Q-Tasks

Gestor de tasques per al treball en equip

Pau Compte Andreś
Enginyeria Tècnica Informàtica de Sistemes
Aplicacions Web per al Treball Col·laboratiu

Consultor: **Ferran Prados Carrasco**
Responsable de l'assignatura: **Atanasi Daradoumis Haralabus**

12 de juny de 2016

FITXA DEL TREBALL FINAL

Títol del treball:	*Q-Tasks. Gestió de tasques per al treball en equip*
Nom de l'autor:	*Pau Compte Andrés*
Nom del consultor/a:	*Ferran Prados Carrasco*
Nom del PRA:	*Atanasi Daradoumis Haralabus*
Data de lliurament (mm/aaaa):	*06/2016*
Titulació o programa:	*Enginyeria Tècnica Informàtica de Sistemes*
Àrea del Treball Final:	*Aplicacions Web per al Treball Col·laboratiu*
Idioma del treball:	*Català*
Paraules clau	*Tasques, JSONApi, SPA*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

L'aplicació implementada és un gestor de tasques que permet als integrants d'un equip gestionar la seva feina en el dia a dia, centrat sobretot en facilitar la visualització de la relació d'una tasca amb altres tasques assignades als seus companys.

Des del punt de vista acadèmic l'objectiu principal del projecte ha sigut experimentar la construcció d'aplicacions javascript que s'executen en el navegador de l'usuari, i que per a la persistència de dades es comuniquen amb una API REST usant JSON.

Per al desenvolupament de l'aplicació client en Javascript s'ha usat EmberJS i l'API s'ha implementat amb Ruby-on-Rails.

Abstract (in English, 250 words or less):

The implemented application is a task manager that allows all members from a team to manage their daily work, with a special focus in easing the visualization of a task and its relation to other tasks assigned to other team members.

From an academic point of view, the main purpose of the project was experimenting with building Javascript-powered applications that run in the user's browser, and that communicate with a REST API for their data persistance by using JSON.

EmberJS was used for the development of the Javascript client application, and Ruby on Rails was used to create the API.

Índex de continguts

1 - Introducció....3
1.1 - El resultat....3
1.2 - Objectiu principal....4
1.3 - Objectius funcionals i tècnics: Q-Tasks....4
2 - Planificació....6
2.1 - Desviaments i ajustos en la planificació....6
3 - Especificació i anàlisi....7
3.1 - Descripció del problema i l'entorn....7
3.2 - Requeriments funcionals....8
3.3 - Requeriments no funcionals....8
3.3.1 - Seguretat....8
3.3.2 - Rendiment, disponibilitat i accessibilitat....9
3.3.3 - Adaptabilitat....9
3.4 - Casos d'ús....9
3.4.1 - Actors....9
3.4.2 - Diagrama de casos d'ús....10
3.4.3 - Especificació textual dels casos d'ús....10
3.5 - Diagrama estàtic de classes....13
3.6 - Disseny de la persistència....13
3.7 - Revisions posteriors de l'especificació i l'anàlisi....15
4 - Disseny....16
4.1 - Disseny de la interficie d'usuari....16
4.2 - Arquitectura de l'aplicació Servidor....16
4.2.1 - Operations....17
4.2.2 - Diagrama de seqüència....19
4.3 - Arquitectura de l'aplicació client....19
4.3.1 - Rutes....20
4.3.2 - Components....23
4.3.3 - Altres elements....24
4.4 - Comunicació entre l'aplicació client i el servidor....25
5 - Codificació i desplegament a producció....26
5.1 - Mètodes de desenvolupament....26

5.2 - Desplegament a producció.....26
6 - Conclusions.....28
6.1 - Per on continuar?.....28
7 - Bibliografia.....29

1 - Introducció

En primer lloc, m'agradaria presentar el resultat del projecte, i així establir d'entrada una sèrie de punts de referència que aniran apareixent durant la lectura d'aquest document.

Tot seguit presentaré els objectius del projecte des del punt de vista didàctic i acadèmic però també els objectius pràctics que pretenia assolir l'aplicació que s'ha desenvolupat i el seu context.

1.1 - El resultat

El resultat pràctic del projecte es pot veure a l'adreça web https://q-tasks.com.

El subdomini habilitat per a proves és "uoc". Es pot indicar aquest subdomini al formulari que es presenta en visitar la Q-Tasks o anar directament a https://uoc.q-tasks.com.

Usuaris i contrasenyes:

- Login: fpradosc@uoc.edu
  Contrasenya: secret123
- Login: aromanos@uoc.edu
  Contrasenya: secret123
- Login: adaradoumis@uoc.edu
  Contrasenya: secret123
- Login: paucompte@uoc.edu
  Contrasenya: secret123
- Login: o@o.o
  Contrasenya: secret123
- Login: z@z.z
  Contrasenya: secret123

El codi font d'aquest projecte consta de dues aplicacions, una desenvolupada en Javascript, que es pot consultar a l'adreça https://github.com/pauc/q-tasks-ember, i una desenvolupada en Ruby que es pot consultar a https://github.com/pauc/q-tasks-rails.

Per acabar, les diapositives de la presentació de l'aplicació es poden veure a http://prezi.com/54pvtb0y4rtr/?utm_campaign=share&utm_medium=copy.

1.2 - Objectiu principal

L'objectiu principal del projecte és de tipus didàctic, i aquest fet ha condicionat moltes de les decisions que s'han pres.

En la meva experiència com a programador web sempre he treballat amb un tipus d'aplicació en el que el contingut que es presenta a l'usuari es construeix al servidor. És el funcionament "clàssic" d'una pàgina web: el navegador fa una petició al servidor, el servidor rep la petició i la processa, fent si cal consultes a una base de dades, i després construeix l'HTML a partir de *templates* i torna aquest contingut al client. Aquest mecanisme segueix sent útil en moltes situacions, sobretot en pàgines web orientades a contingut, per exemple, però en altres ocasions no és el més adient.

En el desenvolupament d'aquest projecte es pretenia aprendre una forma alternativa de construir aplicacions web, consistent en la combinació d'una aplicació Javascript que s'executa al navegador de l'usuari, i d'una API REST que s'executa en un servidor. Les dades entre un i altre sistema s'intercanvien usant JSON, i l'aplicació Javascript reacciona a les accions de l'usuari i els canvis en les dades sense necessitat de refrescar la pàgina sencera. Sovint s'anomena aquesta manera de construir aplicacions web com "SPA" (*single page application*[1]).

A partir d'aquest objectiu principal es van definir també els següents objectius didàctis:

- Ampliar coneixements de javascript.
- Experimentar amb un framework javascript per a crear aplicacions web. En concret
- EmberJS.
- Ampliar coneixements de Ruby i Ruby on Rails, i Programació Orientada a Objectes en
- general.
- Aprendre disseny d'APIs REST.
- Ampliació de coneixements d'administració de xarxes i servidors.

1.3 - Objectius funcionals i tècnics: Q-Tasks

Q-Tasks és el nom de l'aplicació producte d'aquest projecte. Es tractava de construir un prototip avaluable d'una eina per a la gestió de tasques enfocat al seguiment quotidià del treball en grup.

Els requisits funcionals que es van definir a l'inici del projecte són els següents:

1 https://en.wikipedia.org/wiki/Single-page_application

- Fàcil i simple. Començar a fer servir Q-Tasks ha de ser senzill per a qualsevol perfil de treballador.
- Ha de permetre separar tasques en diferents projectes.
- S'ha de poder veure de manera molt directa quines persones estan implicades en cada tasca.
- L'accés i visualització dels recursos associats a una tasca també ha de ser intuïtiu i còmode.
- Ha d'oferir algun mètode per a establir relacions de dependències entre tasques.

I per la part tècnica:

- Desacoblament entre la capa de presentació i interacció amb l'usuari (frontend) i la capa de gestió i processament de dades (backend). Una API REST donarà servei a l'aplicació client, o en un futur, a més d'una.
- Comunicació segura entre clients i servidors (https).
- Autenticació de les peticions al servidor mitjançant OAuth.

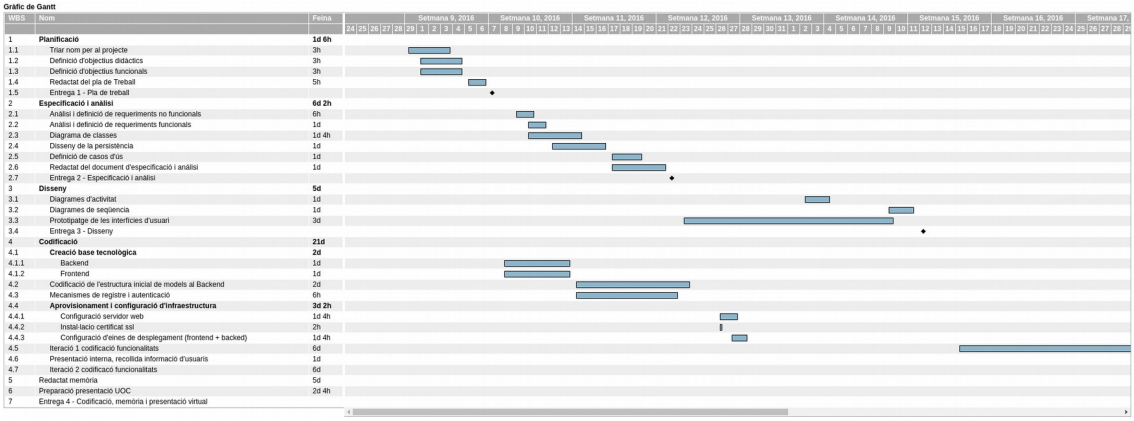
2 - Planificació

La planificació temporal del desenvolupament del projecte es va fer sobretot a partir de les dates d'entrega que s'exigien a l'assignatura, i tenint en compte la disponibilitat de temps en les diferents etapes. La majoria d'hores de treball s'han concentrat en caps de setmana i dies de festa.

Com a punt a destacar indicar que es va planificar arribar a mitjans d'abril amb tot a punt per a poder centrar-se en el desenvolupament de funcionalitats de l'aplicació, és a dir, havent realitzat tot un seguit de tasques tècniques prèvies relacionades amb la infraestructura i la configuració bàsica dels components del projecte, com són la configuració de servidors, de certificats ssl, mecanismes de desplegament a producció, etc...

A continuació es mostra una captura del grafic de Gantt resultat de la fase de planificació, que per qüestions d'espai no es pot veure sencer; l'informe complet es pot consultar a http://cv.uoc.edu/web/~paucompte/TFC.html.

Grafic de Gantt



WBS	Nom	Feina
1	**Planificació**	**1d 6h**
1.1	Triar nom per al projecte	3h
1.2	Definició d'objectius didàctics	3h
1.3	Definició d'objectius funcionals	3h
1.4	Redactat del pla de Treball	5h
1.5	Entrega 1 - Pla de treball	
2	**Especificació i anàlisi**	**6d 2h**
2.1	Anàlisi i definició de requeriments no funcionals	6h
2.2	Anàlisi i definició de requeriments funcionals	1d
2.3	Diagrama de classes	1d 4h
2.4	Disseny de la persistència	1d
2.5	Definició de casos d'ús	1d
2.6	Redactat del document d'especificació i anàlisi	1d
2.7	Entrega 2 - Especificació i anàlisi	
3	**Disseny**	**5d**
3.1	Diagrames d'activitat	1d
3.2	Diagrames de seqüencia	1d
3.3	Prototipatge de les interfícies d'usuari	3d
3.4	Entrega 3 - Disseny	
4	**Codificació**	**21d**
4.1	**Creació base tecnològica**	**2d**
4.1.1	Backend	1d
4.1.2	Frontend	1d
4.2	Codificació de l'estructura inicial de models al Backend	2d
4.3	Mecanismes de registre i autenticació	6h
4.4	**Aprovisionament i configuració d'infraestructura**	**3d 2h**
4.4.1	Configuració servidor web	1d 4h
4.4.2	Instal·lació certificat ssl	2h
4.4.3	Configuració d'eines de desplegament (frontend + backed)	1d 4h
4.5	Iteració 1 codificació funcionalitats	6d
4.6	Presentació interna, recollida informació d'usuaris	1d
4.7	Iteració 2 codificació funcionalitats	6d
5	Redactat memòria	5d
6	Preparació presentació UOC	2d 4h
7	Entrega 4 - Codificació, memòria i presentació virtual	

2.1 - Desviaments i ajustos en la planificació

En general el desenvolupament del projecte ha seguit la força la planificació inicial, i clarament va ser un encert avançar les tasques de preparació de la infraestructura. Tot i així, en l'etapa de disseny no es va poder dedicar tot el temps que s'havia planificat, no es va complir amb els terminis d'entrega i la feina realitzada no complia els requisits mínims de completesa i qualitat. S'ha intentat corregir aquestes mancances a l'etapa d'implementació.

3 - Especificació i anàlisi

En els següents apartats es presenten de manera més detallada les especificacions de l'aplicació desenvolupada i el resultat de la fase d'anàlisi.

3.1 - Descripció del problema i l'entorn.

Per a la realització del projecte s'ha partit de la necessitat real d'un equip de treball real. Tot i que la intenció no ha sigut construir una aplicació completa, el fet d'intentar resoldre una problemàtica que resulta propera ha sigut un element motivador.

L'equip darrere de getquipu.com ens dediquem a construir una aplicació web destinada a autònoms, petites i mitjanes empreses, associacions, i gestories per a que puguin gestionar la facturació de manera fàcil i còmode. Som un equip format per unes 12 persones que bàsicament inclou una àrea d'atenció al client, un àrea de màrqueting, i una àrea de desenvolupament.

L'activitat diària de l'equip, per tant, gira al voltant del manteniment i desenvolupament de l'aplicació Quipu, però volem tenir la possibilitat de definir també treballs i tasques fora d'aquest àmbit. Per exemple, el portar a terme una nova ronda de finançament no involucra l'equip sencer i no encaixa dintre del flux normal de treball. Volem, per tant, poder definir diferents projectes dins de l'aplicació.

Quan s'ha d'afegir una nova funcionalitat a Quipu, llençar una campanya de màrqueting, redissenyar una part de l'aplicació, etc, hi ha un conjunt de tasques que cal portar a terme que involucren a diferents persones de diferents àrees. Les tasques pertanyents a una mateixa "missió", "treball" o "objectiu" s'han de poder veure de manera agrupada, i les tasques s'han de poder assignar a un treballador tant en el moment de crearr-les com més endavant. A més a més, aquestes diferents tasques tenen dependències entre elles, com per exemple quan una persona ha de preparar certs documents per a que un altre treballador pugui fer la seva part de la feina.

En tot moment els usuaris han de poder veure en quin estat es troba una tasca, si encara està pendent, si està en procés o si ja s'ha enllestit i també s'ha de poder visualitzar l'estat de les seves dependències. Una dependència es considerarà satisfeta si la tasca de la que es depèn ja ha estat enllestida.

Finalment, sovint hi ha la necessitat de deixar anotacions i fer comentaris relacionats amb les tasques.

3.2 - Requeriments funcionals

Així doncs, s'ha desenvolupat una aplicació per a la gestió del treball en equip que en resum, té les següents funcionalitats:

- Gestió de projectes. S'han de poder crear projectes i navegar entre els diferents projectes existents. No es preveu que existeixin gaires projectes simultàniament.
- Els projectes serveixen principalment com a contenidors per als grups de tasques o treballs.
- Gestió dels treballs d'un projecte. Crear, llistar i eliminar. Els treballs són grups de tasques encaminats a assolir un objectiu comú.
- Als treballs s'hi poden associar recursos que serveixin de suport per a realitzar les tasques.
- Gestió de les tasques d'un treball o grup de tasques. És la funcionalitat més complexa, i a part de la creació, visualització i eliminació de tasques s'ha de poder:
  - Establir i visualitzar dependències entre tasques.
  - Assignar tasques a usuaris.
  - Modificar l'estat de les tasques.
  - Afegir comentaris a les tasques.

3.3 - Requeriments no funcionals

A continuació es detallen els requeriments d'infraestructura, seguretat, accessibilitat o d'implementació i metodològics que es van definir.

3.3.1 - Seguretat

No ens trobem d'avant d'una aplicació on es treballi amb dades privades, confidencials o sensibles, i a més es tracta, almenys inicialment, d'un prototip per a ús intern de la que no es farà publicitat, per tant no s'han definit uns requeriments de seguretat especials, només aquells que formen part del conjunt de bones pràctiques pel que fa a seguretat i privacitat de qualsevol aplicació web:

- Comunicació entre clients i servidor via https.
- Les contrasenyes es desen a la BdDD usant xifrat irreversible.
- Si n'hi haguessin, no es mostren dades personals (telèfon, adreça...) d'un usuari als altresusuaris. De moment, però, l'aplicació no conté cap dada d'aquest tipus.

3.3.2 - Rendiment, disponibilitat i accessibilitat

Igual que en el cas anterior, tampoc s'ha definit requeriments especials en aquests aspectes. A l'equip on es vol començar a provar l'aplicació no hi ha cap treballador amb diversitat funcional, ni tampoc s'usarà en un entorn crític, a sigui que un error a l'aplicació o una caiguda dels servidors no suposaria un problema greu.

Pel que fa al rendiment, l'únic requeriment és que sigui còmode d'usar, amb temps d'espera i reacció raonables.

3.3.3 - Adaptabilitat

Tant jo com altres companys de l'equip al qui anirà destinat el projecte hem usat al llarg de la nostra activitat professional diferents eines per organitzar tasques. És un problema que es pot abordar des de moltes perspectives diferents i per al que existeixen centenars d'aplicacions que l'intenten solucionar, amb més o menys èxit. Sovint, però, no es tracta tant de que una solució sigui bona o no en sí mateixa, sinó de que s'adapti bé a les pràctiques dels equips i els usuaris.

En un principi l'aplicació no presenta dificultats remarcables a nivell tècnic a part de l'aprenentatge d'una nova eina com és EmberJS, però per mi sí que resulta un repte molt important tot el que fa referència al disseny de la interacció amb l'usuari i la presentació de les dades.

Vist que és un problema amb moltes solucions possibles, i tenint en compte les meves limitacions i dificultats pel que fa a disseny, es tracta per tant de mantenir-se el més flexible possible per a adaptar-se a canvis de requeriments o estendre l'aplicació amb noves funcionalitats quan sigui necessari.

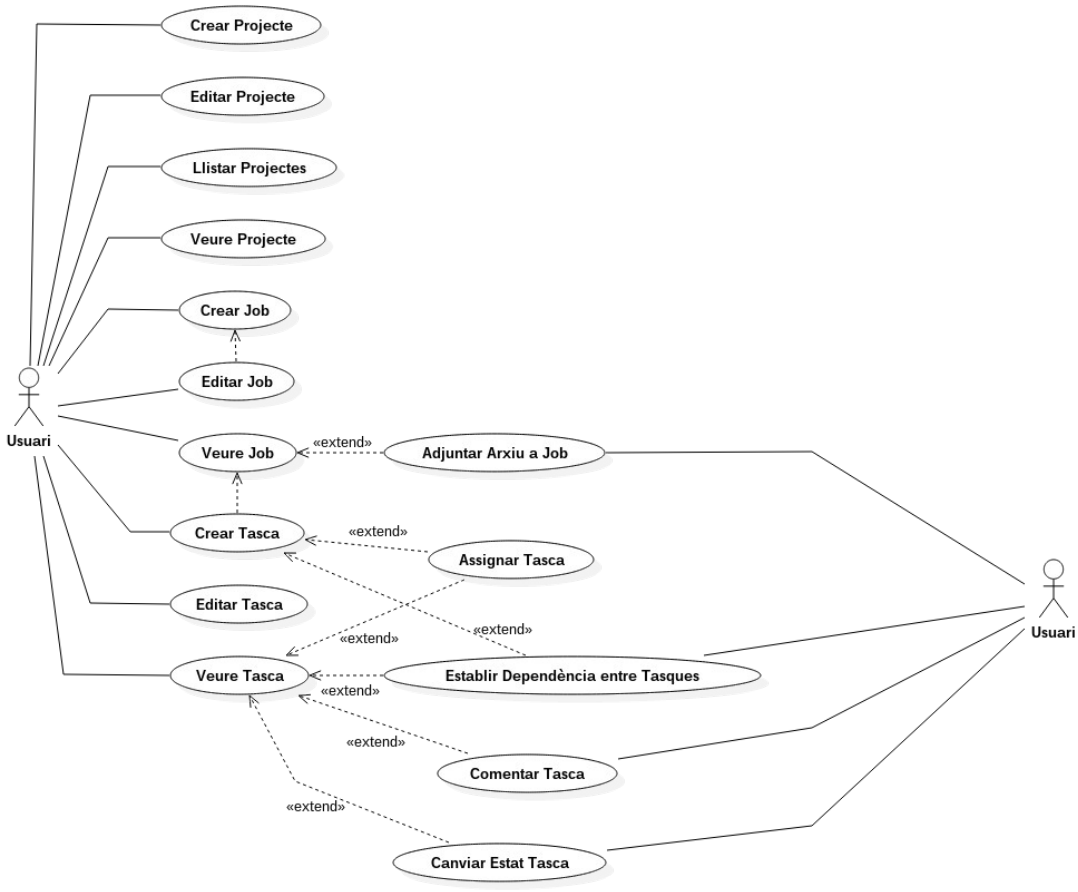
3.4 - Casos d'ús

A continuació es presenten els casos d'ús definits per al projecte.

3.4.1 - Actors

Només s'ha contemplat un actor per a l'aplicació, l'Usuari. Més endavant, si el desenvolupament de l'aplicació continua, caldrà definir-ne de nous, com "administradors d'equips", per exemple, amb privilegis especials per administrar els usuaris o canviar certes configuracions d'un equip.

3.4.2 - Diagrama de casos d'ús



3.4.3 - Especificació textual dels casos d'ús

En tots els casos d'ús es pressuposa que l'actor es troba autenticat a l'equip (Team) amb el que vol operar.

D'alguns casos d'ús no se'n detalla l'especificació per la seva senzillesa.

En tots els casos d'ús només hi intervé l'actor Usuari, així que aquesta informació s'obviarà.

Crear Projecte

Resum de la funcionalitat: permet introduir un nou projecte a l'equip indicantne el nom.

Precondició: El projecte no existeix a la base de dades.

Postcondició: S'ha registrat el projecte al sistema.

Procés: L'Usuari només cal que introdueixi el projecte indicant-ne el nom

Veure Projecte

Resum de la funcionalitat: permet visualitzar i navegar per les dades associades a un projecte.

Precondició: El projecte ja existeix a la base de dades, però no és el projecte actiu per a l'usuari.

Poscondició: L'usuari ha accedit a les dades del projecte i aquest és per ell el projecte actiu.

Casos d'us relacionats: Llistar projectes.

Procés: L'Usuari selecciona el projecte d'entre la llista de projectes de l'equip.

Crear Goal ("Objectiu" o "Grup de tasques")

Resum de la funcionalitat: Permet introduir les dades d'un nou grup de tasques en un projecte.

Precondició: El projecte on s'ha d'afegir el "Goal" o "Grup de tasques" existeix al sistema, el "Goal" a afegir no existeix.

Postcondició: El Goal ha quedat registrat al sistema i associat al projecte.

Casos d'ús relacionats: Veure projecte

Procés: L'usuari introdueix un nom de l'objectiu que vol crear. L'Objectiu quedarà associat al Projecte actiu en el moment de l'acció.

Veure Goal

Resum de la funcionalitat: L'usuari selecciona un "Objectiu" per veure'n les dades, entre elles les tasques associades.

Precondició: El Goal existeix a la base de dades, però no és el Goal actiu per a l'usuari. El projecte actiu és el projecte associat al Goal.

Postcondició: L'usuari ha accedit a les dades del Goal i aquest ha esdevingut el Goal actiu.

Casos d'ús relacionats: Crear tasca, veure projecte, Associar arxiu a un Goal

Procés: L'usuari selecciona el Goal d'entre el llistat d'objectius del projecte actiu.

Adjuntar arxiu a un objectiu

Resum de la funcionalitat: L'usuari adjunta un arxiu al Goal actiu.

Precondició: El Goal al qual es vol adjuntar l'arxiu existeix al sistema.

Poscondició: L'arxiu queda desat als servidors i accessible des del Goal al qual està associat.

Casos d'ús relacionats: Veure Goal

Procés: L'usuari arrossega l'arxiu a en una zona de l'aplicació o acciona un botó que li que obre un navegador d'arxius i selecciona l'arxiu desitjat.

Crear Tasca

Resum de la funcionalitat: L'Usuari afegeix una tasca al Goal actiu.

Precondició: La tasca no existeix a la Base de dades. El Goal al qual es vol afegir la tasca existeix i és l'objectiu actiu per a l'usuari.

Poscondició: La tasca existeix a la Base de Dades i està associada al Goal actiu. La tasca és activa per a l'usuari.

Casos d'ús relacionats: Veure Goal

Procés: L'usuari indica el nom de la tasca i la descripció. A més també pot indicar-ne la dependència respecte d'alguna altra tasca del mateix Goal, afegir-hi un primer comentari i assignar la tasca a un usuari (ell mateix o un altre).

Casos d'ús relacionats: Veure objectiu, Assignar tasca, Establir Dependència entre Tasques, Comentar Tasca.

Establir dependència entre tasques

Resum de la funcionalitat: L'usuari estableix una relació de dependència entre dues tasques del mateix Goal, una és la "dependència" i l'altra la "depenent"

Precondició: Almenys la tasca "dependència" existeix a la base de dades. La tasca "depenent" o bé ja existeix i és la tasca activa o s'està creant.

Poscondició: Si la tasca depenent ja existia al sistema, s'ha creat una dependència entre ella i l'altra tasca. Si la tasca depenent s'està creant, s'ha establert la relació de dependència per a ser persistida en el moment en que es desi la tasca.

Casos d'ús relacionats: Crear Tasca, Veure Tasca

Assignar tasca

Resum de la funcionalitat: L'usuari assigna la tasca a un usuari de l'equip, que pot ser ell mateix.

Precondició: La tasca existeix i és la tasca activa o s'està creant. L'usuari al qual es vol assignar la tasca existeix a la base de dades i forma part de l'equip.

Postcondició: La tasca està assignada a l'usuari indicat.

Casos d'ús relacionats: Veure Tasca, Crear Tasca.

Procés: L'usuari selecciona un usuari de l'equip d'un llistat on consten tots els membres de l'equip.

Comentar Tasca

Resum de la funcionalitat: L'usuari afegeix un comentari en una tasca.

Precondició: La tasca existeix a la base de dades i és la tasca activa o bé s'està creant (només primer comentari).

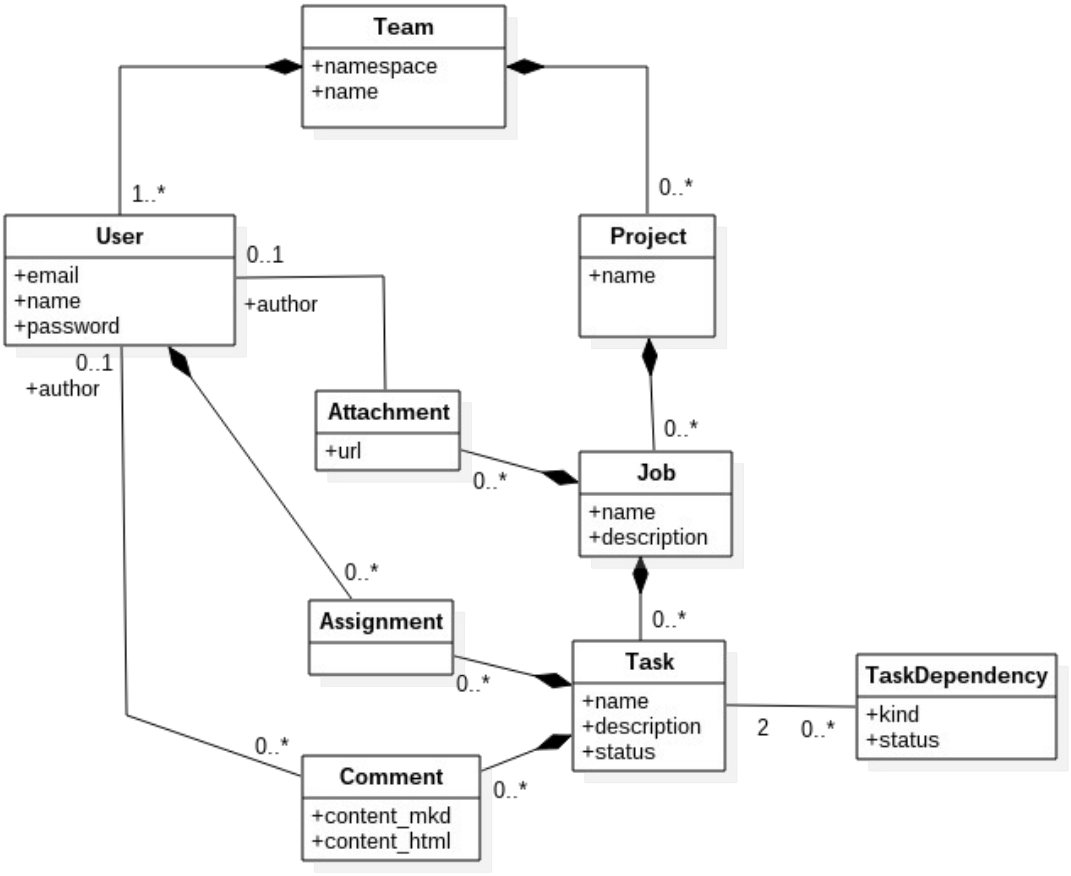
Postcondició: S'ha afegit un comentari a la tasca.

Tasques relacionades: Veure Tasca, Crear Tasca.

Procés: L'usuari afegeix el text del comentari.

3.5 - Diagrama estàtic de classes

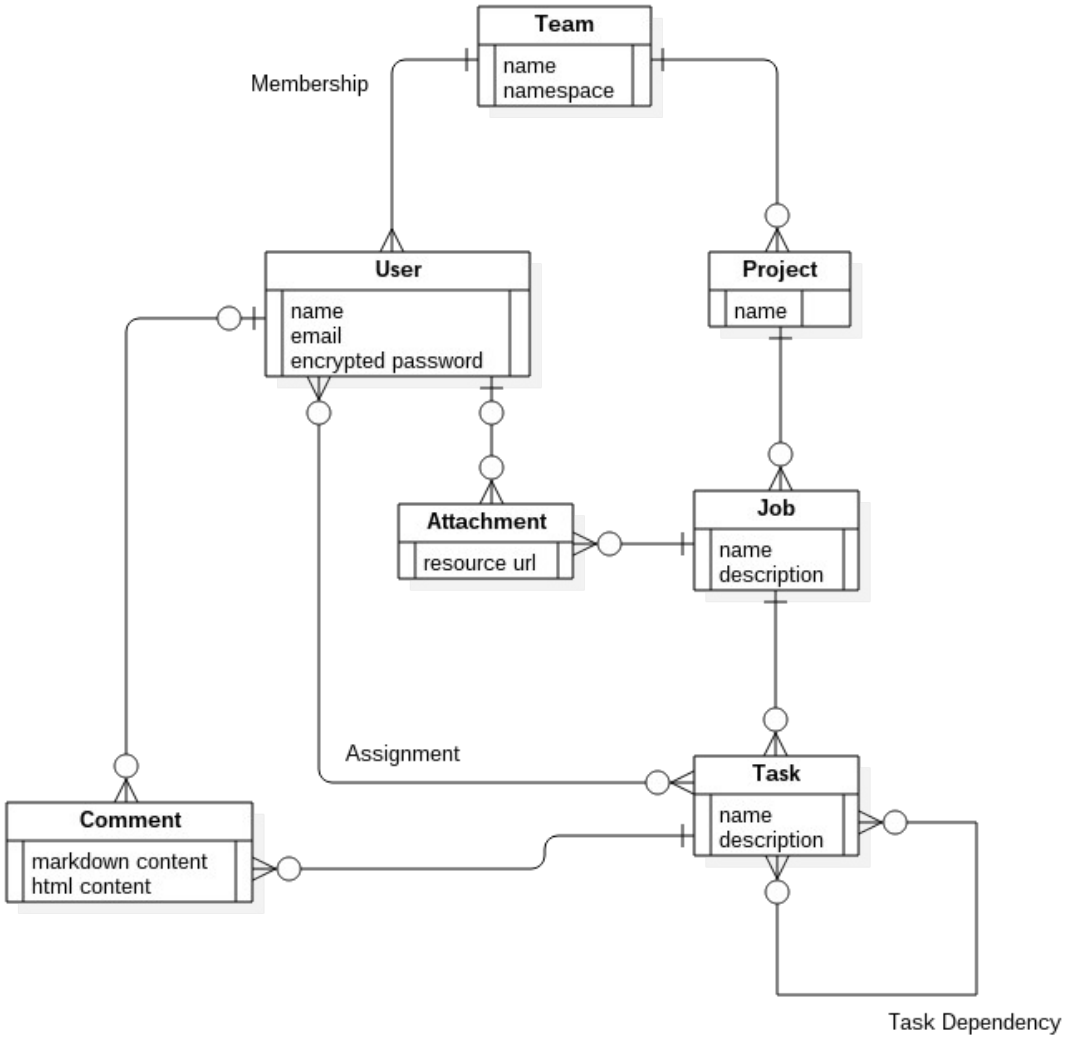
A continuació es presenta el diagrama de models que es va realitzar en la fase d'anàlisi, i que ha servit de base per al desenvolupament de les aplicacions client i servidor de Q-Tasks:



3.6 - Disseny de la persistència

Per a la persistència de dades s'ha utilitzat el gestor de bases de dades relacionals PostgreSQL. L'estructura de la base de dades s'ha definit a partir dels esquemes següents:

Model entitat relació



Model relacional

- TEAMS (id, name, namespace, created_at, updated_at)

- USERS (id, email, name, encrypted_password, team_id, created_at, updated_at)
  on {team_id} referencia TEAMS

- PROJECTS (id, name, team_id, created_at, updated_at)
  on {team_id} referencia TEAMS

- JOBS (id, name, description, project_id, created_at, updated_at)
  on {project_id} referencia PROJECTS

- TASKS (id, name, description, status, job_id, created_at, updated_at)

on {job_id} referencia JOBS

- ASSIGNMENTS (id, task_id, user_id, created_at, updated_at)
  on {task_id} referencia TASKS
  on {user_id} referencia USERS

- COMMENTS (id, mkp_content, html_content, task_id, author_id, created_at, updated_at)
  on {task_id} referencia TASKS
  on {author_id} referencia USERS i pot prendre valors nuls

- ATTACHMENT (id, url, task_id, uploaded_by_id, created_at, updated_at)
  on {task_id} referencia TASKS
  on {uploaded_by_id} referencia USERS i pot prendre valors nuls

- TASK_DEPENDENCIES (id, dependent_id, dependency_id, kind, created_at, updated_at)
  on {dependent_id} referencia TASKS
  on {dependency_id} referencia TASKS

3.7 - Revisions posteriors de l'especificació i l'anàlisi

Durant les etapes de disseny i implementació s'han revisat alguns dels casos d'ús definits inicialment, i també algun detall de les entitats que s'havien definit en els diagrames de classes o de casos d'ús.

Pel que fa als casos d'ús, s'ha decidit no implementar els casos d'ús més senzills i que aporten poc per a l'assoliment dels objectius del projecte. En concret no s'ha implementat l'edició i eliminació de projectes.

Pel que fa al diagrama de classes i al model entitat relació s'ha canviat el nom de la classe "Job" a "Goal".

Pel que fa a l'assignació de tasques a treballadors, s'ha implementat tal com estava definida inicialment, podent assignar més d'un usuari a una tasca, però aquesta possibilitat s'ha limitat al frontend de manera que només es pot assignar un treballador. S'ha pres aquesta decisió a partir de consultar dos companys de treball a qui s'ha presentat l'aplicació en un estat de desenvolupament inicial i se'ls ha demanat que en donessin la seva opinió i en destaquessin errors i mancances.

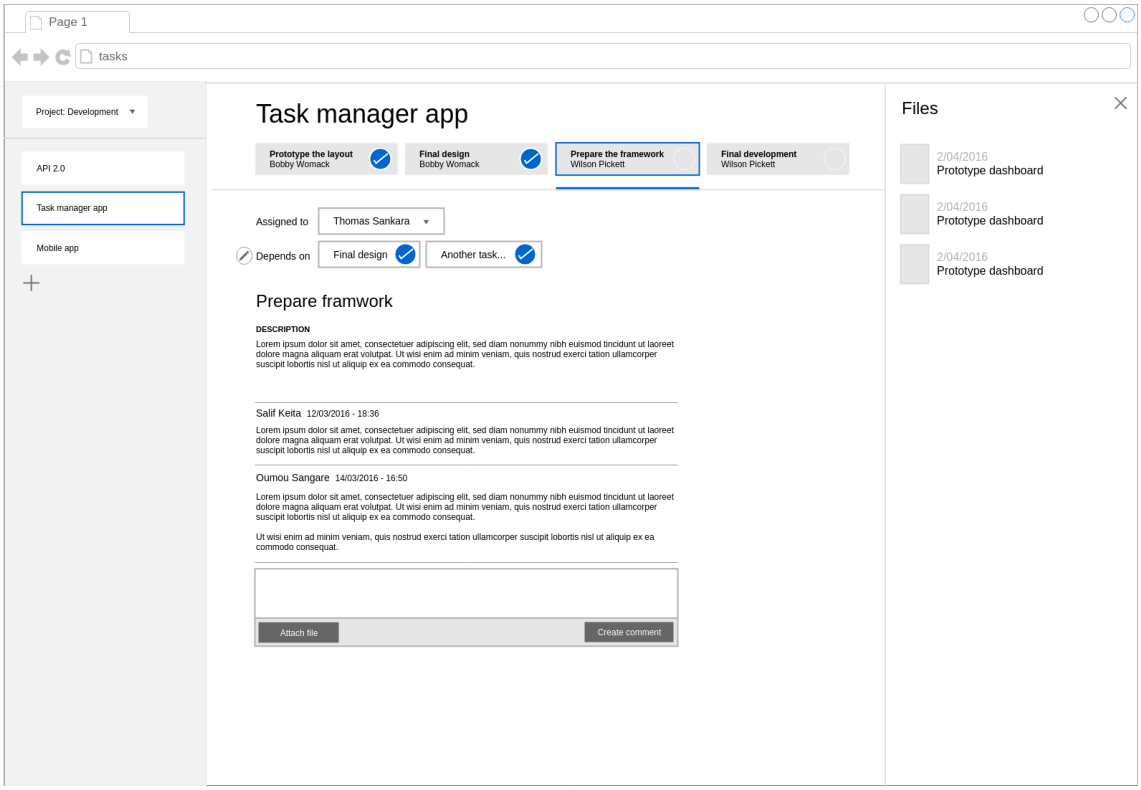
4 - Disseny

Com ja he assenyalat anteriorment en el capítol sobre planificació, no es va poder dedicar el temps necessari al disseny de l'aplicació. Però tot i que no es va completar la feina de manera exhaustiva, la feina que es va fer ha resultat molt útil en l'etapa de codificació.

4.1 - Disseny de la interficie d'usuari

Es va dissenyar una sola pàgina de la interfície d'usuari. Tot i que en un principi pot semblar que s'hagi dissenyat només una fracció de l'aplicació, en realitat quasi bé totes les funcionalitats i casos d'ús definits entren en aquesta única pàgina.

Es pot comparar el disseny elaborat amb el resultat final a https://uoc.q-tasks.com.



Molt poques coses han canviat en el moment de programar l'aplicació. Només petits detalls de disposició d'alguns elements.

4.2 - Arquitectura de l'aplicació Servidor

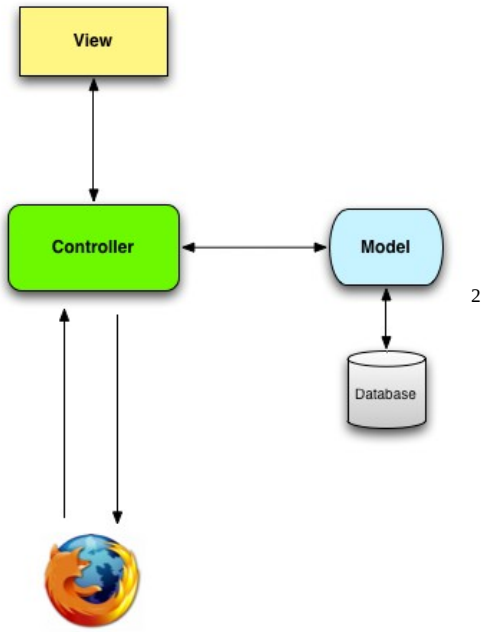
L'aplicació servidor s'ha desenvolupat amb Ruby on Rails. Ruby on Rails és un framework per al desenvolupament web escrit en Ruby, molt complet i que compta amb una comunitat i un ecosistema extensos.

Ruby on Rails implementa una aproximació del patró model-vista-controlador, i en general l'aplicació realitzada aprofita aquesta arquitectura, però s'ha afegit un tipus de component més: les operacions. Així que d'alguna manera l'aplicació que serveix la API de Q-Tasks és una implementació del patró Model-Vista-Controlador-Operació.

4.2.1 - Operations

Per què les operacions?

Aquest és l'esquema típic de l'estructura d'una aplicació Rails:



2

El controlador rep les peticions del navegador, demana o envia la informació que calgui als models (rails per defecte usa una implemetació d'Active Record) que són els responsables de la interacció amb la base de dades.

Amb la informació que retorna el model el controlador construeix la resposta usant les vistes o *templates*, i tornarà el contingut al client.

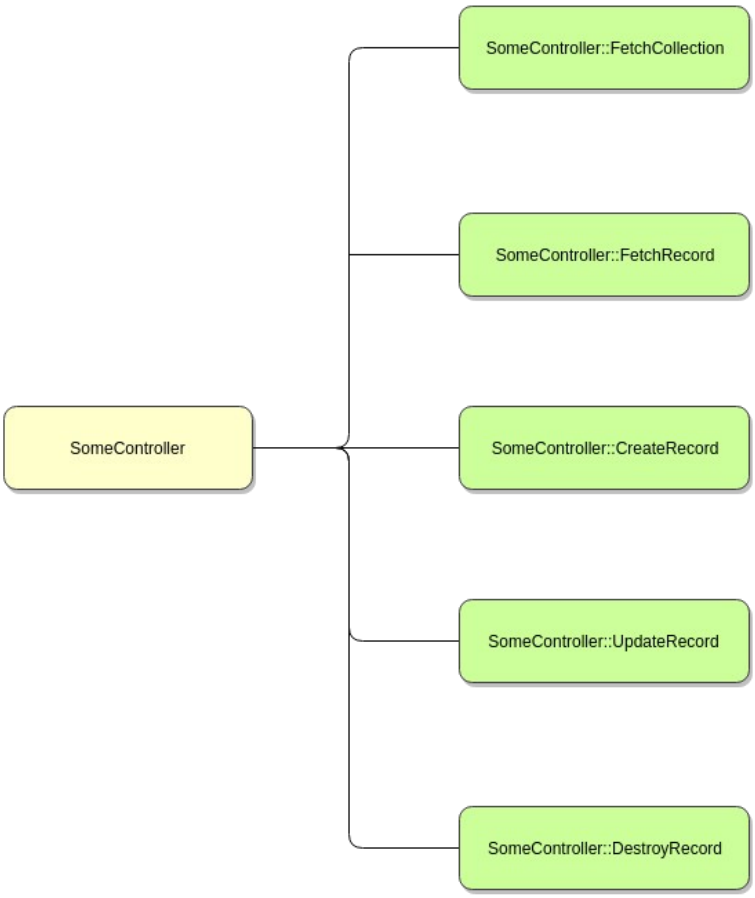
Aquesta estructura en general és satisfactòria per a aplicacions molt senzilles, però té mancances importants quant la lògica de negoci de l'aplicació es complica. Hi ha accions que s'han de portar a terme durant el transcurs d'una petició que no acaben de tenir un bon encaix ni en els controladors ni en els models, i que poden fer que la complexitat de les classes que els implementen creixi de manera descontrolada.

Com a proposta per a intentar resoldre aquest problema s'ha experimentat amb aquest nou tipus de component del sistema, les operacions, destinades a encapsular precisament la lògica de negoci a satisfer en les diferents accions.

2 https://www.railstutorial.org/book/beginning#sec-mvc

La implementació concreta de les classes Operation a Q-Tasks es pot veure a https://github.com/pauc/q-tasks-rails/tree/master/app/operations.

Com es pot veure a l'enllaç anterior cada controlador usa un conjunt d'operacions per a satisfer els diferents tipus de peticions que pot rebre seguint l'esquema bàsic següent:

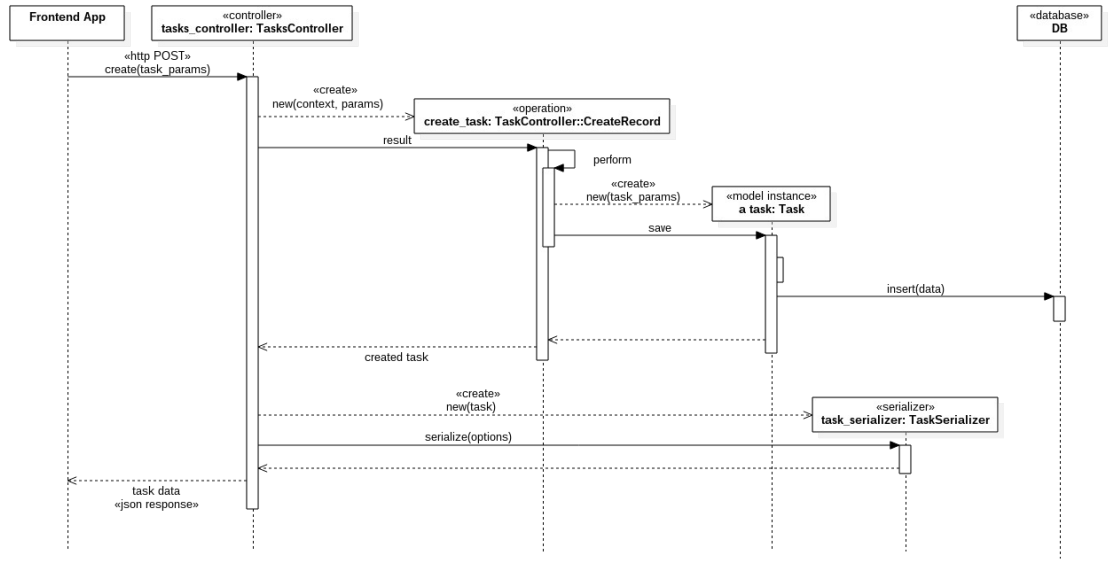


Aquest esquema bàsic conté només les operacions típiques d'un esquema REST, però res impedeix implementar-ne d'altres si calgués.

No soc, ni molt menys, el primer en detectar aquesta "feblesa" de l'arquitectura per defecte que proposa Ruby on Rails, ni el primer en pensar aquest tipus de solució. En gran mesura aquestes classes estan basades en la proposta que fa Nick Sutterer (apotonick) al projecte Trailblazer, tot i que ell en fa una implementació molt més completa i ambiciosa. Podria haver optat per usar una implementació ja feta d'aquesta estructura, però ha sigut més interessant i didàctic implementar-la jo mateix.

4.2.2 - Diagrama de seqüència

A Continuació es presenta el diagrama de seqüencia on es veu la interacció i el flux de missatges entre els diferents components descrits durant el processament d'una petició d'exemple:

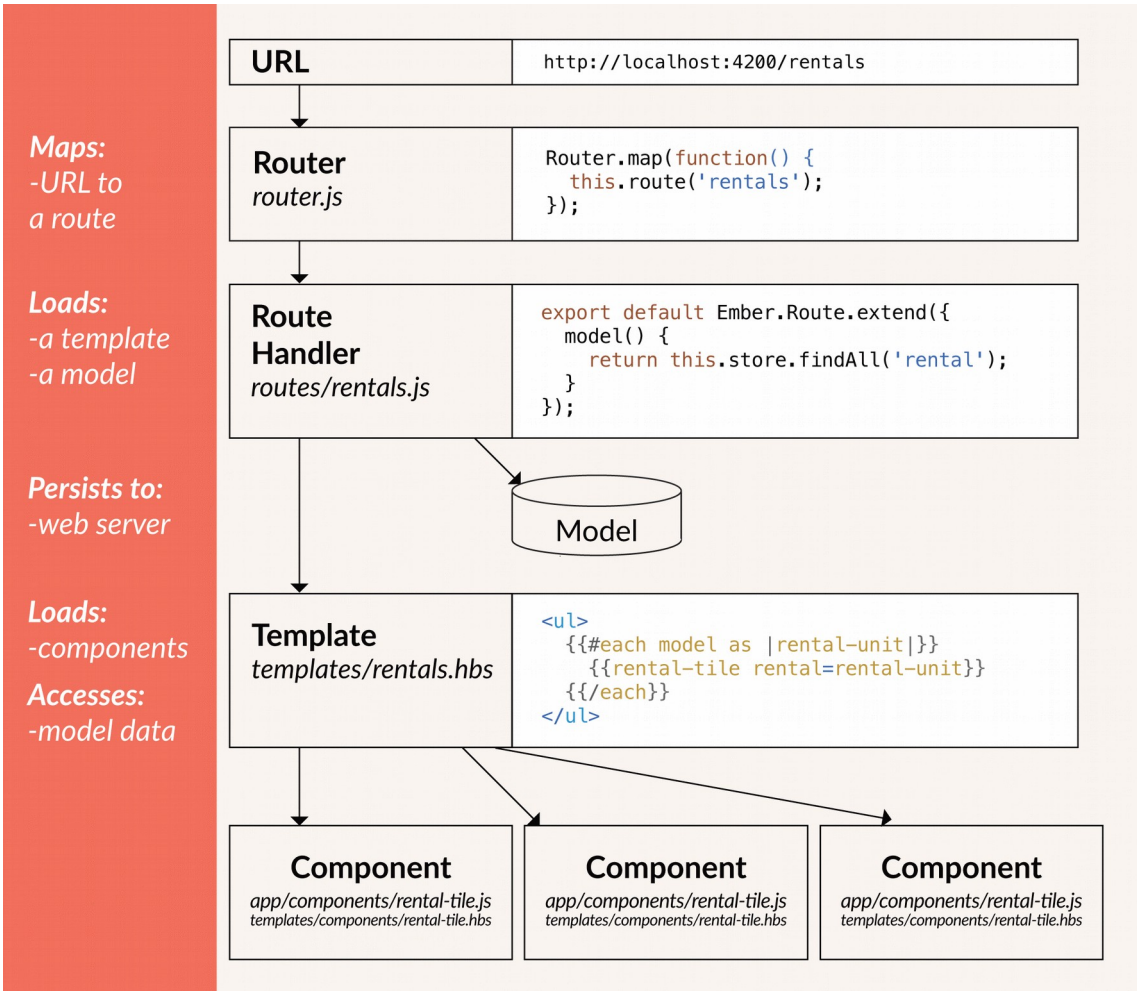


En aquest cas es tracta d'una petició per a crear una registre d'una nova *Task*, però en tots els tipus de peticions se segueix el mateix esquema d'interacció entre els components del sistema.

4.3 - Arquitectura de l'aplicació client

L'aplicació client s'ha implementat amb EmberJS, que a la pàgina web del projecte es defineix com *"A framework for creating ambitious web aplications"*.

Usaré un esquema de les guies d'EmberJS per presentar l'esquema bàsic i els principals elements d'una aplicació feta amb EmberJS, i par tant també del client de Q-Tasks:



En línies generals podríem dir que els elements centrals d'una aplicació Ember són les les Rutes i Router.

El Router és el que "decideix", en funció de la URL del navegador, quines rutes s'han d'activar, i aquestes són les encarregades de decidir què (quines dades / models) i com (*templates*) es mostraran a l'usuari. Així doncs podríem dir que aquest subsistema format per les rutes i el router és l'encarregat de traduir les URLs a un estat concret de l'aplicació.

En els següents apartats entrarem una mica més en detall sobre el rol i responsabilitats dels dos tipus d'elements que per a mi resulten més importants en l'arquitectura d'una aplicació Ember, les rutes i els components, i seguidament es presentaran molt breument la resta d'elements que la conformen.

4.3.1 - Rutes

Podríem resumir les responsabilitats de les rutes amb els següents punts:

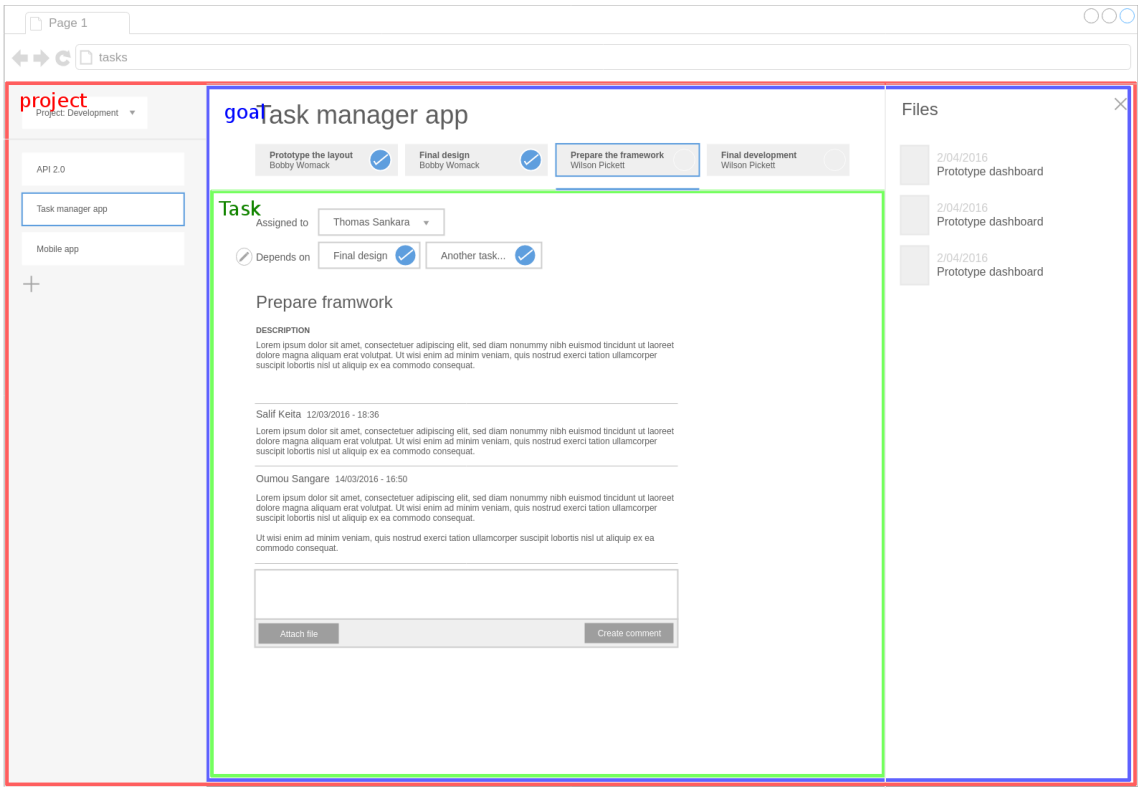
- Estableixen l'estat de l'aplicació. Quines dades es presenten i com es visualitzen.

- S'encarreguen de la comunicació amb el subsistema de persistència de dades, que pot ser una API remota, com és el cas de Q-Tasks, o qualsevol altre mecanisme (local-storage, per exemple).

A continuació es mostra el Router de l'aplicació q-tasks-ember, que també es pot consultar a https://github.com/pauc/q-tasks-ember/blob/master/app/router.js.

```
import Ember from 'ember';
import config from './config/environment';

const Router = Ember.Router.extend({
  location: config.locationType
});

Router.map(function() {
  this.route('login');

  this.route('first-project');

  this.route('new-project');

  this.route('projects', function() {

    this.route('project', { path: '/:project_id', resetNamespace: true }, function() {

      this.route('first-goal');

      this.route('goal', { path: 'goals/:goal_id', resetNamespace: true }, function() {

        this.route('task', { path: 'tasks/:task_id', resetNamespace: true });
      });
    });
  });

  this.route('not-found');
});

export default Router;
```

Cal assenyalar també que en una aplicació Ember rutes niuades impliquen *templates* niuats. Per visualitzar aquest fet pot ser útil la figura següent on es mostra la relació entre els *templates* i rutes a partir del disseny de la interfície d'usuari presentada anteriorment:



S'ha omès la ruta *projects* perquè en el seu *template* no conté HTML, i per tant no té "efectes visibles", però això no vol dir que no tingui cap utilitat, ja que sí que té lògica associada.

4.3.2 - Components

Dèiem en l'apartat anterior que les rutes defineixen la informació que es presentarà a l'usuari (quines dades s'utilitzaran) i com es presentarà aquesta informació (quin *template* s'usarà). Doncs bé, alhora de presentar la informació i construir l'HTML hi ha un tipus d'element que s'usa de manera extensiva a les aplicacions Ember que són els components.

Podríem dir que els components són una implementació dels custom-elements[3] d'HTML.

Els components **encapsulen** un contingut (HTML) però també un comportament (que es defineix amb javascript), i tota la informació que necessitin se'ls ha de proporcionar de manera explícita quan són invocats.

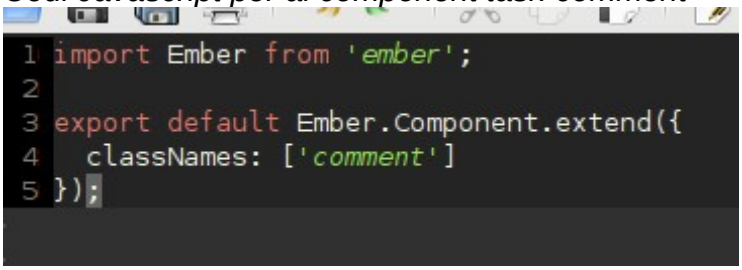
Són un mecanisme perfecte per construir elements re-usables, i de fet en l'ecosistema de complements d'EmberJS trobem molts complements que consisteixen en components.[4]

3 https://developer.mozilla.org/en-US/docs/Web/Web_Components/Custom_Elements

4 Alguns exemples de components usats a q-tasks-ember que es distribueixen com a complements poden ser http://www.ember-power-select.com/ o https://github.com/tim-evans/ember-plupload

A continuació es presenten tres figures per presentar el component més senzill usat a q-tasks-ember, anomenat *task-comment*, que mostra el comentari que se li passa com a paràmetre en invocar-lo:

Codi Javascript per al component task-comment

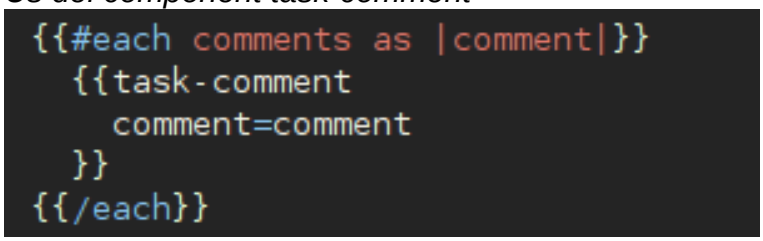


```
import Ember from 'ember';

export default Ember.Component.extend({
  classNames: ['comment']
});
```

Template amb l'HTML del component task-comment

```
<div class="comment-author-name">
  {{comment.author.username}}
</div>

<div class="comment-body">
  {{{comment.bodyHtml}}}
</div>
```

Ús del component task-comment



```
{{#each comments as |comment|}}
  {{task-comment
    comment=comment
  }}
{{/each}}
```

Una llista completa de tots els components desenvolupats per a l'aplicació q-tasks-ember es pot trobar a https://github.com/pauc/q-tasks-ember/tree/master/app/pods/components/.

4.3.3 - Altres elements

A continuació s'enumeren altres tipus d'objectes presents a l'aplicació q-tasks amb una petita explicació del seu rol:

- Models: representen les entitats persistents del sistema, amb els seus attributs i relacions, en el cas de q-tasks serien els projectes, les tasques, els usuaris, els comentaris, etc.
- Serveis: com el seu nom indica aquest tipus d'objecte proveeix serveis per a altres actors del sistema. L'accés al sistema de comunicació amb la API (persistència) de q-tasks, per exemple, es fa a partir d'un servei anomenat *store*.

- Serialitzadors: S'encarreguen de traduir les dades en JSON que torna la API a objectes (models), i viceversa.
- Adaptadors: Contenen la lògica que defineix a quina URL de la API s'ha de dirigir cada petició.
- Helpers: utilitats per a "enriquir" els *templates*. Sovint són funcions pures que donat uns arguments retornen un resultat. A q-tasks només he desenvolupat un helper per a convertir la primera lletra d'una frase en majúscules.
- Templates: Ja n'hem parlat abastament. S'encarreguen de construir l'HTML que presentarà l'aplicació.

4.4 - Comunicació entre l'aplicació client i el servidor

La comunicació entre les dues aplicacions que conformen Q-Tasks es realitza en JSON, i més concretament seguint l'especificació JSON API que es pot consultar en detall a http://jsonapi.org/format/.

Hi ha moltes maneres d'estructurar dades en JSON, i aquesta especificació defineix un seguit de normes i mecanismes per estandarditzar el formatat de dades de manera que es resolen la majoria de problemes, o almenys les necessitats més comunes, associats a aquest intercanvi de dades. En concret JSON API especifica com definir els següents conceptes:

- Identificador únic d'un recurs.
- Tipus d'un recurs.
- Atributs d'un recurs.
- Relacions amb altres recursos.
- Filtrar recursos.
- Paginació.
- Etc...

Existeixen implementacions d'aquesta especificació per a molts llenguatges, i en concret per a Q-Tasks he utilitzat la implementació per defecte d'Ember, Ember-Data, per a l'aplicació client, i ActiveModelSerializers per a l'aplicació rails.

5 - Codificació i desplegament a producció

El codi font de les dues aplicacions és públic i estar allotjat a github. Les adreces per consultar-lo o descarregar-lo són les següents:

- q-tasks-rails: https://github.com/pauc/q-tasks-rails.
- q-tasks-ember: https://github.com/pauc/q-tasqs-ember.

En les properes línies explicaré breument els aspectes claus en les metodologies de desenvolupament usades en l'etapa de codificació del projecte, així com els mecanismes i eines utilitzades per a al desplegament a producció de les aplicacions.

5.1 - Mètodes de desenvolupament

Hi ha diferències importants en com he desenvolupat la API i com he desenvolupat l'aplicació client.

Per l'experiència prèvia amb Ruby i Ruby on Rails he pogut fer un procés més directe des del disseny a la implementació. Ni l'estructura de dades ni les funcionalitats que havia de proporcionar la API presentaven dificultats importants, de manera que ha sigut força senzill traduir el disseny a codi, alhora que es codificava una petita bateria de tests.

En canvi, en el desenvolupament de l'aplicació Ember això no ha sigut així. Degut al desconeixement de l'eina el disseny era molt menys extens i precís, i el procés de codificació s'ha fet d'una manera més "explorativa", a mesura que anava aprenent a usar el framework. Un dels aspectes que més s'ha resentit d'aquesta manca d'experiència ha sigut el *testing*, ja que l'objectiu principal en el que m'he centrat en escriure els tests era "aprendre a testejar" i no tant assegurar l'estabilitat i el correcte funcionament del sistema.

5.2 - Desplegament a producció

El desplegament a producció de les dues aplicacions que conformen Q-Tasks es fa per separat.

Per a l'aplicació ember s'ha implementat la tècnica presentada per Luke Melia a la Rails Conf de 2014 - *Lightning Fast Deployment of Your Rails-backed JavaScript app*[5]. Com deia anteriorment l'ecosistema al voltant d'EmberJS és extens, i s'han aprofitat un seguit d'utilitats[6] que han facilitat establir el següent mecanisme per desplegar una nova actualització de l'aplicació client:

5 https://www.youtube.com/watch?v=QZVYP3cPcWQ
6 https://github.com/ember-cli-deploy/ember-cli-deploy

- Construir els *assets* de l'aplicació: traducció d'Ecmascript-2015 a javascript "normal".
- compressió i preprocessat dels estils.
- afegir marques per a convertir aquest *assets* en fitxers únics per a aquesta versió.
- Pujar aquest fitxers javascript, css i imatges a Amazon-S3.
- Pujar l'HTML estàtic a la base de dades Redis del servidor.

De manera semblant, per a l'aplicació Rails s'ha utilitzat Capistrano, per a automatitzar les següents tasques:

- Actualitzar el codi Ruby del servidor amb el codi de l'última version al repositori git del projecte.
- Actualitzar l'estructura de la base de dades en cas de que s'hagin inclòs canvis.
- Reiniciar el servidor de l'aplicació Ruby amb l'última versió de l'aplicació.

6 - Conclusions

En general he quedat molt satisfet del resultat del projecte, i l'objectiu principal, que era aprendre sobre aquest tipus d'arquitectura usant un client en javascript i una API rest al servidor el dono per assolit.

He après moltíssim sobre desenvolupament en Javascript, i tot un seguit de tècniques i pràctiques que crec que em poden ser molt útil en futurs projectes, i a més ho he fet disfrutant i divertint-me.

D'altra banda, els objectius pràctics que em vaig proposar en iniciar el projecte també s'han assolit: he desenvolupat un prototip per a posar a prova davant d'usuaris reals. El que no està tant clar és que les idees proposades pel que fa a la visualització i interacció amb la informació passin aquest procés d'avaluació satisfactòriament.

6.1 - Per on continuar?

A continuació es llisten algunes funcionalitats i aspectes que caldria implementar o completar si es volgués continuar el desenvolupament de Q-Tasks:

- Registre i creació d'equips i eines per a la gestió dels usuaris d'un equip.
- Sistema de privilegis i rols d'usuari, amb mecanismes d'autorització d'accions segons aquests rols.
- Mòdul de planificació de tasques a més llarg plaç.
- Enviament de dades del servidor als clients (Websockets?).
- Sistema de notificacions i avisos als usuaris davant de certs esdeveniments.
- Aplicació per a mòbils.
- Possibilitat de mencionar usuaris en els comentaris de les tasques.
- Internacionalització.
- Testing més exhaustiu de l'aplicació client.
- Testing de la pila tecnològica completa, es a dir, no només tenir tests de cada aplicació per separat sinó també tests que posin a prova la correctesa de la interacció entre els dos sistemes.

7 - Bibliografia

A continuació es presenta un recull no exhaustiu dels recursos utilitzats durant el desenvolupament del projecte:

- EmberJS Guides and Tutorials. https://guides.emberjs.com/v2.6.0/
- EmberJS API Reference. http://emberjs.com/api/
- Ruby on Rails Documentation. http://api.rubyonrails.org/
- AMBLER, Scott. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York: John Wiley & Sons, 2002.
- ERDI, Balint. *Rock and Roll with EmberJS*. 2015.
- CAMBA, Miguel. *Ember Closure Actions in Depth. 2014. http://miguelcamba.com/blog/2016/01/24/ember-closure-actions-in-depth/*
- ABERNETHY, Marin. *Ember Best Practices: Actions Down, Data Up*. 2015. https://dockyard.com/blog/2015/10/14/best-practices-data-down-actions-up
- TREACY, Frank. *Saving Models and Their Relationships with JSON API*. 23/05/2016. *http://emberigniter.com/saving-models-relationships-json-api/*
- ANICAS, Mitchel. *How To Secure Nginx with Let's Encrypt on Ubuntu 14.04*. 17/12/2015. *https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-14-04*
- WOODS, Micah. *How to write smoke tests for an Ember-Rails stack*. *https://blog.codeship.com/how-to-write-smoke-tests-for-an-ember-rails-stack*
- *ember-data: Model Lifecycle Hooks #123*. https://github.com/emberjs/rfcs/pull/123
- *ES6 modules #68*. https://github.com/emberjs/rfcs/pull/68
- RAUSCHMAYER, Dr. Axel. *Exploring ES6: Upgrade to the next version of JavaScript*. http://exploringjs.com/