ORARIOLEZIONI

Progetto di Linguaggi Dinamici

Nicolò Cavedoni, Informatica 2016/17

1. Indice

2. Presentazione dell'applicazione

2.1 Anagrafica

2.1.1 Nome dell'applicazione:
Orariolezioni

2.1.2 Sviluppatore:
Nicolò Cavedoni

2.1.3 Ambiente di esecuzione:
Applicazione Web

2.1.4 Linguaggi di sviluppo:
Python e HTML principalmente, anche Javascript e CSS

2.1.5 Framework:
Django

2.1.5 Database:
PostgreSQL

2.2 Breve introduzione
Orariolezioni è un'applicazione web che si occupa di calcolare in automatico l'orario delle lezioni di una facoltà universitaria. Ogni utente del sito può visualizzare gli orari di lezione della propria e di tutte le facoltà che aderiscono al sistema, in modo semplice ed immediato.

2.3 Terminologia

Admin	Amministratore del sito
Reg	Utente registrato al sito, corrisponde a un docente nella vita reale
Anon	Utente anonimo, tipicamente uno studente nella vita reale

2.4 Come utilizzare l'applicazione
Leggere il file README.txt per installare correttamente l'applicazione. Leggere il resto della documentazione per impararne il contenuto.

3. Specifiche

T6**) Sistema di generazione automatica dell'orario delle lezioni**
Creare un sito per la generazione automatica dell'orario delle lezioni
- Ogni utente registrato e con diritti da amministratore deve poter inserire (attraverso l'interfaccia di amministrazione) i dati relativi ai corsi (numero di ore settimanali, vincoli – es. uso di laboratorio - numero di studenti), ai docenti ed i loro desiderata rispetto agli orari da ricoprire, e calcolare automaticamente un orario compatibile
- Ogni utente registrato come docente deve poter visualizzare l'elenco dei propri orari di insegnamento
- Ogni utente anonimo deve poter visualizzare la situazione complessiva dei corsi del semestre in corso
- Gli orari devono essere automaticamente calcolati dal sistema in modo da rispettare i desiderata inseriti ed gli altri vincoli specifici

Facoltativo (obbligatorio se svolto in 2 persone)
Valutare più alternative diverse (almeno due) di algoritmo per generare l'orario rispettando i vincoli e farne una valutazione in termini di prestazioni

---

3.1 Punti fermi

Dalla lettura delle specifiche risultano chiare le seguenti direttive:

- Ci sono tre diversi attori del sistema: l'amministratore (*admin*), l'utente registrato come docente (*reg*) e l'utente anonimo (*anon*).
- Ogni admin agisce tramite interfaccia di amministrazione e inserisce i dati nel sistema. I dati sono relativi ai corsi, ai docenti e ai desiderata dei docenti.
- Ogni reg ha la possibilità di consultare il proprio orario personale.
- Ogni anon ha la possibilità di consultare l'orario dei corsi.
- Entrambi gli orari devono essere calcolati da un algoritmo che rispetti i desiderata dei docenti.

3.2 Punti interpretabili => interpretazioni

Dalla lettura delle specifiche sorgono invece questi dubbi:

- Dati relativi ai corsi? => Un corso universitario ha un nome, un docente, una facoltà, un certo numero di crediti e di ore di lezione. Alcune di queste ore possono essere dedicate alla pratica, quindi ha anche delle possibili ore di laboratorio. Inoltre è importante contare il numero di studenti partecipanti al corso per assegnarlo a un'aula sufficientemente capiente.
- Dati relativi ai docenti? => Un docente ha nome e cognome, insegna un corso, ma può anche insegnare più corsi in più facoltà. Ipoteticamente potrebbe anche non tenere alcun corso e fare comunque parte dello staff universitario.
- Dati relativi ai desiderata? => Innanzitutto, cos'è un desiderata? È un insieme di richieste che un lavoratore fa al suo superiore, nel nostro caso un docente nei confronti dell'università. Poiché il tema del progetto è il calcolo degli orari di

lezione, scelgo che i desiderata riguardino solo giorni e ore della settimana lavorativa (che va da Lunedì a Venerdì).
Rimane comunque il problema della specificità delle richieste: se venisse chiesto a un docente di compilare i suoi desiderata nello specifico di ore e giorni della settimana, risulterebbe incredibilmente difficile modellare un algoritmo sullo stato dell'insieme delle richieste, per via dell'enorme varianza delle condizioni di partenza. Risulta invece più semplice "generalizzare" il raggio d'azione dei desiderata, concedendo ai docenti di esprimere le loro preferenze meno nel particolare. Decido quindi di includere nei desiderata le richieste di *giorno libero*, *giorno impegnato*, *prima delle nove* e *dopo le cinque*. Per semplicità ho quindi abbandonato l'idea iniziale di creare un'entità Desiderata unica per ogni docente e includere le quattro richieste nei campi dell'entità Docente.

- L'admin deve calcolare automaticamente un orario compatibile? => Questa specifica mi lascia un po' dubbioso, in quanto conferisce all'admin la capacità ambigua di calcolare un orario di lezioni compatibile con tutti, ma farlo in automatico (l'admin è un utente, non un computer). Interpreto quindi che non sia una specifica legata alle azioni dell'admin, ma che sia invece una precisazione riguardo l'utilità dei desiderata, che servono a indicare all'algoritmo come calcolare l'orario finale in automatico.
- Che relazione c'è tra un utente registrato come docente e un docente nel database? => La soluzione più immediata è che un reg possa registrarsi anche come elemento del database, coi suoi dati e i suoi desiderata. Questo però concederebbe a un normale utente abilità esclusive di un superutente (admin), quali le modifiche al database, il che va contro le specifiche. Scelgo quindi di associare ogni reg a un docente nel database, al momento della registrazione.
- Un reg può visualizzare l'orario generale dei corsi? => Non vedo perché no, anche se non è esplicitato.
- Un anon può visualizzare l'orario dei docenti? => Questa volta no. Innanzitutto verrebbe a mancare l'idea di "orario personale", rendendo l'orario dei docenti un secondo orario generale. Secondo e più importante, verrebbe a mancare la distinzione tra utente registrato e utente anonimo, che invece è chiara.
- Esistono utenti registrati che non siano reg o admin? => No, non mi sembra che aggiungerebbero niente al sistema. Ogni utente al di fuori degli admin che si registra al sito è per forza un reg (ovvero un docente nella vita reale).

3.3 Punti aggiunti in fase di interpretazione

- I vincoli di un corso descritti al paragrafo precedente. Questi comportano:
- L'aggiunta di *Facoltà* al sistema. Ogni facoltà ha semplicemente un nome e un codice identificativo unico. A posteriori noto che avrei potuto suddividere l'idea di facoltà da quella del dipartimento fisico in cui locare le aule, avrebbe reso meglio l'idea di facoltà diverse accorpate geograficamente (magari che

condividono aule). Con esse arriva la distinzione degli orari per ogni facoltà, portando quindi una lista di orari disponibili ogni semestre, e non uno soltanto.

- L'aggiunta di *Aule* al sistema. Ogni aula appartiene a una facoltà (intesa anche come dipartimento), ha un codice identificativo unico, una capienza massima e può essere o non essere un laboratorio. Con esse arriva l'aggiunta delle aule all'interno dell'orario di lezione (si veda […]).
- I campi *email* e *registration key* all'entità Docente. Il campo email è fondamentale per l'associazione reg-docente al momento del signup e la registration key è un artificio che impedisce a un visitatore qualsiasi di registrarsi come docente. Ipoteticamente questa chiave viene distribuita da un admin al corpo dei docenti prima della registrazione, consegnando a ciascuno la chiave di registrazione necessaria ad associare il suo account con un elemento Docente nel database.

4. Implementazione

Orariolezioni è un'applicazione costruita sul framework Django, e fa parte di un progetto denominato "ProgettoLD". È importante evidenziare questa distinzione, in quanto alcune delle funzionalità implementate (tipo i meccanismi di login, logout, registrazione di un utente…) non fanno effettivamente parte dell'applicazione in sé, ma sono generiche all'interno del progetto, e quindi disponibili anche a future applicazioni affiliate allo stesso progetto.
Da qui una seconda importante precisazione: *admin, reg* e *anon* sono utenti del sito generato da ProgettoLD, di cui fa parte anche Orariolezioni. Ne consegue che l'estendibilità del progetto in futuro con altre applicazioni "sorelle" implicherà una revisione del sistema degli utenti, attualmente pensati per rispettare le specifiche di Orariolezioni.

4.1 Filesystem

4.1.1 manage.py:

Eseguibile python creato di default da Django. Permette allo sviluppatore di eseguire operazioni predefinite sul progetto, quali:

- *makemigrations* e *migrate*. Entrambe eseguono operazioni sul database, applicando le modifiche non ancora salvate
- *runserver*. Avvia il server di prova in locale
- *createsuperuser*. Crea un admin per il sito
- *sampledata*. Popola il database con dei dati iniziali (comando non di default nella suite Django)
- *newuser*. Crea un nuovo reg inserendo una delle registration key trovate nel database, nel codice o nel file README (comando non di default)

4.1.2 ProgettoLD:

Cartella di progetto Django. Contiene i file che regolano il comportamento generale di ogni applicazione Django creata allo stesso livello, quale orariolezioni ma anche possibili applicazioni future.

4.1.3 templates:

Cartella che contiene i template HTML generali, utilizzati per le funzioni di base di ogni applicazione Django.

4.1.4 orariolezioni:

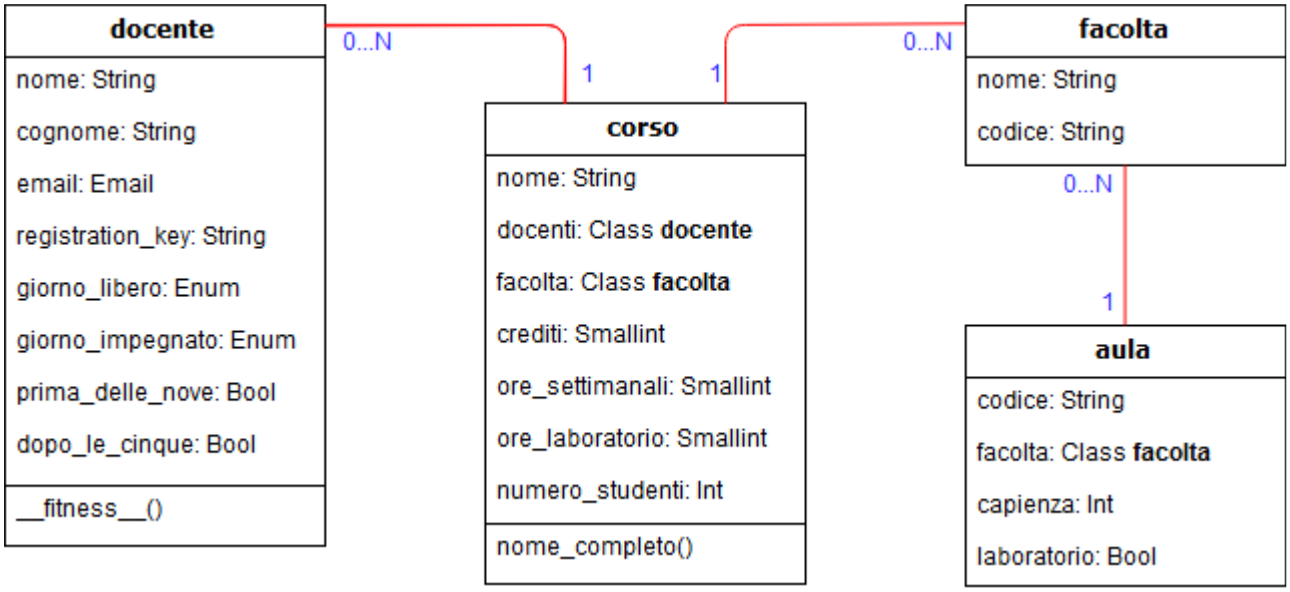
La directory dell'applicazione in questione. Contiene:

- Una cartella migrations di default che ospita le migrazioni del database
- Una cartella templates con i template HTML relativi a Orariolezioni
- Una cartella management con i due comandi aggiunti sampledata e newuser, invocabili da manage.py
- *__init__.py* e *apps.py* di default

- I restanti file python di configurazione, il cui funzionamento verrà illustrato nelle pagine seguenti

4.2 Diagramma delle classi

Tutte le entità partecipanti a Orariolezioni hanno una classe di tipo Model all'interno del file *models.py*. Ho avuto problemi di integrità con il database postgres dovuti all'errata referenziazione del nome dei model nel database, che veniva trasformato in minuscolo producendo errori. L'ho risolto creando dei Model con la prima lettera minuscola, e anche senza accenti, come nel caso di "facolta". Ad ogni modo la cosa non influenza l'output finale dell'applicazione, è solo un difetto a livello di ingegneria del codice.



Ognuna di queste classi è un Model in Django, dunque il tipo di dato di ogni valore non è propriamente quello indicato nel diagramma, ma una sua estensione specifica del framework.

4.2.1 Classe docente:

Può essere istanziata senza problemi di referenze, sia per scelta tecnica che per rispecchiare il caso (possibile) in cui un docente non abbia all'attivo alcun insegnamento per quel periodo. I campi email e registration_key sono chiavi primarie della classe. Strano ma vero, lo è anche la coppia nome + cognome, per motivi che si vedranno al punto 4.4.2. giorno_libero e giorno_impegnato consentono solo di scegliere 6 opzioni corrispondenti ai cinque giorni lavorativi della settimana e 'N.D' che significa che è indifferente quale sia il giorno libero o quello impegnato del docente. __fitness__() invece è una funzione che calcola la fitness statistica del docente nel momento in cui deve competere con altri docenti per avere la priorità nel rispetto dei desiderata. Il

valore è calcolato ordinando sempre per primi i docenti che tengono più corsi, qualsiasi siano le loro richieste, a seguire quelli che hanno avuto "meno pretese" a parità di corsi insegnati. Poiché il calcolatore di orario non sovrascrive ore di lezione già assegnate, ne consegue che il primo docente ad essere controllato (quello con più fitness) avrà tutti i desiderata soddisfatti.
Chiaramente, un docente non può chiedere lo stesso giorno libero e giorno impegnato, sarebbe un controsenso, e viene impedito all'atto di validazione dell'oggetto nel database.

4.2.2 Classe facolta:

L'unica cosa degna di precisazione è che il codice deve essere scritto nella forma "AAA-00" e deve essere unico. La lunghezza fissa del codice permette di sapere quanti caratteri troncare all'atto della visualizzazione dell'orario di un corso, che altrimenti verrebbe stampato col suo __nome_completo__(), esteticamente più brutto.

4.2.3 Classe aula:

Il codice dell'aula può essere di massimo 5 caratteri e la capienza deve essere di almeno 25 posti. Entrambe per motivi estetici e di modellazione verosimile.

4.2.4 Classe corso:

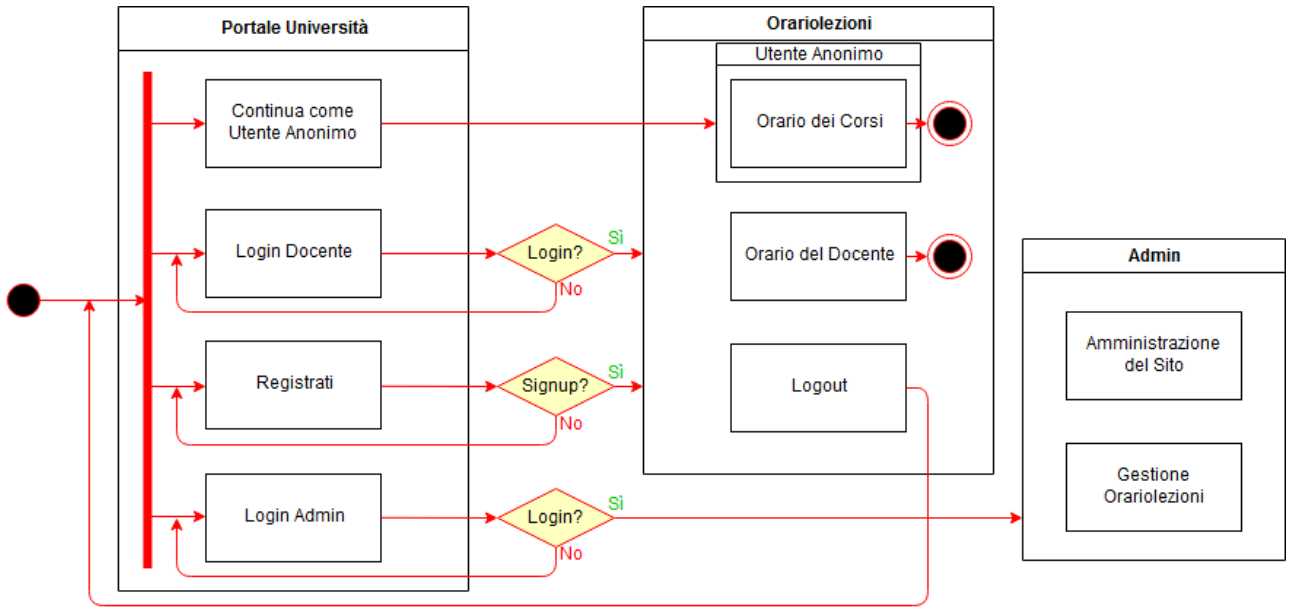
La chiave primaria della classe è costituita dal suo __nome_completo__(), ovvero la combinazione di nome e facolta. All'atto di validazione del Model (prima di inserire un oggetto nel database), viene controllato che le ore_laboratorio non siano superiori alle ore_settimanali, non avrebbe senso. Viene anche controllato che le ore_settimanali non siano troppe rispetto a quelle disponibili in una settimana (nonché 50, 10 ore al giorno), controllando anche le ore settimanali degli altri corsi della stessa facolta. Infine viene controllato che in quella facoltà vi sia almeno un aula/laboratorio che possa contenere tutti gli studenti del corso.

4.2.5 La questione "numero docenti partecipanti allo stesso corso":

So bene che è possibile che un corso sia tenuto da più di un docente, anzi, accade abbastanza spesso. Tuttavia questo problema rendeva enormemente più complessa la valutazione delle condizioni necessarie per il completamento dell'algoritmo a livello della distribuzione dell'orario dei docenti; dopo tentativi diversi non sono comunque riuscito a ottenere un algoritmo che fosse robusto e che spartisse l'orario dei docenti in modo decente, e mi sono trovato costretto a semplificare il modello dei dati rinunciando a un po' di veridicità.

4.3 Diagramma delle attività

Come salta subito all'occhio, ci sono tre macro-aree di interesse (quattro se consideriamo la sotto-area "Utente Anonimo") che rappresentano le tre schermate principali di tutto il progetto.



4.3.1 Portale Università:

Quest'area (e la successiva 4.3.2) è gestita e generata dai file nella cartella ProgettoLD e quindi è relativa all'intero progetto e non specificatamente a Orariolezioni. Future applicazioni (di carattere scolastico, si pensa) avranno la stessa homepage di login e registrazione al sistema, anche se chiaramente bisognerà controllare le redirezioni del controller, al momento settate per re direzionare verso l'area Orariolezioni (4.3.4). L'URL della pagina iniziale è il classico **" / "**.

Le azioni consentite all'utente del sito sono:

- Login come Docente. Autenticando il proprio account, un docente può accedere ad Orariolezioni come reg, e quindi visualizzare anche il proprio orario personale, oltre a quello generale delle lezioni. Se il login fallisce, all'utente sarà chiesto di ritentare. L'URL corrisponde a **"/login"**.
- Login come Amministratore. Autenticando il proprio account, un admin può accedere al pannello di amministrazione (4.3.2). Se il login fallisce, all'utente sarà chiesto di ritentare. L'URL corrisponde a **"/admin"**.
- Continuare come Utente Anonimo. Cliccando su questo link, un utente può accedere ad Orariolezioni come anon, visualizzando quindi soltanto l'orario generale di ogni facoltà. È possibile che le credenziali di un utente registrato vengano memorizzate nella sessione in corso nonostante esca dal sito, e quindi acceda ad Orariolezioni come reg

nonostante usi questo link. Anche se non era preventivato, non ho ritenuto necessario correggere il problema, può addirittura rivelarsi comodo per la navigazione. Questo link manda a **"/orariolezioni"**, indipendentemente che l'utente sia reg o anon.

- Registrarsi come docente. Vedi 4.3.3.

4.3.2 Admin:

L'interfaccia di amministrazione è quella di default di Django (in lingua italiana), ma la parte relativa all'applicazione è lievemente personalizzata all'interno del file *admin* di orariolezioni. Non ho ritenuto necessario creare azioni di amministratore particolari, in quanto per far funzionare l'applicazione non serve altro che inserire oggetti nel database, compito che non può essere facilitato più di così. In compenso ho reso più apprezzabile il layout della piattaforma.

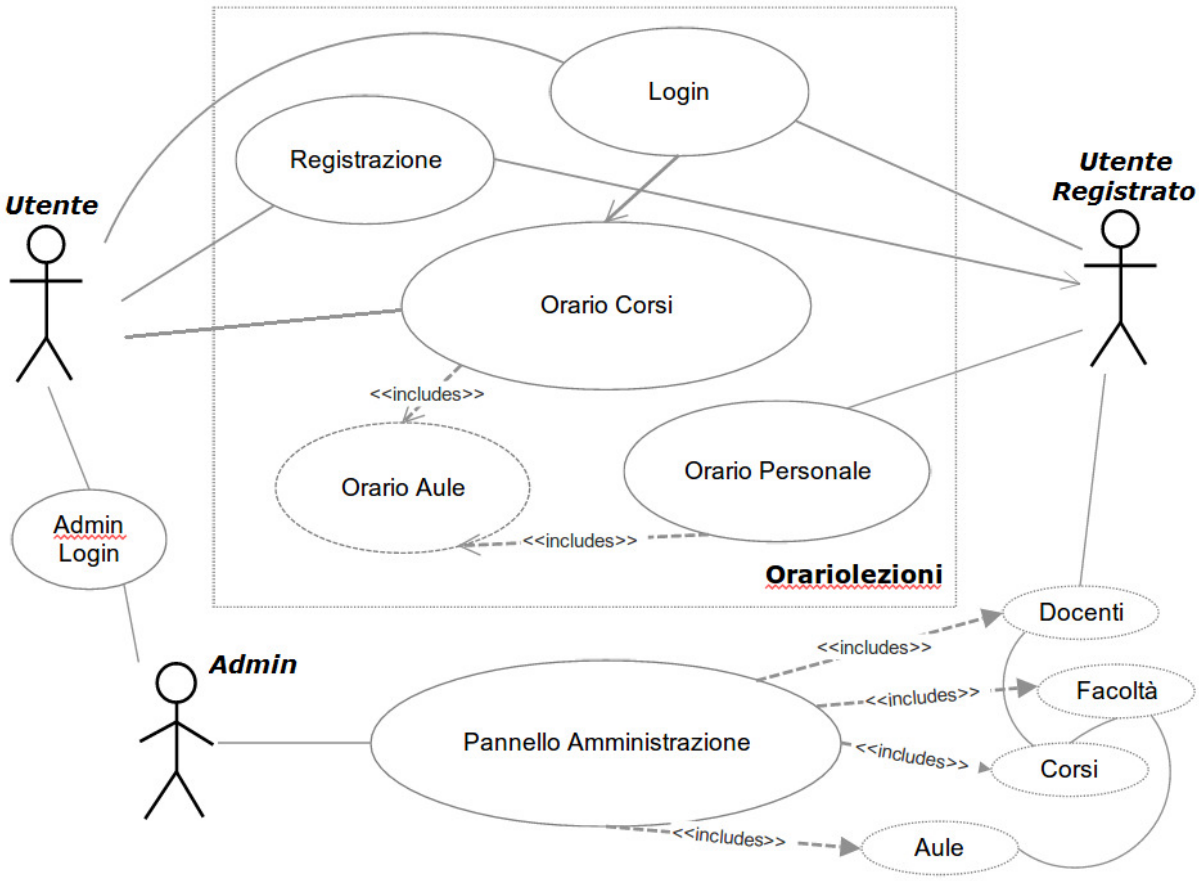
4.3.3 Registrazione:

La registrazione di un utente come docente avviene in questa pagina, all'URL **"/signup"**. L'utente inserisce le tipiche credenziali di creazione di un account, e può aggiungere opzionalmente il suo Nome e Cognome. Oltre al classico check dell'integrità della password (oltre a una serie di controlli di semplicità di default), il sistema controlla la validità dell'indirizzo email inserito nel campo obbligatorio *Email* e della registration key inserita nel campo obbligatorio *Chiave di Registrazione*. Qui entra in gioco un ragionamento strutturale: per funzionare, il sistema degli account ha bisogno di un'asserzione esterna al software, ovvero che l'amministratore di sistema distribuisca in modo univoco le chiavi di registrazione ai docenti, così come la mail universitaria. Per questo motivo si assume vera la condizione che ogni docente conosca solo la sua mail e la sua registration key, e ogni utente malintenzionato non possa conoscerle entrambe. Questa assunzione permette di non dover controllare la validità della coppia Email-Chiave, lasciando la briga dell'univocità a chi amministra il sistema. E poiché l'assegnazione dell'account a un oggetto docente nel database avviene tramite il campo email (vedi 4.2.1), è possibile per l'admin gestire le registration key in modo differente da quello di default, tipo impostandone una uguale per tutti (quindi scalabile) o distribuendole casualmente da una lista, senza preoccuparsi di quale docente ha quale registration_key.

4.3.4 Orariolezioni:

Giunto a questa pagina ("/orariolezioni"), l'utente dispone di una lista di tutte le facoltà nel database, per consultarne l'orario semestrale con un click. Cliccando su un elemento della lista, si avrà un URL del tipo **"/orariolezioni/id"** dove id = facolta_id, uno dei campi predefiniti di un Model Django. Un utente registrato può anche visualizzare il suo orario personale, che avrà sempre e comunque URL **"/orariolezioni/orario_personale"** e verrà

caricato controllando il campo email relativo all'account invocante. Per maggiori dettagli sulla costruzione di queste due view, consultare 4.4.



4.4 View e Controller

In questo paragrafo si illustrano le scelte implementative contenute nei files *views.py* e *urls.py* sia di ProgettoLD che di orariolezioni.

4.4.1 ProgettoLD – views e urls:

Il controller del sistema è piuttosto semplice, e associa /login, /logout e /admin alle view di default di Django.auth. L'URL /orariolezioni è gestito dal controller dell'applicazione Orariolezioni e verrà spiegato in seguito. L'unico vero elemento di personalizzazione (oltre a qualche semplice template aggiunto) è costituito dalla classe SignUpExtendedForm nel file *forms.py*, che aggiunge al layout di registrazione di default i campi descritti al punto 4.3.3, assegnando al nuovo utente il campo Email inserito (fondamentale poi per visualizzare correttamente l'orario personale).

4.4.2 Orariolezioni – views e urls:

Il controller dell'applicazione deve gestire solamente quattro tipi di URL:

- /orariolezioni è l'URL della pagina principale dell'applicazione. La view collegata si chiama *index* e inserisce nel template HTML la lista delle

facoltà, ordinate per id. Questo permette al template di disporre degli oggetti di classe facolta e stamparne i nomi in una lista ordinata. È risultato fondamentale ordinare la query per id, dal momento che l'ordinamento di default di objects.all() è per data di ultima modifica, e questo avrebbe delle conseguenze sbagliate sugli orari contenuti in ciascun link fornito.

- /orariolezioni/logout rimanda l'utente alla pagina principale del sistema (l'area Portale Università al punto 4.3), annullandone l'accesso tramite la view di default.
- /orariolezioni/orario_personale rimanda alla view *orario_docente*. Il decoratore @login_required impedisce l'accesso a questa view da qualsiasi utente che inserisca forzatamente l'URL, costringendolo ad autenticarsi. Inoltre, se l'utente è registrato ma non è assegnato a nessun docente (tipo l'admin stesso o qualche utente creato da pannello di amministrazione saltando la procedura di signup), la view ritorna un errore 404. Viene quindi avviato il calcolatore degli orari (4.5) e dai suoi risultati viene filtrato quello relativo all'orario del docente ottenuto dal controllo via email. Qui si spiega il motivo della forzatura nome+cognome unici nel database: l'orario corrispondente viene cercato tra tutti gli orari dei docenti controllandone la prima cella, che contiene appunto nome e cognome del docente, e viene passata al template HTML l'intera tabella, una volta trovata, assieme alla corrispondente tabella aule. **NB**: sono consapevole del fatto che ci fossero soluzioni anche semplici a questo problema, come ad esempio far scrivere al calcolatore il campo email (già unico di suo) nella prima cella dell'orario personale, e poi sostituirlo con nome e cognome una volta ottenuto quello giusto dall'elenco. Purtroppo la deadline mi ha impedito di migliorare questo punto, che in ogni caso funziona.
- /orariolezioni/<facolta_id> è invece l'URL principale di tutto il sistema e la sua view si comporta in maniera molto simile a quella appena descritta, con la differenza che è accessibile a tutti gli utenti e prende in ingresso un intero (che poi sarebbe una String, nell'URL, ma parliamo di python dopotutto), che userà per trovare il corretto orario di facoltà nella lista ritornata dal calcolatore. Per un layout più bello degli orari, al template viene passata una tabella modificata che contiene i nomi dei corsi sottratti di 9 caratteri, ovvero "_/_AAA-00" che corrispondono al codice di facoltà relativo al corso. Il calcolatore ha bisogno di stampare il __nome_completo__() del corso, in quanto questo è sicuramente unico e non ci sono pericolose sovrapposizioni di nomi (tipo un corso di Statistica o di Inglese presente in più facoltà).

4.4.3 Il template orario.html:

È il template più importante dell'applicazione (nonché praticamente l'unico) e merita un sottoparagrafo dedicato. Si presenta come una semplice HTML

Table, abbellita con CSS. Per chiarezza del codice e maggiore semplicità, le celle della tabella sono hard-coded e non sono generate programmaticamente con l'ausilio di javascript. La prima riga e la prima colonna sono dedicate rispettivamente ai giorni della settimana e alle ore di lezione, colorate diversamente e non cliccabili. La cella (0,0) invece contiene il nome della facoltà o del docente passato in ingresso. Ogni altra cella è dedicata sia a un corso che alla rispettiva aula in cui si terrà, e con un semplice click l'utente può cambiare il contenuto e il background della cella visualizzando a scelta se l'aula o il corso relativi a quell'ora. Questa funzionalità è implementata in-code con javascript.

4.5 Il calcolatore automatico di orari:

Per ultima lascio la spiegazione più importante, ovvero l'implementazione dell'algoritmo che genera automaticamente gli orari delle lezioni e delle aule. Questo può essere trovato all'interno di orariolezioni/*methods.py*.
La funzione che genera tutti gli orari delle facoltà e dei docenti prende il nome di *crea_orari()* ed è quella che viene invocata dalle view ogni volta che c'è bisogno di stampare un orario. Il primo passo consiste nell'ordinare i docenti in base alla loro __fitness__(), per passarli al calcolatore in ordine di importanza. Poi crea le tre List che conterranno gli orari di ogni facoltà (e una per le aule) e ogni docente, e a queste assegna i valori iniziali ritornati da *orariovuoto(titolo)*, al quale viene passato in ingresso il nome che si vuole visualizzare nella cella (0,0) della tabella.
Le tre liste diventano così liste-di-tabelle, ovvero array tridimensionali o matrici tridimensionali, contenenti tutte la stessa tabella inizializzata con la legenda "riga 0 = giorni della settimana" e "colonna 0 = ore di lezione giornaliere". Prima di entrare nell'algoritmo si crea infine una List aule_assegnate[], che servirà per ricordarsi quali aule sono già state utilizzate per corsi scritti in precedenza e assegnarne di diverse, se possibile.
Detto questo, il calcolatore inizia a scorrere la lista ordinata dei docenti e per ognuno di questi scorre la lista dei corsi da questo tenuti. Si ha quindi una copertura totale dei corsi di lezione, poiché non può esistere un corso senza docenti (e comunque non sarebbe nemmeno insegnato) e non può esistere un corso con più di un docente, che verrebbe quindi calcolato più volte producendo errori.
Il vero algoritmo di calcolo è lasciato alla funzione *inserisci(d,c,t,o)* che come suggerisce il nome non fa altro che inserire il corso *c* tenuto dal docente *d* nelle tabelle *t* e *o*, rispettivamente orari della facoltà e del docente. Come lo fa?

- Innanzitutto salva i desiderata del docente in alcune variabili e crea un contatore di ore_rimaste che diminuisce ogni volta che una o più ore vengono assegnate. Inoltre crea una List coordinate[] che conterrà la posizione delle celle a cui è stato assegnato un corso e un'altra List giorni_usati[] che si occuperà di tenere il conto di quali giorni sono già stati impiegati per le lezioni del corso (includendo fin da subito il giorno libero).
- Da qui inizia una snervante e ripetitiva serie di controlli passando al setaccio ogni volta la tabella (saltando sempre la prima riga e colonna) e controllando

che la cella, o le celle in caso volesse inserire un blocco di due ore, in questione sia libera.

- Per controllare le condizioni, si usa la funzione *check(tab,row,column)* che molto banalmente controlla che non ci sia già qualche altra lezione in *tab[row][column]*. Oltre a salvare molto della lunghezza del codice, permette anche di usare un solo blocco di codice per controllare sia l'orario della facoltà che quello del docente (magari ha già un'altra lezione in un'altra facoltà in quel momento). Generalmente poi c'è anche qualche condizione legata alle celle adiacenti in colonna, quando si cerca di inserire un blocco di 2 ore o di aggiungere un'ora a un blocco di lezione già consolidato dello stesso corso.
- Come già detto, questi controlli vengono fatti a ripetizione, ogni volta perturbando un po' le condizioni di partenza in modo da testare se con le nuove condizioni si apre qualche spiraglio per esaurire il contatore ore_rimaste. Questo metodo consente quindi di rispettare i desiderata **il più possibile**, ma non in maniera totale, e chiaramente i risultati vanno peggiorando con l'aumentare del numero di docenti che insegnano nella stessa facoltà, o di ore di lezione settimanali da inserire (rifarsi a "manage.py test" da shell per questi controlli).
- Una volta trovata una cella (o coppia di celle) adatta, le tabelle *t* e *o* vengono aggiornate, così come i contatori giorni_usati e ore_rimaste. Inoltre viene aggiunta una coppia (o due) di coordinate alla lista coordinate[], che contiene il riferimento della cella all'interno di entrambe le tabelle.
- L'algoritmo termina senza mai fallire, in collaborazione con il metodo di validazione della classe corso, che impedisce che ci siano meno spazi disponibili in una facoltà che ore di lezione ancora da inserire. Nel peggiore dei casi (a cui giunge raramente), la funzione *inserisci* assegna le ultime ore rimaste ai primi spazi liberi che trova, abbandonando l'idea di un orario di lezioni "bello" per dare priorità a uno "completo".

La funzione *inserisci* ritorna quindi una tripla composta da entrambe le tabelle di orario e dalla lista di coordinate[] essenziali per assegnare le aule in un secondo momento.

Ecco, è giunto quel secondo momento. Viene innanzitutto valutato se il corso ha o meno ore di laboratorio, e in quel caso gli viene assegnato un laboratorio in cui tenere quelle lezioni, tramite la funzione *assegna_laboratorio(c,aule_usate)*. Lo stesso viene fatto al passo successivo con la funzione *assegna_aula(c,aule_usate),* che prevedibilmente assegna un aula al corso per le lezioni normali. Entrambe queste funzioni prendono in ingresso le coordinate *c* ritornate dall'algoritmo in precedenza e la lista aule_assegnate[] per garantire una miglior distribuzione degli spazi (ovvero non usare sempre e solo la stessa aula). Il funzionamento di ciascuna è analogo a *inserisci*, ma con molti meno controlli e condizioni. Semplicemente viene assegnata la prima aula in ordine crescente di capienza che non è già impegnata per altre lezioni, e così iterando in peggio per essere sicuri di garantire almeno un'aula/laboratorio a ciascun corso.

Non basta però assegnare le aule al corso, come ultimo passaggio vanno anche scritte nella relativa tabella orari_aule[]. Il pattern di generazione è lo stesso di sempre, partendo da possibili soluzioni apprezzabili e arrivando a meno apprezzabili, scorrendo però invece che una tabella, stavolta una lista di coordinate, che vengono raggruppate per giorno (ovvero la seconda coordinata) tramite una mappatura dictionary.

4.6 Considerazioni finali sull'algoritmo e sull'applicazione

L'algoritmo di generazione degli orari prende ispirazione dagli orari di lezione che troviamo in ogni facoltà universitaria. Costruendolo, ho pensato che avrei evitato le ore "limite" della giornata, ovvero la prima ora della mattina, l'ultima del pomeriggio e le ore dedicate alla pausa pranzo. Ho cercato inoltre che risultasse apprezzabile dal punto di vista puramente razionale, complicando un po' le condizioni di assegnazione per ottenere ore di lezione meno spezzettate. Mi sono comunque trovato davanti all'ostacolo "ripetitività dell'output", che non ho nemmeno provato a superare; con questo intendo che mediamente gli orari calcolati automaticamente risultano abbastanza ripetitivi nell'arco della settimana, proponendo ogni giorno lo stesso pacchetto di lezioni. Non è certo un gran problema, ma non sono riuscito a renderlo migliore di così, per via dell'eccessiva dose di condizioni che avrei dovuto controllare ogni volta che l'algoritmo avrebbe deciso "qui va bene, ma è ripetitivo, passiamo oltre".

Sono complessivamente soddisfatto del risultato ottenuto a livello algoritmico, un po' meno a livello di struttura dati e di realizzazione visiva (che comunque non era richiesta), ma per quello incolpo il mio approccio non-tdd che mi ha fatto perdere un mucchio di tempo in correzioni estemporanee.