amb el formulari de convidar obert

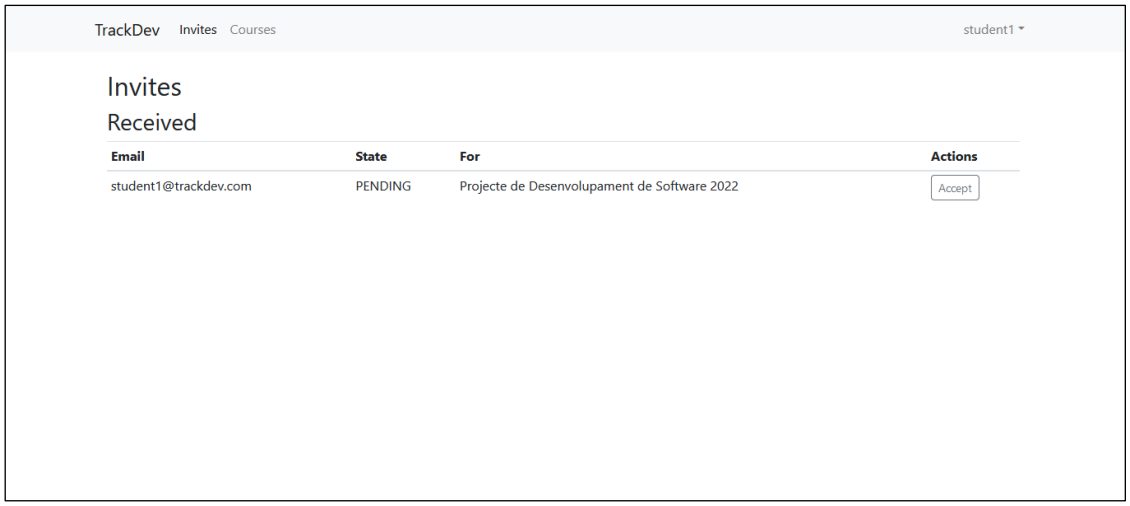


Il·lustració 43 Captura de pantalla del modal mostrat per confirmar l'eliminació d'un curs d'una assignatura

Les següents captures de pantalla mostren el punt de vista per part d'un alumne. L'alumne no té accés a la llista d'assignatures, sinó que només té accés als cursos d'un any concret de les assignatures. Dins la pàgina del curs de l'assignatura només pot veure la pestanya de grups i els grups on és membre. Les invitacions que rebi per cursos nous un cop registrat les pot veure dins la pàgina de invitacions.

Il·lustració 44 Captura de pantalla de la llista de cursos on s'està registrat vist per un alumne

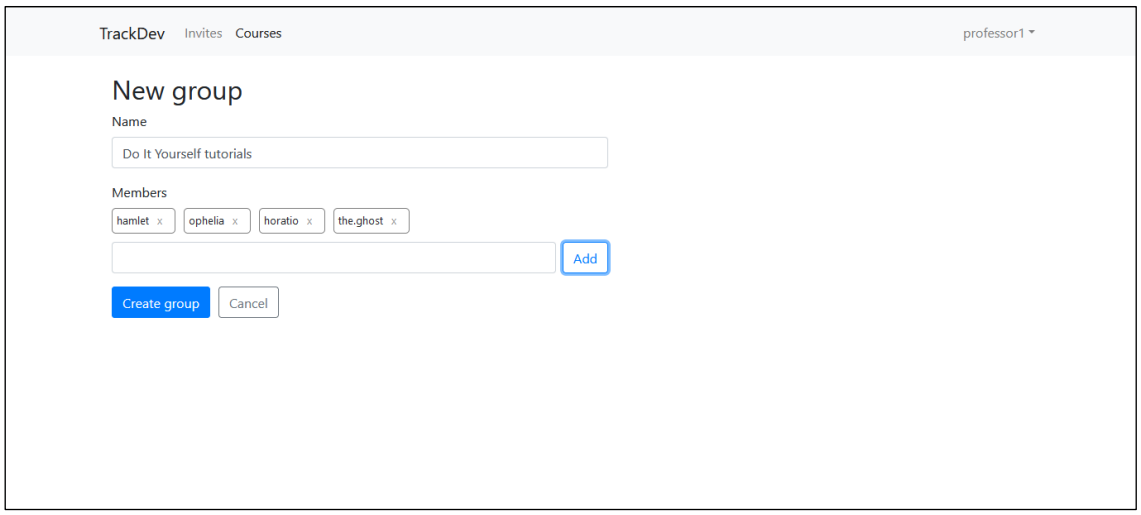
Il·lustració 45 Captura de pantalla de la vista d'un curs d'una assignatura vist per un alumne, que només veu la pestanya grups amb els grups on pertany



Il·lustració 46 Captura de pantalla de les invitacions rebudes a un curs de l'assignatura pendents d'acceptar vist per un alumne

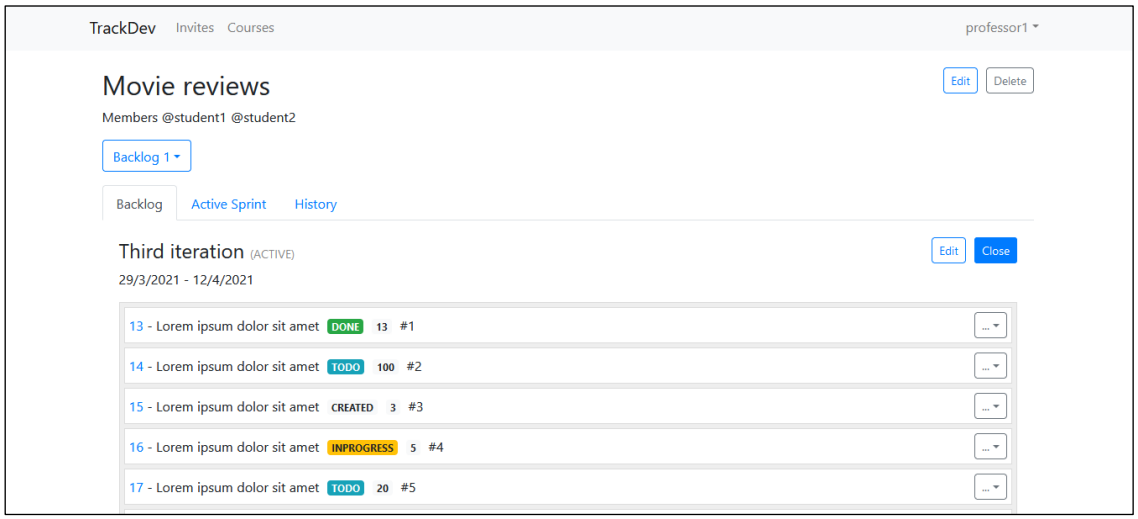
10.1.3 Gestió de grups

Quan es crea un grup de treball ja s'assignen els alumnes que el formen, que s'escullen de la llista d'alumnes registrats.



Il·lustració 47 Captura de pantalla de la pàgina amb el formulari per crear un grup de treball nou

Dins la pàgina del grup de treball es veu com element principal el backlog.



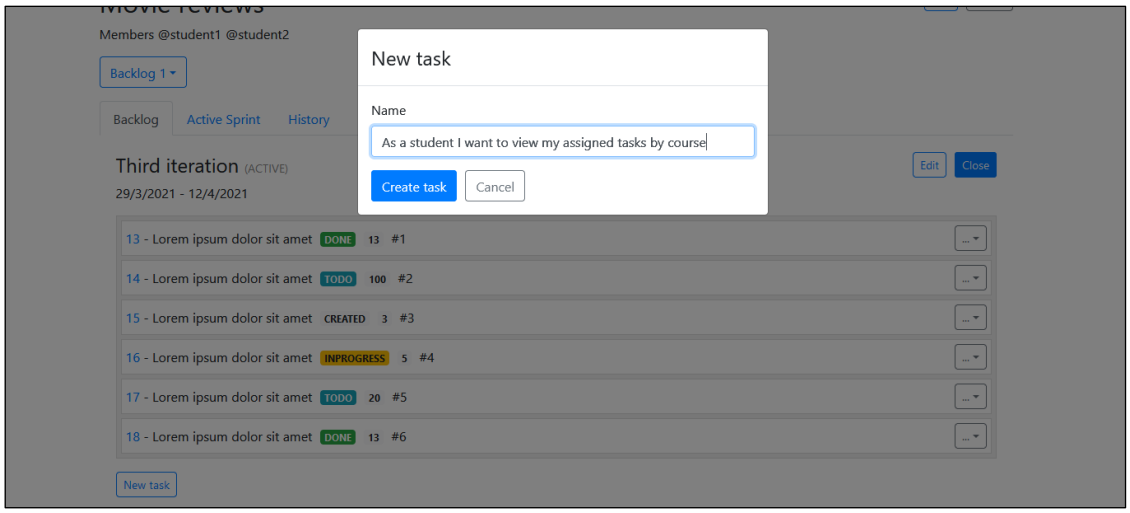
Il·lustració 48 Captura de pantalla de la pàgina d'un grup de treball on es veu el backlog amb un sprint iniciat

10.2 GESTIÓ DE BACKLOGS, ITERACIONS I TASQUES

10.2.1 Gestió de tasques

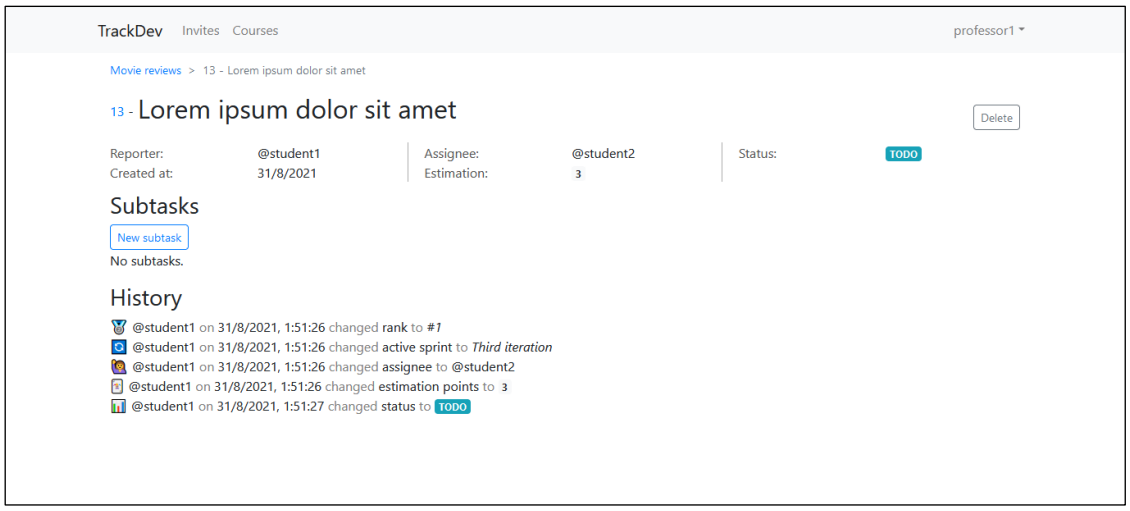
En aquest apartat afegeixo captures de pantalles mostrant la creació de tasques i la pàgina d'una tasca, on es poden crear noves sub-tasques i editar els diferents camps per separat – alumne assignat, nom, estimations i estat.

Dins del backlog he utilitzat modals per la creació de tasques i sprints ja que semblava millor opció que una pàgina a part o un formulari desplegable com en les llistes anteriors.

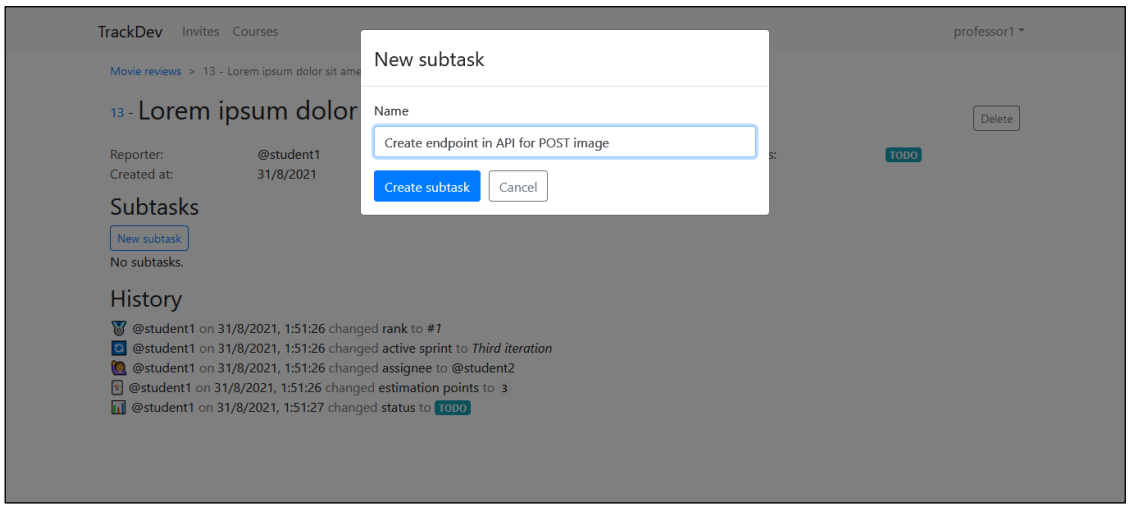


Il·lustració 49 Captura de pantalla del backlog amb el modal per crear una tasca nova

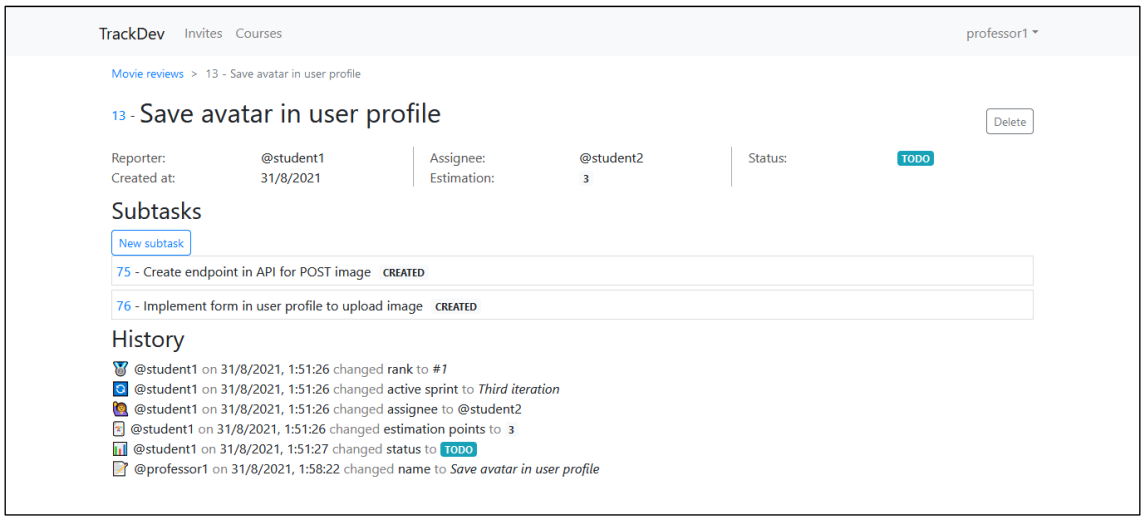
Dins la pàgina d'una tasca es poden diferenciar tres regions: els camps de la tasca, les sub-tasques i la història de canvis fets.



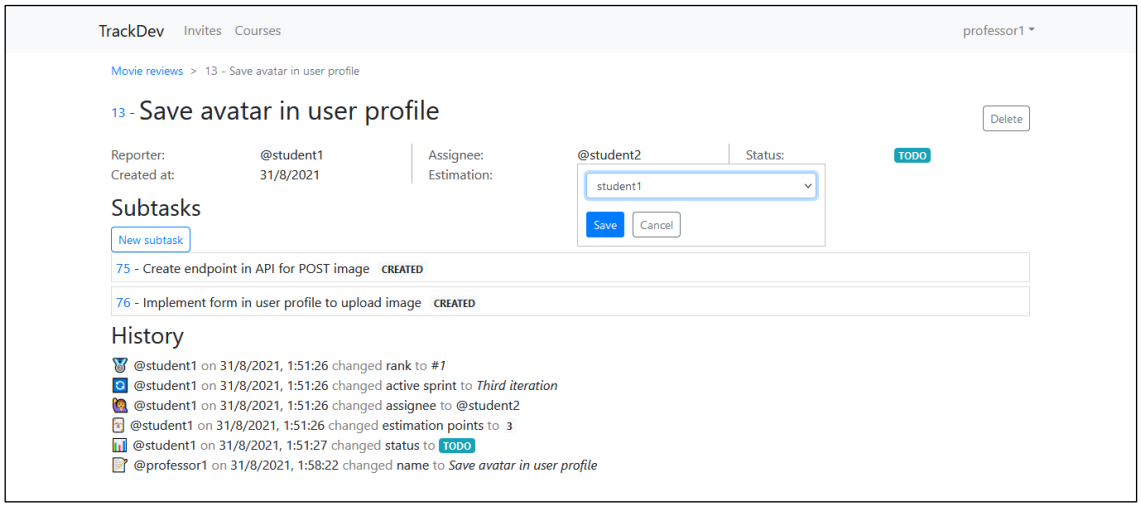
Il·lustració 50 Captura de pantalla de la pàgina d'una tasca



Il·lustració 51 Captura de pantalla mostrant el modal per afegir sub-tasques



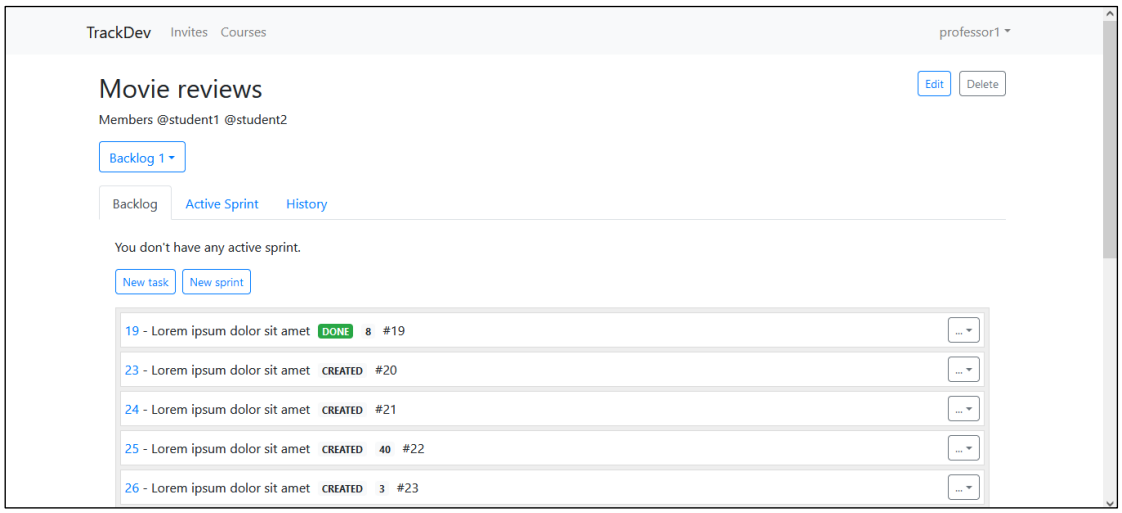
Il·lustració 52 Captura de pantalla mostrant una tasca amb sub-tasques



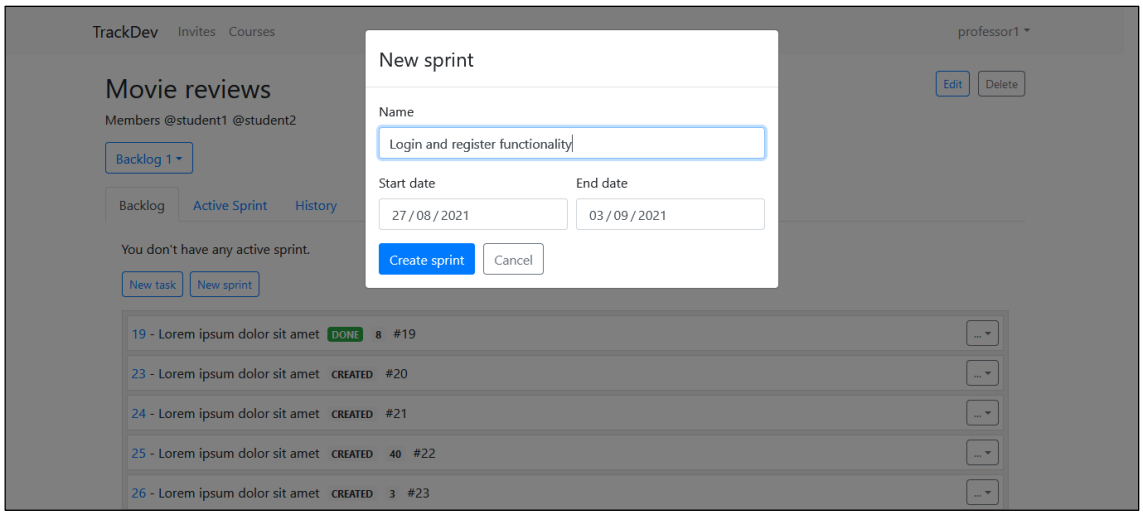
Il·lustració 53 Captura de pantalla mostrant com es canvia l'usuari assignat per la tasca

10.2.2 Gestió de sprints

A continuació mostro els diferents passos de la creació i planificació d'un sprint.

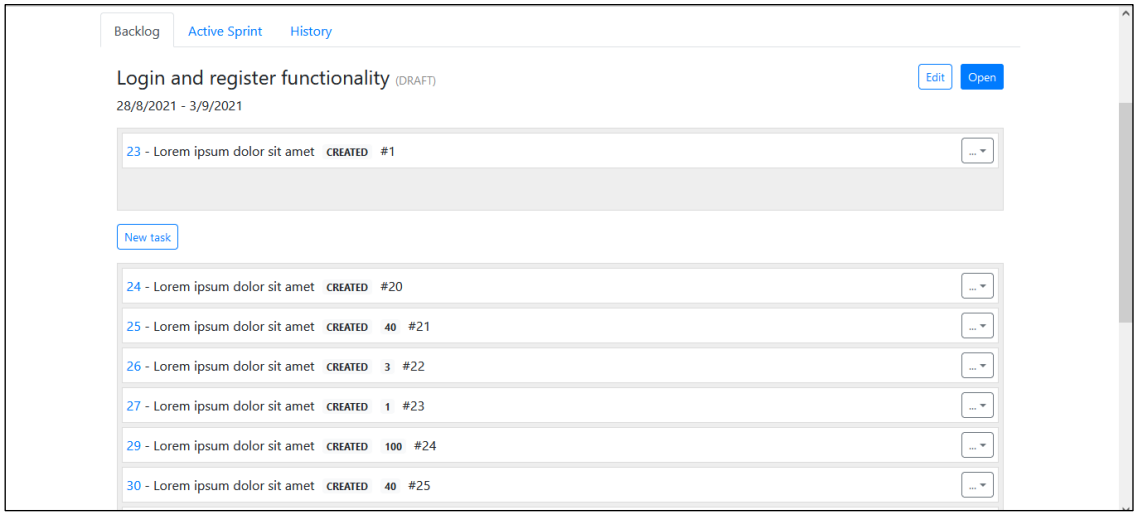


Il·lustració 54 Captura de pantalla del backlog sense cap sprint actiu



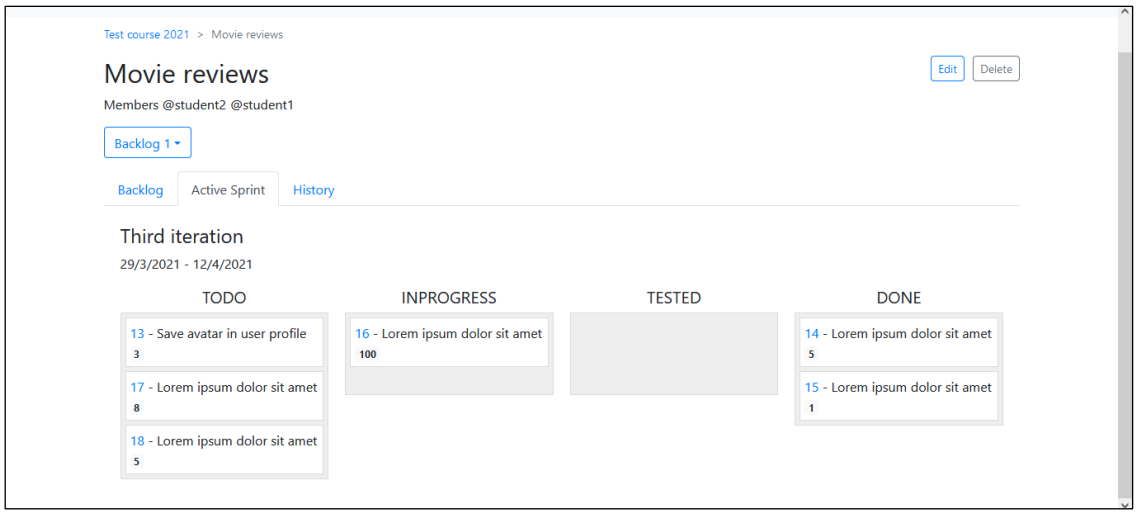
Il·lustració 55 Captura de pantalla del modal per crear un sprint nou

Les tasques es poden afegir i treure de l'sprint arrossegant-les.



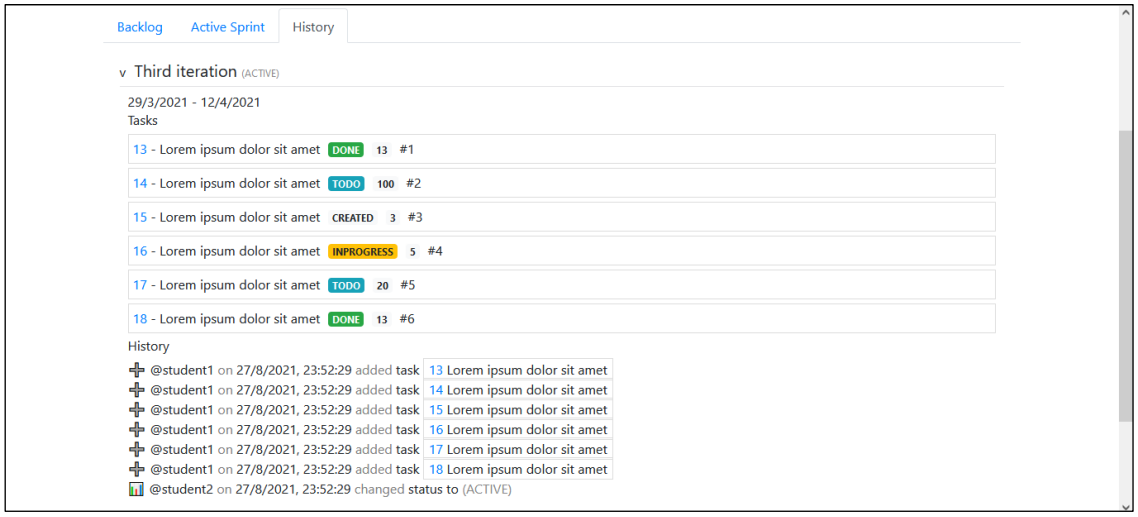
Il·lustració 56 Captura de pantalla durant l'sprint planning en el que s'afegeixen les tasques a fer el proper sprint

La pestanya "Active Sprint" mostra les tasques de l'sprint en columnes segons el seu estat. És un format molt utilitzat per a visualitzar a simple vista l'estat de les tasques i la feina pendent a fer. Les tasques es poden moure d'una columna a l'altre arrossegant-les.



Il·lustració 57 Captura de pantalla de les tasques de l'sprint actiu en columnes segons el seu estat

De la mateixa que es pot visualitzar la història de canvis d'una tasca, es pot visualitzar dins de la pestanya "History" tots els sprints passats del backlog. Per cada sprint es pot veure les tasques que es varen acabar dins d'aquest, els canvis d'estat o de nom i les tasques que es treuen o s'afegeixen.



Il·lustració 58 Captura de pantalla de la pestanya de la història dels sprints del backlog

11 CONCLUSIONS

L'objectiu del projecte era desenvolupar una aplicació web per al control de projectes de software dins una aula amb metodologia àgil. Les principals funcionalitats de l'aplicació desenvolupada són:

- La gestió d'assignatures, grups d'alumnes i usuaris
- La gestió de backlogs i les seves tasques
- La creació i gestió de sprints

Aquesta aplicació s'ha implementat amb una API REST i una SPA amb React.

Com a desviació principal dels requeriments inicials és que no he implementat la funcionalitat d'introduir avaluacions individualitzades. Els motius són dos, es va acordar amb el professor deixar més estable la part de funcionalitats bàsiques de gestió del backlog i tasques; i també per sub-estimacions del temps necessari per implementar cada bloc.

Aquest projecte m'ha permès per una part aprendre a fer una SPA amb React. He treballat amb pàgines web clàssiques renderitzades a servidor fent algunes parts dinàmiques amb AJAX, però no havia implementat mai tota l'aplicació amb totes les pàgines renderitzades a la part de client com es fa en una SPA. He hagut d'aprendre a utilitzar React i també familiaritzar-me de nou amb tot l'entorn de Java. Per un altre costat, he fet la planificació de les diferents tasques del projecte, prenent decisions sobre què prioritzar.

12 TREBALL FUTUR

Si comparo l'aplicació desenvolupada amb altres eines del mercat com Jira o YouTrack, que he utilitzat com a referència, hi ha un conjunt de funcionalitats de la part de gestió de backlogs, tasques i sprint que seria interessant implementar:

- Registre de les hores treballades en cada tasca (el worklog).
- Integració automàtica amb GitHub dels pull requests i tiquets que es creen en els repositoris on els alumnes allotgen la part client i servidor del seu projecte. Això facilitaria al professor l'avaluació de les tasques i el seu codi.
- Cerca de tasques amb filtres, on també es poguessin veure les tasques tancades.
- Creació de dashboards personalitzats en el quals es pugui tenir una visió global de l'activitat d'una assignatura o d'un grup de treball concret. Per exemple, veure tots els canvis d'assignacions fets.

Pel que fa a la part distintiva que vol aportar aquesta aplicació, ajudar a avaluar el procés d'aprenentatge que fan els alumnes en la gestió d'un projecte amb metodologia àgil dins l'aula, els treballs futurs a realitzar poden ser:

- Afegir comentaris i avaluacions de part del professor a les tasques i accions que facin els alumnes.
- Facilitar la importació de les assignatures i alumnes. Es podria fer utilitzant fitxers en format .csv o bé suportant el protocol Learning Tools Interoperability (LTI) [28] proposat pel IMS Global Learning Consortium per a la comunicació entre aplicacions d'aprenentatge. Moodle suporta la integració amb altres eines amb aquesta especificació [29].

A nivell més general, es podrien fer les següents millores:

- Delegar el procés de login a un servei extern, que podria ser la Universitat de Girona o GitHub per tal de no guardar les contrasenyes. Altrament s'hauria d'implementar funcionalitats que falten: recuperació de contrasenya, establir un màxim d'intents fallits i monitoreig de l'activitat relacionada amb l'inici de sessió.
- Enviar un correu electrònic quan s'invita a un altre usuari amb un enllaç de registre de tal manera que l'usuari convidat no hagi d'escriure manualment el correu electrònic al formulari i fer servir aquest pas com a verificació del correu electrònic.
- Organitzar millor el codi de la SPA de forma que el flux de les dades i la comunicació amb l'API REST fos més intuïtiva.

13 BIBLIOGRAFIA

[1] Atlassian, «Jira - Issue & Project Tracking Software» [En línia]. Disponible: https://www.atlassian.com/software/jira.

[2] JetBrains s.r.o., «YouTrack: The project management tool designed for agile teams» [En línia]. Disponible: https://www.jetbrains.com/youtrack/.

[3] Facebook Inc., «React – A JavaScript library for building user interfaces» [En línia]. Disponible: https://reactjs.org.

[4] «trackdevel/trackdev-spring: Spring backend for the trackdev project» [En línia]. Disponible: https://github.com/trackdevel/trackdev-spring.

[5] «GitHub: Where the world builds software» [En línia]. Disponible: https://github.com/.

[6] VMWare, Inc. or its affiliates, «Spring» [En línia]. Disponible: https://spring.io/.

[7] V. Mihalcea, S. Ebersole, A. Boriero, G. Morling, G. Badner, G. Badner, C. Cranford, E. Benard, S. Grinovero, B. Meyer, H. Ferentschik, G. King, C. Bauer, M. R. Anderse, K. Maesen, R. Vansa i L. & Jacomet, «Hibernate ORM 5.5.7. Final User Guide» [En línia]. Disponible: https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html.

[8] «FasterXML/jackson: Main Portal page for the Jackson project» [En línia]. Disponible: https://github.com/FasterXML/jackson.

[9] Facebook Inc., «Create React App,» [En línia]. Disponible: https://create-react-app.dev/.

[10] «Babel - The compiler for next generation JavaScript» [En línia]. Disponible: https://babeljs.io/.

[11] «webpack - bundle your assets» [En línia]. Disponible: https://webpack.js.org/.

[12] ReactTraining, «React Router: Declarative Routing for React.js» [En línia]. Disponible: https://reactrouter.com/web/guides/quick-start.

[13] Bootstrap team and contributors, «Boostrap - The most popular HTML, CSS, and JS library in the world.» [En línia]. Disponible: https://getbootstrap.com/.

[14] React Bootstrap team and contributors, «React-Bootstrap - React-Bootstrap Documentation» [En línia]. Disponible: https://react-bootstrap.github.io/.

[15] «reactstrap - React Bootstrap 4 components» [En línia]. Disponible: https://reactstrap.github.io/.

[16] «atlassian/react-beautiful-dnd: Beautiful and accessible drag and drop for lists with React» [En línia]. Disponible: https://github.com/atlassian/react-beautiful-dnd.

[17] JetBrains s.r.o., «IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains» [En línia]. Disponible: https://www.jetbrains.com/idea/.

[18] Microsoft, «Visual Studio Code» [En línia]. Disponible: https://code.visualstudio.com/.

[19] GitHub, «Atom: A hackable text editor for the 21st century» [En línia]. Disponible: https://atom.io/.

[20] Oracle Corporation and/or its affiliates, «MySQL Workbench Manual» [En línia]. Disponible: https://dev.mysql.com/doc/workbench/en/.

[21] GitHub, «GitHub Desktop» [En línia]. Disponible: https://desktop.github.com/.

[22] Postman, Inc., «Postman API Platform» [En línia]. Disponible: https://www.postman.com/.

[23] JGraph Ltd, «Diagram Software and Flowchart Maker» [En línia]. Disponible: https://www.diagrams.net/.

[24] «Inkscape: Draw freely» [En línia]. Disponible: https://inkscape.org/es/.

[25] D. Abramov i autors, «Redux - A Predictable State Container for JS Apps» [En línia]. Disponible: https://redux.js.org/.

[26] «RxJS» [En línia]. Disponible: https://rxjs.dev/.

[27] A. Reardon, «Beautiful and Accessible Drag and Drop with react-beautiful-dnd» 2021. [En línia]. Disponible: https://egghead.io/courses/beautiful-and-accessible-drag-and-drop-with-react-beautiful-dnd.

[28] IMS Global Learning Consortium Inc., «Learning Tools Interoperability» [En línia]. Disponible: https://www.imsglobal.org/activity/learning-tools-interoperability.

[29] Moodle, «LTI and Moodle» 7 Agost 2017. [En línia]. Disponible: https://docs.moodle.org/311/en/LTI_and_Moodle.

[30] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Pearson Education, Inc., 2003.

[31] MDN contributors, «Cross-Origin Resource Sharing (CORS) - HTTP» 17 Agost 2021. [En línia]. Disponible: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[32] MDN contributors, «Same-origin policy - Web security» 4 Juny 2021. [En línia]. Disponible: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.

[33] MDN contributors, «SameSite cookies» 13 Agost 2021. [En línia]. Disponible: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite.

[34] Oracle and/or its affiliates, «Overview (Java(TM) EE 8 Specification APIs)» [En línia]. Disponible: https://javaee.github.io/javaee-spec/javadocs/.

[35] Oracle and/or its affiliates, «Overview (Java SE 16 & JDK 16)» [En línia]. Disponible: https://docs.oracle.com/en/java/javase/16/docs/api/index.html.

[36] Oracle Corporation and/or its affiliates, «MySQL :: MySQL 8.0 Reference Manual» [En línia]. Disponible: https://dev.mysql.com/doc/refman/8.0/en/.

[37] B. Holt, «Complete Intro to React, v6» 4 Maig 2021. [En línia]. Disponible: https://frontendmasters.com/courses/complete-react-v6/.

[38] Col·laboradors, «RESTful web API design» 1 Desembre 2018. [En línia]. Disponible: https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design.

[39] Baeldung, «Baeldung» [En línia]. Disponible: https://www.baeldung.com/.

[40] E. Gamma, R. Helm, R. Johnson i J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[41] S. Krug, *Don't Make Me Think, Revisited. A Common Sense Approach to Web Usability*, New Riders, 2014.

[42] E. A. Scott, *SPA Design and Architecture*, Manning, 2015.

[43] I. Martin, «Multitier architectures: Spring» [Apunts acadèmics].

[44] SmartBear Software, «Best Practices in API Design» [En línia]. Disponible: https://swagger.io/resources/articles/best-practices-in-api-design/.

[45] B. Holt, «Intermediate React, v3» 4 Maig 2021. [En línia]. Disponible: https://frontendmasters.com/courses/intermediate-react-v3/.

[46] Refsnes Data, «W3Schools Online Web Tutorials» [En línia]. Disponible: https://www.w3schools.com/.