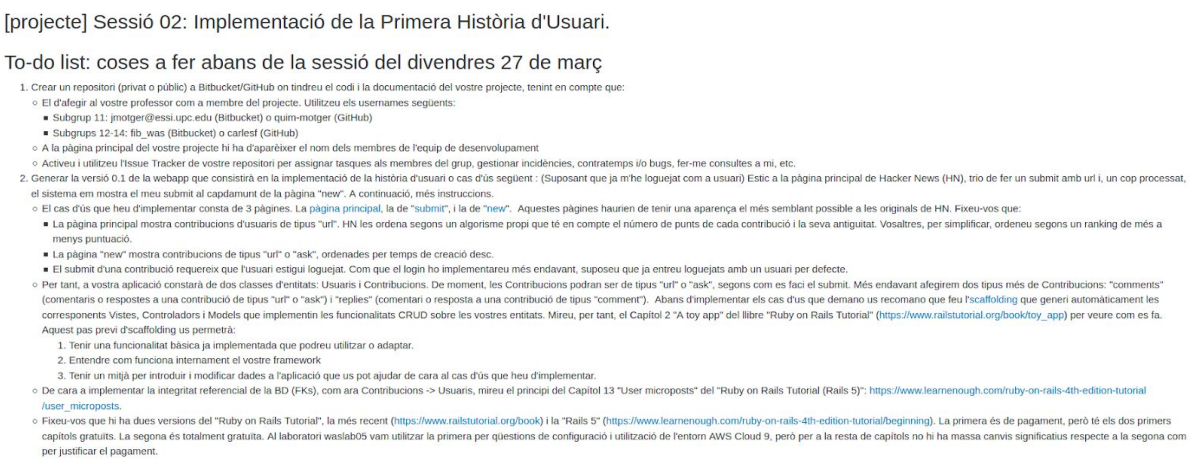
<u>Primer entrega:</u> desarrollo de una aplicación web con un framework MVC Full server-side (tanto el frontend como el backend está en el lado del servidor)**.** El framework que hemos elegido es “DJANGO”, que implementa el patrón MVC = **MTV**



[projecte] Sessió 02: Implementació de la Primera Història d'Usuari.

To-do list: coses a fer abans de la sessió del divendres 27 de març

1. Crear un repositori (privat o públic) a Bitbucket/GitHub on tindreu el codi i la documentació del vostre projecte, tenint en compte que:
   - El d'afegir al vostre professor com a membre del projecte. Utilitzeu els usernames següents:
     - Subgrup 11: jmotger@essi.upc.edu (Bitbucket) o quim-motger (GitHub)
     - Subgrups 12-14: fib_was (Bitbucket) o carlesf (GitHub)
   - A la pàgina principal del vostre projecte hi ha d'aparèixer el nom dels membres de l'equip de desenvolupament
   - Activeu i utilitzeu l'Issue Tracker de vostre repositori per assignar tasques als membres del grup, gestionar incidències, contratemps i/o bugs, fer-me consultes a mi, etc.
2. Generar la versió 0.1 de la webapp que consistirà en la implementació de la història d'usuari o cas d'ús següent : (Suposant que ja m'he loguejat com a usuari) Estic a la pàgina principal de Hacker News (HN), trio de fer un submit amb url i, un cop processat, el sistema em mostra el meu submit al capdamunt de la pàgina "new". A continuació, més instruccions.
   - El cas d'ús que heu d'implementar consta de 3 pàgines. La pàgina principal, la de "submit", i la de "new". Aquestes pàgines haurien de tenir una aparença el més semblant possible a les originals de HN. Fixeu-vos que:
     - La pàgina principal mostra contribucions d'usuaris de tipus "url". HN les ordena segons un algorisme propi que té en compte el número de punts de cada contribució i la seva antiguitat. Vosaltres, per simplificar, ordeneu segons un ranking de més a menys puntuació.
     - La pàgina "new" mostra contribucions de tipus "url" o "ask", ordenades per temps de creació desc.
     - El submit d'una contribució requereix que l'usuari estigui loguejat. Com que el login ho implementareu més endavant, suposeu que ja entreu loguejats amb un usuari per defecte.
   - Per tant, a vostra aplicació constarà de dos classes d'entitats: Usuaris i Contribucions. De moment, les Contribucions podran ser de tipus "url" o "ask", segons com es faci el submit. Més endavant afegirem dos tipus més de Contribucions: "comments" (comentaris o respostes a una contribució de tipus "url" o "ask") i "replies" (comentari o resposta a una contribució de tipus "comment"). Abans d'implementar els cas d'ús que demano us recomano que feu l'scaffolding que generi automàticament les corresponents Vistes, Controladors i Models que implementin les funcionalitats CRUD sobre les vostres entitats. Mireu, per tant, el Capítol 2 "A toy app" del llibre "Ruby on Rails Tutorial" (https://www.railstutorial.org/book/toy_app) per veure com es fa. Aquest pas previ d'scaffolding us permetrà:
     1. Tenir una funcionalitat bàsica ja implementada que podreu utilitzar o adaptar.
     2. Entendre com funciona internament el vostre framework
     3. Tenir un mitjà per introduir i modificar dades a l'aplicació que us pot ajudar de cara al cas d'ús que heu d'implementar.
   - De cara a implementar la integritat referencial de la BD (FKs), com ara Contribucions -> Usuaris, mireu el principi del Capítol 13 "User microposts" del "Ruby on Rails Tutorial (Rails 5)": https://www.learnenough.com/ruby-on-rails-4th-edition-tutorial/user_microposts.
   - Fixeu-vos que hi ha dues versions del "Ruby on Rails Tutorial", la més recent (https://www.railstutorial.org/book) i la "Rails 5" (https://www.learnenough.com/ruby-on-rails-4th-edition-tutorial/beginning). La primera és de pagament, però té els dos primers capítols gratuïts. La segona és totalment gratuita. Al laboratori waslab05 vam utilitzar la primera per qüestions de configuració i utilització de l'entorn AWS Cloud 9, però per a la resta de capítols no hi ha massa canvis significatius respecte a la segona com per justificar el pagament.

Primero configurar el entorno virtual del framework django
Antes de empezar el proyecto, primero hay que definir la metodología de trabajo. Para crear la aplicación hay que construir la estructura **scaffolding** (mirar el libre django for beginners) en django que permitirá:

1. Tener una funcionalidad básica ya implementada que se podrá utilizar o adaptar.
2. Entender cómo funciona internamente su framework
3. Tener un medio para introducir y modificar datos a la aplicación que le puede ayudar de cara al caso de uso que ha de implementar.

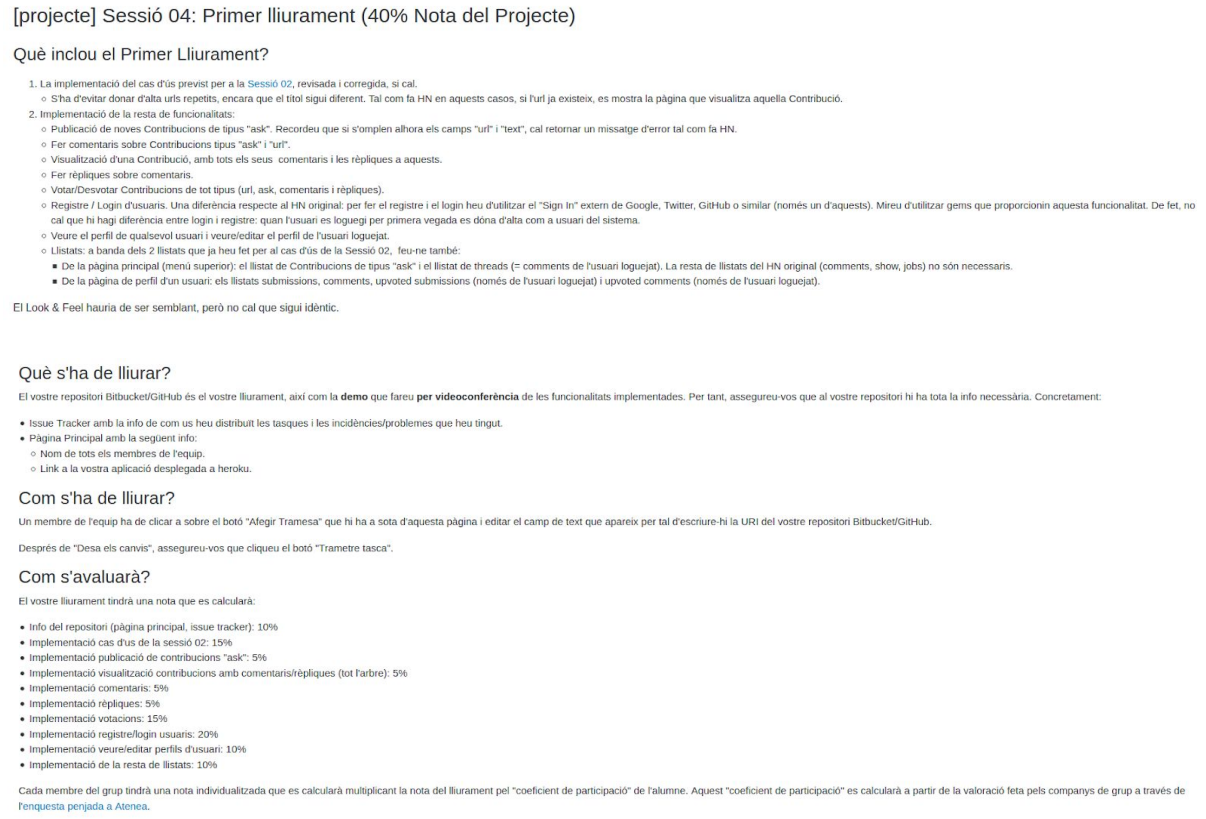
Si hay un **GET** en path del navegador ***newest (/newest),*** el archivo **url.py** mira el path e indica a qué método de la **view (controlador)** debe ir, esta comprueba o agrupar los datos necesarios accediendo al **modelo** (si es necesario) para enviar a la **template** (plantilla) y esta podrá el **render** de los datos. También hay que tener en cuenta que la plantilla (html) puede acceder al modelo para pedir los datos que necesite.

Patrón a seguir: añadir el path de la página en urls.py ⇒ definir el método asociada al path anterior en la view (p.e class AskPageView(ListView)) ⇒ crear plantilla html para mostrar los datos enviados por la view

En la implementación de la página **/ask** hay que mostrar las contribuciones de tipo ask, cuando se pulse una contribución de tipo ask debe llevar a una pagina **/item?id=x**. Este link tiene el mismo aspecto que una página de **comment** dónde

hemos descubierto cómo personalizar **los forms** (formulario) en django sin tener que usar View, como p.e TemplateView, ListView, etc.

Al hacer login, en django la clase **User** guarda todos los parámetros necesarios, como puede ser nombre, email, fecha de creación, etc.



[projecte] Sessió 04: Primer lliurament (40% Nota del Projecte)

Què inclou el Primer Lliurament?

1. La implementació del cas d'ús previst per a la Sessió 02, revisada i corregida, si cal.
   - S'ha d'evitar donar d'alta urls repetits, encara que el títol sigui diferent. Tal com fa HN en aquests casos, si l'url ja existeix, es mostra la pàgina que visualitza aquella Contribució.
2. Implementació de la resta de funcionalitats:
   - Publicació de noves Contribucions de tipus "ask". Recordeu que si s'omplen alhora els camps "url" i "text", cal retornar un missatge d'error tal com fa HN.
   - Fer comentaris sobre Contribucions tipus "ask" i "url".
   - Visualització d'una Contribució, amb tots els seus comentaris i les rèpliques a aquests.
   - Fer rèpliques sobre comentaris.
   - Votar/Desvotar Contribucions de tot tipus (url, ask, comentaris i rèpliques).
   - Registre / Login d'usuaris. Una diferència respecte al HN original: per fer el registre i el login heu d'utilitzar el "Sign In" extern de Google, Twitter, GitHub o similar (només un d'aquests). Mireu d'utilitzar gems que proporcionin aquesta funcionalitat. De fet, no cal que hi hagi diferència entre login i registre: quan l'usuari es loguegi per primera vegada es dóna d'alta com a usuari del sistema.
   - Veure el perfil de qualsevol usuari i veure/editar el perfil de l'usuari loguejat.
   - Llistats: a banda dels 2 llistats que ja heu fet per al cas d'ús de la Sessió 02, feu-ne també:
     - De la pàgina principal (menú superior): el llistat de Contribucions de tipus "ask" i el llistat de threads (= comments de l'usuari loguejat). La resta de llistats del HN original (comments, show, jobs) no són necessaris.
     - De la pàgina de perfil d'un usuari: els llistats submissions, comments, upvoted submissions (només de l'usuari loguejat) i upvoted comments (només de l'usuari loguejat).

El Look & Feel hauria de ser semblant, però no cal que sigui idèntic.

Què s'ha de lliurar?

El vostre repositori Bitbucket/GitHub és el vostre lliurament, així com la **demo** que fareu **per videoconferència** de les funcionalitats implementades. Per tant, assegureu-vos que al vostre repositori hi ha tota la info necessària. Concretament:

- Issue Tracker amb la info de com us heu distribuït les tasques i les incidències/problemes que heu tingut.
- Pàgina Principal amb la següent info:
  - Nom de tots els membres de l'equip.
  - Link a la vostra aplicació desplegada a heroku.

Com s'ha de lliurar?

Un membre de l'equip ha de clicar a sobre el botó "Afegir Tramesa" que hi ha a sota d'aquesta pàgina i editar el camp de text que apareix per tal d'escriure-hi la URI del vostre repositori Bitbucket/GitHub.

Després de "Desa els canvis", assegureu-vos que cliqueu el botó "Trametre tasca".

Com s'avaluarà?

El vostre lliurament tindrà una nota que es calcularà:

- Info del repositori (pàgina principal, issue tracker): 10%
- Implementació cas d'us de la sessió 02: 15%
- Implementació publicació de contribucions "ask": 5%
- Implementació visualització contribucions amb comentaris/rèpliques (tot l'arbre): 5%
- Implementació comentaris: 5%
- Implementació rèpliques: 5%
- Implementació votacions: 15%
- Implementació registre/login usuaris: 20%
- Implementació veure/editar perfils d'usuari: 10%
- Implementació de la resta de llistats: 10%

Cada membre del grup tindrà una nota individualitzada que es calcularà multiplicant la nota del lliurament pel "coeficient de participació" de l'alumne. Aquest "coeficient de participació" es calcularà a partir de la valoració feta pels companys de grup a través de l'enquesta penjada a Atenea.

Ha sido difícil en tratar **/edit** y **/delete** de **contribuciones**. Hay que programar con calma, primero hay que pensar y después hacer, y cuando se programa hay que pensar en otras posibles soluciones.

Un breve repaso:
en la view especificamos el nombre de la plantilla para decir dónde ha de ir para renderizar.

```
context = super().get_context_data(**kwargs)
context["current_page"] = "newest"
```

es como una variable global, lo que se guarde aquí se podrá consultar en la plantilla o en otro logar (es una manera de comunicar entre view y plantilla)

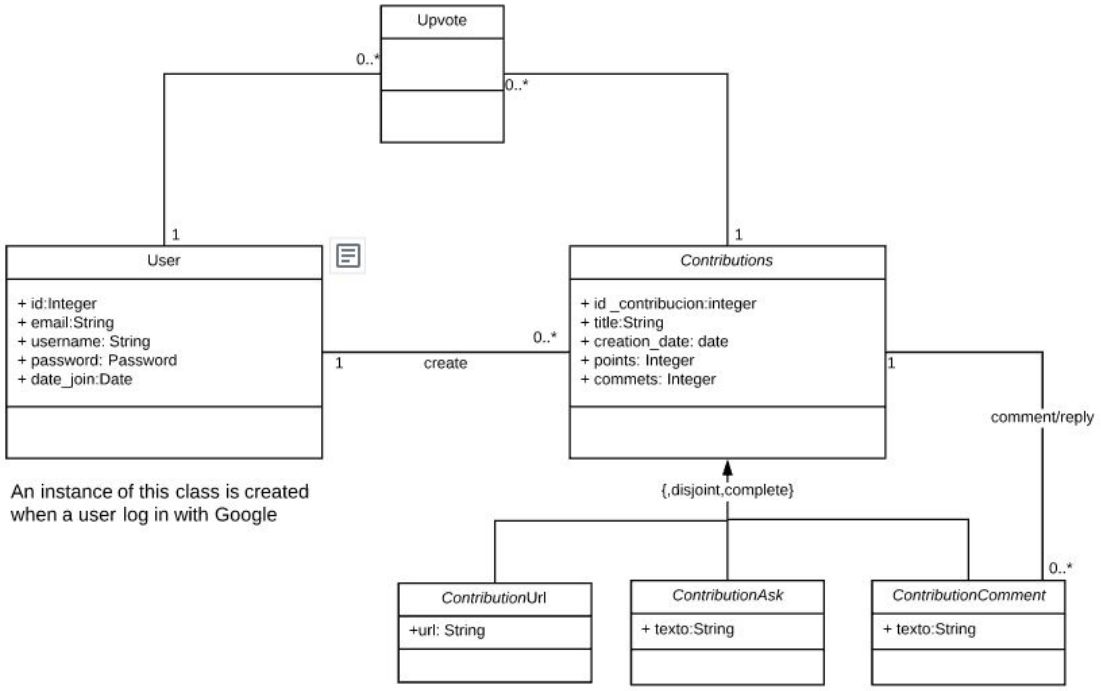
```python
class SubmitPageView(View):

    form_class = SubmitEditForm #form que voy a usar
    initial = {'key': 'value'}
    template_name = 'submit.html'

    #se llama automaticamente cuando el usuario hace un get /submit
    def get(self, request, *args, **kwargs):

        form = self.form_class(initial=self.initial)
        form.title = "Title"
        return render(request, self.template_name, {'form': form})

    #se llama cuando el usuario rellena el formulario y envia; se recoge los valores, se comprueba y se envia a la plantilla
    def post(self, request, *args, **kwargs):
        form = self.form_class(request.POST)
        #cogo todos los valores que ha introducido el usuarios en el formulario
        title = form['title'].value()
        url = form['url'].value()
        text = form['text'].value()
        author_id = form['author_id'].value()

        author_sended = User.objects.get(id=author_id)

        if author_sended == None:
            form.add_error('User does not exist', 'Please try again.')
            return HttpResponse(render(request, self.template_name, {'form': form}))

        # Posting too fast
        contributionAsk = ContributionAsk.objects.filter(author=author_id)
        contributionUrl = ContributionUrl.objects.filter(author=author_id)
        datetimeNow = datetime.now() - timedelta(hours=1)
        sub_in_last_hour = 0
        for i in contributionAsk:
            if i.creation_date > datetimeNow:
                sub_in_last_hour = sub_in_last_hour + 1
        for i in contributionUrl:
            if i.creation_date > datetimeNow:
                sub_in_last_hour = sub_in_last_hour + 1
        if sub_in_last_hour > 5:
            return HttpResponseRedirect(reverse('too_fast'))

    # Submit exceptions

    if title == "":
        form.add_error('title', 'Please try again.')
        return HttpResponse(render(request, self.template_name, {'form': form}))
    if len(title) > 80:
        form.add_error('title', "Please limit title to 80 characters. This had  " + str(len(title)) + ".")
        return HttpResponse(render(request, self.template_name, {'form': form}))

    if url != "":
        if text == "":
            try:
                urlC = ContributionUrl(title=title, url=url, creation_date=datetime.now(), author=author_sended)
                urlC.save()
            except IntegrityError as e:
                if 'UNIQUE constraint failed' in e.args[0]:
                    url_existent = ContributionUrl.objects.filter(url=url)
                    print(url_existent)
                    # form.add_error('title', 'URL already exist. Unique')
                    # return HttpResponse(render(request, self.template_name, {'form': form}))
                    return HttpResponseRedirect('/item?id=%s' % (url_existent[0].id_contribution))
                    # return redirect('item/id?' + str(url_existent.get_id()))
        else:
            form.add_error('title',"Submissions can't have both urls and text, so you need to pick one. \
            If you keep the url, you can always post your text as a comment in the thread.")
            return HttpResponse(render(request, self.template_name, {'form': form}))
    else:
        ask = ContributionAsk(title=title, text=text, creation_date=datetime.now(), author=author_sended)
        ask.save()
    return HttpResponseRedirect(reverse('newest'))
```

urls.py | views.py ● | submit.html | models.py | forms.py ×

ASW-Hackernews > hackernews > forms.py > ...

```python
class SubmitEditForm(forms.Form):
    #Used by both Submit and Edit
    title = forms.CharField(required=False, widget=forms.TextInput(attrs={'size': '50'}))
    url = forms.CharField(required=False, widget=forms.TextInput(attrs={'size': '50'}))
    text = forms.CharField(required=False, widget=forms.Textarea(attrs={'rows': '4', 'cols': '49'}))
    #Used just by Submit
    author_id = forms.CharField(required=False)
    #Used just by Edit
    id = forms.CharField(required=False)
    goto = forms.CharField(required=False)
```

Usuario hace un GET /submit ⇒ mirara en el path url.py e ira a la clase SubmitPageView, ejecutará la función **get()** y va a la plantilla submit.html en que decimos que haga render de **form** personalizado (que por ahora los campos son vacíos). Cuando el usuario rellene el formulario y de en submit `<td><input type="submit" value="submit"></td>` (aqui el type tiene que ser “submit”), entonces ira a la función **post()**, el controlador recoge los datos introducidos por el usuario y comprueba las restricciones y comunica con submit.html para hacer render.



RT1 : PK :User(id),Contribution(id_contribution)
RT2: Un usuario puede votar sólo una vez a una contribución.

Feedback de la primera entrega:

A) Info. del repositori (10):	10				
B) Implementació cas d'us de la sessió 02 (15):	15				
C) Implementació publicació de contribucions "ask" (5)	5				
E) Implementació visualització contribucions amb comentaris/rèpliques (tot l'arbre) (5)	5				
F) Implementació comentaris (5)	5				
E) Implementació rèpliques (5)	5				
F) Implementació votacions (15)	10	No hi ha persistència en upvoted comments. Fixeu-vos què passa si faig refresh de la pàgina; o si vaig a la meva llista de upvoted comments.			
G) Registre/login (20)	20				
H) Veure/editar perfils d'usuaris (10)	8	No hi ha el camp e-mail			
I) Implementació de la resta de llistats (10)	8	A la vista threads no tots els icones de vote apareixen correctament: sembla que pugui votar comentaris propis.			
TG) Total Grup (sobre 100):	91	Bona feina en general, detalls menors			
Tanvir Hossain		C1	C2	C3	C4
V) Valoració companys ((C1+C2+C3)/60):	1	15	15	15	15,00
Nota Alumne (TG*V):	9,10				

Segunda entrega: Diseño e implementación de una API REST

Hasta ahora lo que hemos es construir una aplicación web desde principio. Hemos desplegado la app en un servidor remoto (heroku) para que la gente pudiera entrar y usarla. Nuestro objetivo final es que el usuario pueda comunicarse con la app mediante el uso del protocolo http. Para ello, en esta entrega primeramente vamos a implementar una API REST de nivel 2 y documentarlo en formato yaml.

Richardson Maturity Level (RMM)

- Level 0: una sola URI, un solo tipos HTTP request (generalmente POST)
- Level 1: múltiples URIs, un solo tipos HTTP request
- **Level 2: múltiples URIs, múltiples tipos de HTTP request (GET, POST, PUT, DELETE)**
- Level 3: múltiples URIs, múltiples tipos de HTTP request, usa HATEOAS

Los objetivos de esta entrega:

- Exponer las **mismas funcionalidades** que la aplicación web de la primera entrega (excepto login)
- Representaciones de recursos y formatos de datos a peticiones PUT y POST seran en **JSON**
- Documentación API en base a **OpenAPI Specification** en formato yaml (v2.0 o superior)
- **Swagger editor** para diseñar y probar API

Por una parte, hemos hecho el setup de swagger y por otra parte hemos creado la API (definir los endpoints), las rutas de api son diferentes a las de web (que definimos en la primera entrega). Por ejemplo, antes teníamos /ask que nos llevaba a la página web de ask (html), y ahora definimos **/api/asks** que nos devolverá el listado de ask en formato JSON. Es un poco lo que hicimos con twitter (lab4), cuando hacíamos un GET nos devolvía un listado de tweets e incluso informaciones adicionales pensando que un futuro las vamos a necesitar. También hemos configurado CORS para no tener problemas de acceso en otros dominios que no fueran al nuestro (el link que se crea cuando desplegamos).

La prueba de api se hace desde el editor swagger; es como postman. Hay que hacer la configuración o integración de api rest con swagger. Hay varias formas de documentar. Python puede generar automáticamente el documento swagger o rdoc. Pero en esta entrega se ha de hacer manualmente en yaml. Para que haya esta sincronización se ha de desplegar el trabajo en hereku y para que el swagger pueda hacer peticiones y trabajar sobre esta página (hay que indicar el link de heroku en el swagger).

Usamo app swaggerhub con cuenta de github (tanvir08)

Feedback de la segunda entrega:

A) Info. del repositori (10):	10	https://github.com/arle21/ASW-Hackernews			
B) Disseny i documentació API (45):	25	- Heu separat com a 4 entitats independents ASK, URL, COMMENTS i REPLIES de manera confusa. REPLIES i COMMENTS no cal diferenciar-los: una reply és un comentari d'un comentari. Afegir un controlador específic resulta redundant i confús. Com a mínim, haurieu de tenir /api/urls/{id}/comments/{commentId}/replies/{replyId}, però tot i així resultaria innecessari. A més heu separat ASK i URL però després a VOTES els tracteu de manera uniforme com a CONTRIBUTIONS. Pel que veig en el cas dels atributs, a nivell de model de la webapp heu treballat amb una superclasse CONTRIBUTION però la vostra API presenta inconsistències entre uniformitzar aquesta classe o tractar URL i ASK per separat. - Separar GET /api/asks/{id} & ../completeDescription és redundant (penseu en quin escenari farieu servir la 1a i no la 2a) - Vistes implementades parcialment			
C) Implementació API (45)	40	- Operacions PUT retornen "Success" en comptes de l'entitat modificada - Vistes implementades parcialment La implementació general OK			
TG) Total Grup (sobre 100):	75				
Tanvir Hossain		C1	C2	C3	C4
V) Valoració companys ((C1+C2+C3)/60):	1	15	15	15	15
Nota Alumne (TG*V):	7,50				