The Reception
Progetto ISW

User Stories e Diagrammi di sequenza

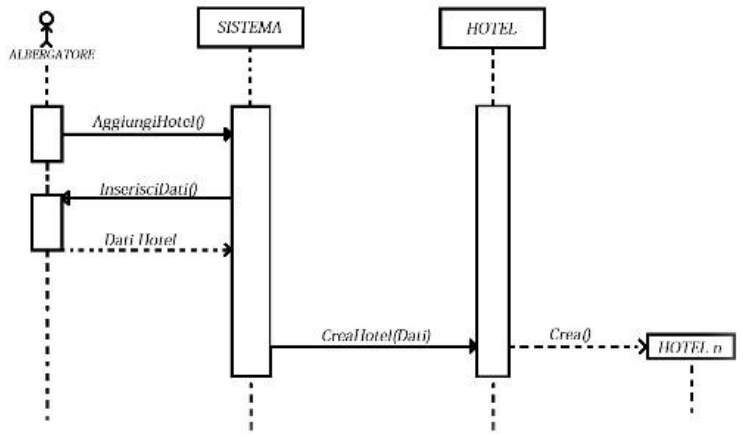
User Story : Aggiunta Hotel

1) Titolo: Aggiunta hotel
2) Priorità: 1
3) Test di Accettazione: TestAggiuntaHotel
4) Punti Storia: 2
5) Descrizione: In qualità di albergatore voglio registrarmi al sistema, accedere e aggiungere un Hotel alla mia lista hotel in modo che clienti abbiano più ampia scelta sulle camere.

Requisiti

1. L'albergatore deve potersi registrare al sistema.
2. L'albergatore deve poter aggiungere un Hotel alla sua lista Hotel inserendo i dati di quest'ultimo (nome,descrizione,città e indirizzo).

Diagramma di sequenza



User Story : Lista Hotel

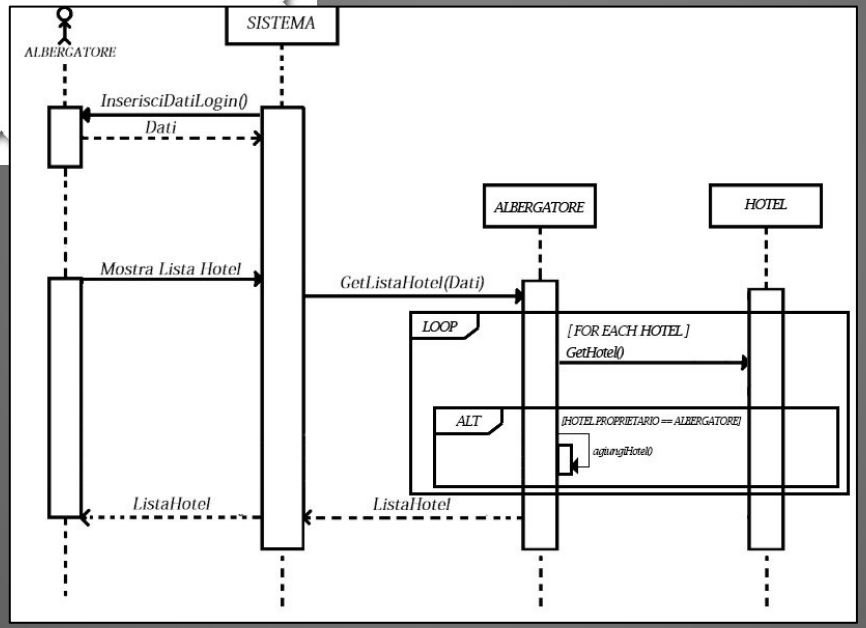
1) Titolo: Lista Hotel
2) Priorità: 2
3) Test di Accettazione: TestListaHotelPosseduti
4) Punti Storia: 2
5) Descrizione: In qualità di Albergatore, voglio potermi loggare e controllare la lista degli Hotel che possiedo, in modo da vedere se tutti i miei Hotel sono presenti

Prerequisiti

L'utente deve aver effettuato l'accesso al sistema

Requisiti

L'albergatore deve poter visualizzare tutti i suoi Hotel.



User story : Aggiungi Camera

1) Titolo: Aggiunta camera
2) Priorità: 2
3) Test di accettazione: TestAggiungiCamera
4) Punti storia: 3
5) Descrizione: In qualità di albergatore voglio accedere al sistema e aggiungere delle camere ai miei hotel in modo che i clienti le possano prenotare.

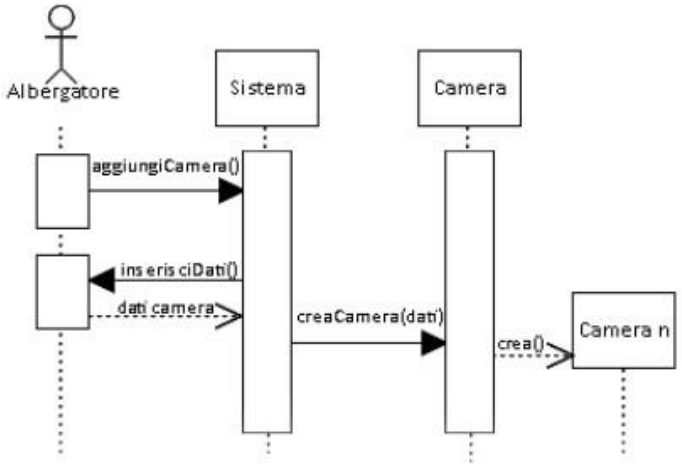
Prerequisiti

L'utente deve essere registrato al sistema

Requisiti

1. L'utente deve poter accedere al sistema inserendo i propri dati (email e password)
2. L'utente deve poter aggiungere una camera ad uno degli hotel posseduti e registrati nel sistema.

Diagramma di sequenza



User story : Elenco Camere

1) Titolo: Elenco camere
2) Priorità: 3
3) Test di accettazione: TestElencoCamere
4) Punti storia: 2
5) Descrizione: In qualità di albergatore voglio accedere al sistema in modo da poter visualizzare l'elenco delle camere del mio hotel.

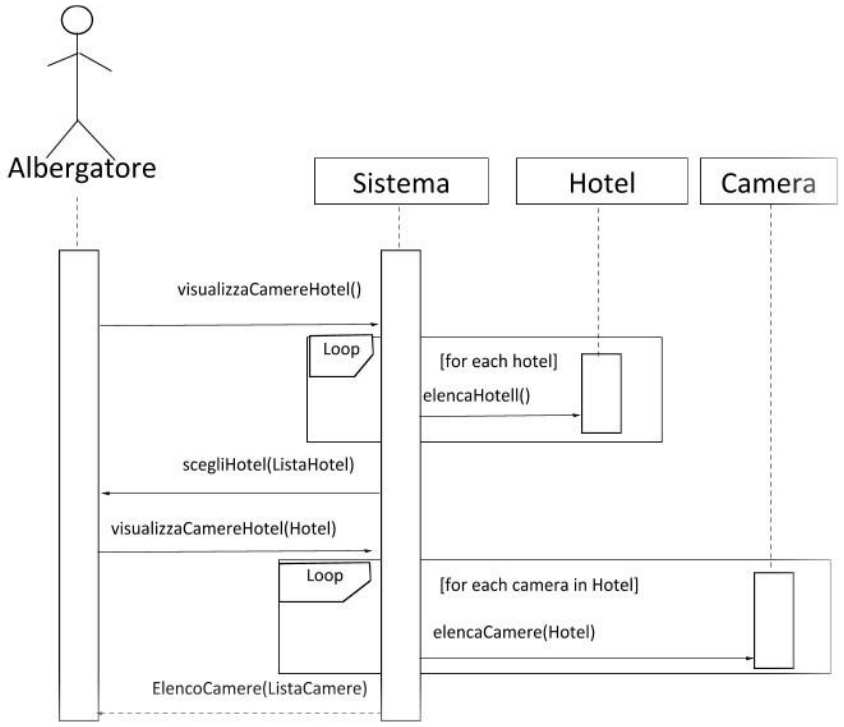
Prerequisiti

L'utente ha avuto accesso al sistema inserendo i propri dati (email e password)

Requisiti

L'utente deve poter visualizzare l'elenco delle camere degli hotel posseduti e registrati nel sistema.

Diagramma di sequenza



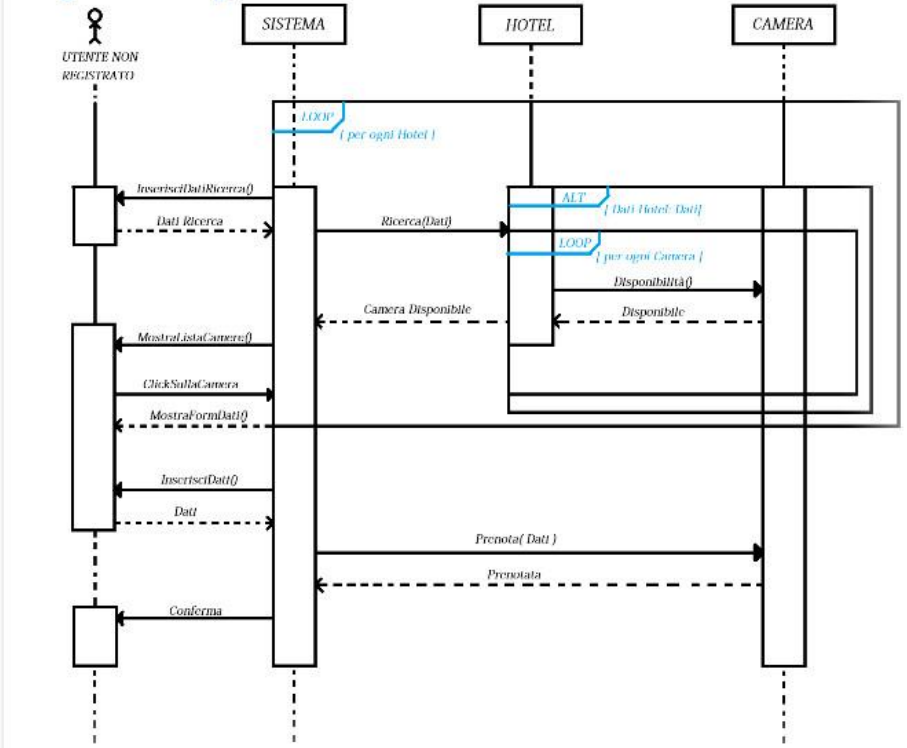
User Story : Prenotazione Camera

1) Titolo: Prenotazione camera
2) Priorità: 3
3) Test di Accettazione: TestPrenotazioneCamera
4) Punti Storia: 3
5) Descrizione: In qualità di utente non registrato voglio poter cercare una camera disponibile in una città in un dato periodo (dal, al) e con un preciso numero di posti letto in modo da poterla prenotare.

Requisiti

1. L'utente non registrato deve poter ricercare una camera disponibile inserendo i dati di ricerca (città, n° di posti letto, data checkin, data checkout).
2. L'utente non registrato deve poter prenotare una camera, se è disponibile, inserendo i suoi dati (email).

Diagramma di sequenza



User Story : Lista Prenotazioni

1) Titolo: Lista Prenotazioni
2) Priorità: 4
3) Test di Accettazione: TestListaPrenotazioni
4) Punti Storia: 2
5) Descrizione: In qualità di Albergatore, voglio potermi loggare e controllare la lista delle prenotazioni in modo da sapere da chi, da quando e fino a quando le camere dei miei Hotel sono occupate

Prerequisiti

L'utente deve essere registrato al sistema

Requisiti

1. L'albergatore deve poter eseguire l'accesso al sistema.
2. L'albergatore deve poter visualizzare l'elenco delle prenotazioni effettuate sulle camere dei suoi Hotel

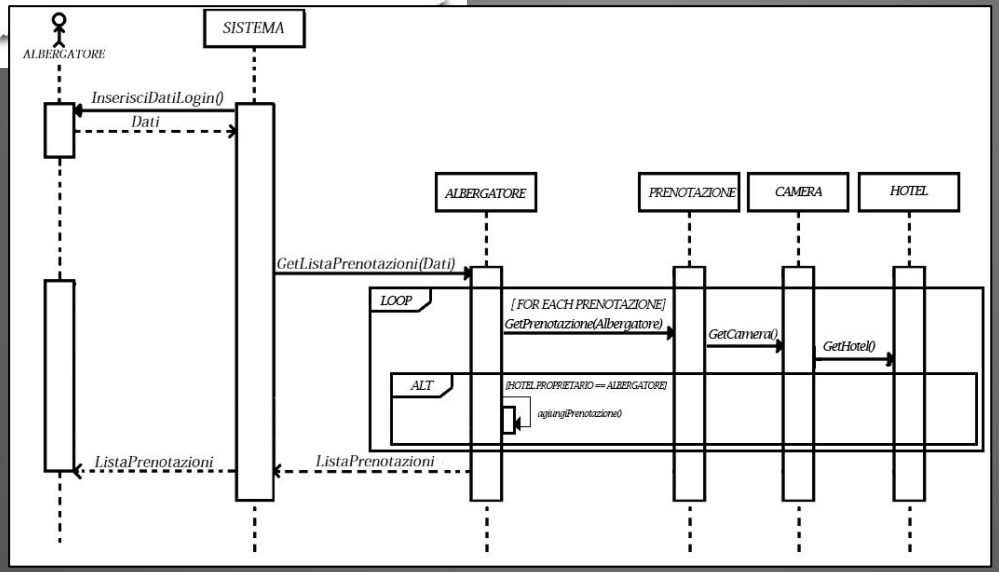
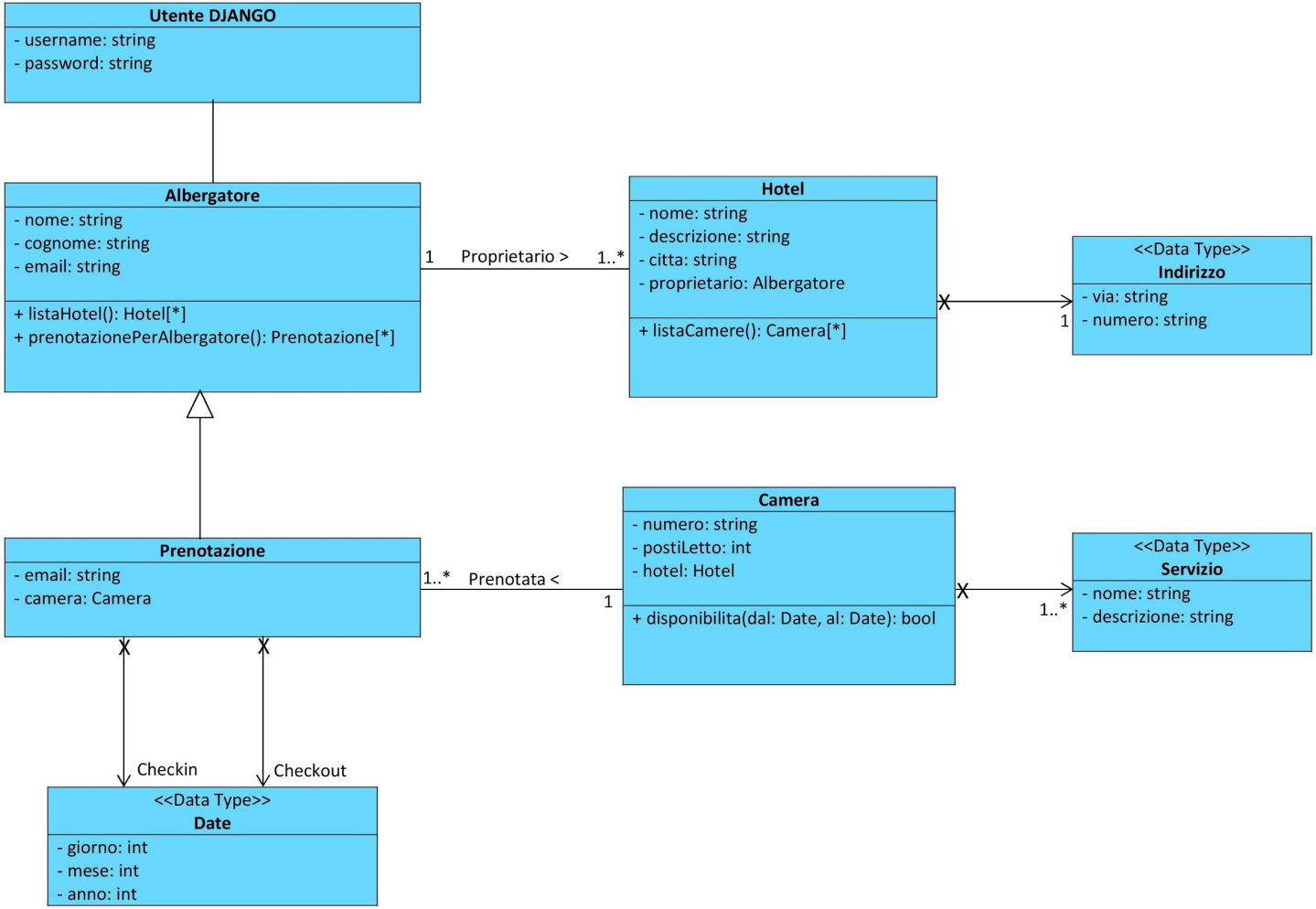
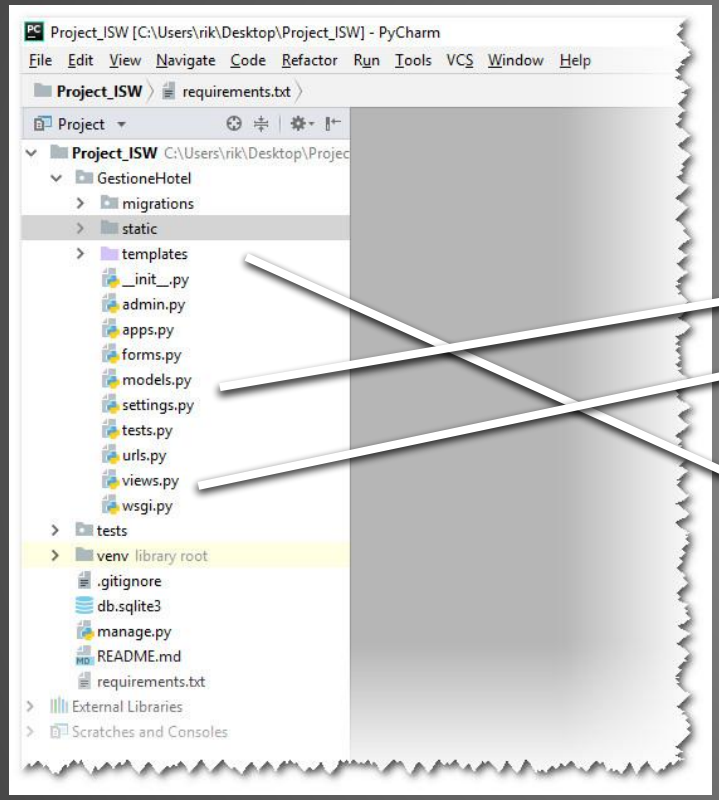


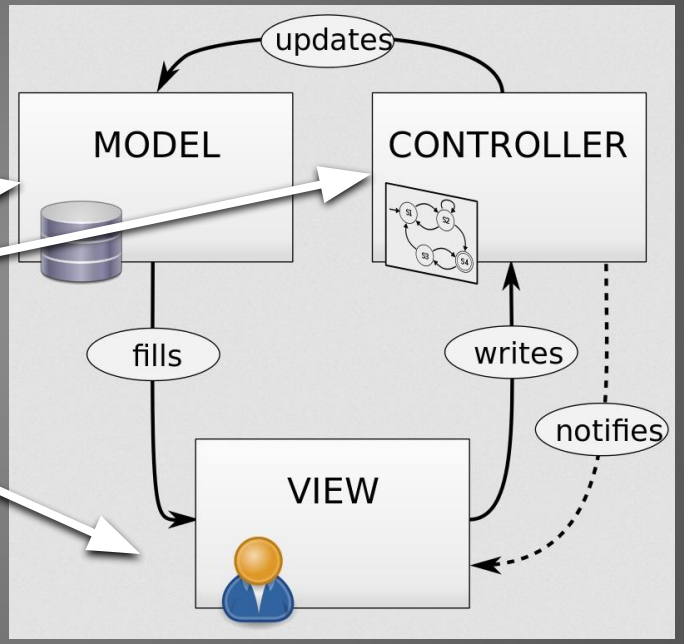
Diagramma delle Classi


Utente DJANGO
- username: string
- password: string
Albergatore
- nome: string
- cognome: string
- email: string
+ listaHotel(): Hotel[*]
+ prenotazionePerAlbergatore(): Prenotazione[*]
1
Proprietario >
1..*
Hotel
- nome: string
- descrizione: string
- citta: string
- proprietario: Albergatore
+ listaCamere(): Camera[*]
1
<<Data Type>>
Indirizzo
- via: string
- numero: string
Prenotazione
- email: string
- camera: Camera
1..*
Prenotata <
1
Camera
- numero: string
- postiLetto: int
- hotel: Hotel
+ disponibilita(dal: Date, al: Date): bool
1..*
<<Data Type>>
Servizio
- nome: string
- descrizione: string
Checkin
Checkout
<<Data Type>>
Date
- giorno: int
- mese: int
- anno: int

Il Framework Django

Pattern Architetturale MVC - Model View Controller





Gli Stili di scrittura e di commento

```python
#Classe Hotel
class Hotel(models.Model):
    #Nome dell'hotel
    nome = models.CharField(max_length=50,default="")
    #Descrizione dell'hotel
    descrizione = models.TextField(default="")
    #Citta' in cui si trova l'hotel
    citta = models.CharField(max_length=50,default="")
    #Indirizzo in cui si trova l'hotel
    indirizzo = models.ForeignKey(Indirizzo)
    #Albergatore che possiede l'hotel
    proprietario = models.ForeignKey(Albergatore)
    #Lista delle camere presenti nell'hotel
    def listaCamere(self):
        return Camera.objects.filter(hotel=self)
    #Override stampa oggetto Hotel
    def __unicode__(self):
        #Nome hotel, citta' hotel
        return "%s, %s" % (self.nome, self.citta)
```

Commenti

Nomi di classi: iniziale maiuscola

Nomi variabili e metodi: camel-case

Le Classi

Albergatore

```python
class Albergatore(AbstractUser): #Estende user django
    email = models.EmailField(_('email address'), unique=True)
    nome = models.CharField(max_length=50, default="")
    cognome = models.CharField(max_length=50, default="")
    #Lista degli Hotel di un Albergatore
    def listaHotel(self):
        listaHotel = []
        for hotel in Hotel.objects.filter(proprietario=self.id):
            listaHotel.append(hotel)
        return listaHotel
    #Lista delle prenotazioni effettuate sulle camere di un Hotel di uin albergatore
    def prenotazioniPerAlbergatore(self):
        listaPrenotazioni=[]
        for prenotazione in Prenotazione.objects.all():
            if prenotazione.camera.hotel.proprietario == self:
                listaPrenotazioni.append(prenotazione)
        return listaPrenotazioni

    def __unicode__(self):
        return "%s" % (self.email)

@receiver(pre_save, sender=Albergatore)
def aggiornamentoCampoEmail(instance, **kwargs):
    if instance.email == "":
        instance.email = instance.username
```

Estendendo lo user di Django abbiamo potuto utilizzare le funzioni signup, signin, e signout di Django

Attributi:
- email
- nome
- cognome

Indirizzo

```
37
38  class Indirizzo(models.Model):
39      via = models.CharField(max_length=50,default="")
40      numero = models.CharField(max_length=5,default="")
41
42      def __unicode__(self):
43          return "%s, %s" % (self.via, self.numero)
44
```

Implementazione del Data-Type

Attributi:
- via
- numero

Hotel

```
57  #Classe Hotel
58  class Hotel(models.Model):
59      #Nome dell'hotel
60      nome = models.CharField(max_length=50,default="")
61      #Descrizione dell'hotel
62      descrizione = models.TextField(default="")
63      #Citta' in cui si trova l'hotel
64      citta = models.CharField(max_length=50,default="")
65      #Indirizzo in cui si trova l'hotel
66      indirizzo = models.ForeignKey(Indirizzo)
67      #Albergatore che possiede l'hotel
68      proprietario = models.ForeignKey(Albergatore)
69      #Lista delle camere presenti nell'hotel
70      def listaCamere(self):
71          return Camera.objects.filter(hotel=self)
72      #Override stampa oggetto Hotel
73      def __unicode__(self):
74          #Nome hotel, citta' hotel
75          return "%s, %s" % (self.nome, self.citta)
76
```

Attributi:

- nome
- cognome
- descrizione
- indirizzo
- proprietario

Servizio

```
48  class Servizio(models.Model):
49      nome = models.CharField(max_length=15,default="")
50      descrizioneServizio = models.TextField(default="")
51
52      def __unicode__(self):
53          return "%s: %s" % (self.nome, self.descrizioneServizio)
54
```

Attributi:
- nome
- descrizione

Camera

```python
class Camera(models.Model):
    numero = models.CharField(max_length=15,default="")
    postiLetto = models.IntegerField(default=1)
    servizi = models.ManyToManyField(Servizio, through="ServiziDisponibili",through_fields=("camera","servizio"))
    hotel = models.ForeignKey(Hotel)

    def listaServizi(self):
        serviziPerCamera = []
        for servizioDisponibile in ServiziDisponibili.objects.filter(camera=self.id):
            serviziPerCamera.append(Servizio.objects.filter(id=servizioDisponibile.servizio.id))
        return serviziPerCamera

    def disponibilitaCamera(self, da, a):
        listaPrenotazioniPerCamera = Prenotazione.objects.filter(camera=self)
        for prenotazione in listaPrenotazioniPerCamera:
            if not(da.year > prenotazione.checkout.year and a.year > Prenotazione(camera=self).checkout.year and d
                return False
        return True

    def __unicode__(self):
        return "%s, %s" % (self.numero, self.hotel)
```

L'attributo 'servizio' è un ManyToManyField

Attributi:
- numero
- postiLetto
- servizi
- hotel

ServiziDisponibili

```
108 class ServiziDisponibili(models.Model):
109     camera=models.ForeignKey(Camera)
110     servizio=models.ForeignKey(Servizio)
111
112     def __unicode__(self):
113         return "%s: %s" % (self.camera, self.servizio)
114
```

E' la classe che implementa l'associazione (molti-a-molti) tra Camera e Servizio

Attributi:
- camera
- servizio

Prenotazione

```
121 class Prenotazione(models.Model):
122     camera=models.ForeignKey(Camera)
123     utente=models.CharField(max_length=50,default="")
124     checkin=models.DateField(default=django.utils.timezone.now)
125     checkout=models.DateField(default=django.utils.timezone.now)
126     def __unicode__(self):
127         return "%s: %s, %s - %s" % (self.camera, self.utente, self.checkin, self.checkout)
```

Attributi:

- camera
- utente
- checkin
- checkout

I Metodi

signup

```python
def signup(request):
    # Se risulta un utente gia' loggato nella sessione
    if request.user.is_authenticated and not request.user.is_anonymous and not request.user.is_superuser:
        id = Albergatore.objects.all().get(email=request.user).id
        return redirect("/Home/" + str(id))  # L'utente loggato che cerca di registrarsi viene rimandato alla sua home
    if request.method == 'POST':
        form = SignUpForm(request.POST)
        if form.is_valid():
            form.save()
            raw_password = form.cleaned_data.get('password1')
            username = form.cleaned_data.get('username')
            user = authenticate(username=username, password=raw_password)
            if user != None:
                login(request, user)
                if request.user.is_authenticated:
                    # Recupero l'id
                    id = request.user.id
                    # Redidrect alla sua Home
                    return redirect("/Home/" + str(id))
                else:
                    #Redirect a Main
                    return redirect("/")
    else:
        form = SignUpForm()

    return render(request, 'signup.html', {'form': form})
```

Funzioni svolte:
se l'utente è già registrato, si esegue un redirect alla sua home;
altrimenti lo si registra e si esegue un signin (login) automatico e infine un redirect alla sua home

signin

```python
#Login
def signin(request):
    #Se risulta un utente gia' loggato nella sessione
    if request.user.is_authenticated and not request.user.is_superuser and not request.user.is_anonymous:
        #Inizializzazione variabile id
        id = Albergatore.objects.all().get(email=request.user).id
        return redirect("/Home/" + str(id)) #L'utente loggato che cerca di accedere viene rimandato alla sua home
    error = False #Inizializzazione variabile per i messaggi di errore
    if request.method == "POST":  # Se il metodo della richiesta è POST
        loginForm = LoginForm(request.POST) #Si recuperano le informazioni del login form
        #Se il form è valido (le informazioni inserite nel form)
        if loginForm.is_valid():
            #Si recupera il testo dal campo email
            email = loginForm.cleaned_data['email']
            #Si recupera il testo dal campo password
            password = loginForm.cleaned_data['password']
            # Autenticazione: chi è l'utente
            user = authenticate(username=email, password=password)
            #Se l'utente esiste
            if user != None:
                id = Albergatore.objects.all().get(email=user).id
                #Login utente
                login(request, user)
                #L'utente viene mandato nella sua home
                return redirect("/Home/" + str(id))
            else:
                #Errore: l'utente non esiste
                error = True
        #Errore: form non valido
        else:
            error = True
    #Nuovo form vuoto nel caso in cui non sia stata presentata una richiesta POST, es: primo caricamento della pagina senza inserimento dati
    loginForm = LoginForm()
    #Restituzione template e campi dinamici
    return render(request, "Login.html", {
        "form": loginForm,
        "error": error
    })
```

Funzioni svolte:
se l'utente è già registrato (ma non loggato e non si tratta nè di superuser, nè di anonymous) si controllano le credenziali e si esegue un redirect alla sua home

signout

```
42  #Logout
43  @login_required(login_url='/Login') #Per effettuare il logout è richiesto il login
44  def signout(request): #View logout
45      #Distrugge la sessione
46      logout(request)
47      #Si ritorna al main, la pagina del sito che vedono gli utenti non registrati
48      return redirect("/")
49
50
```

Funzioni svolte:
se l'utente risulta realmente loggato si esegue un logout, altrimenti si esegue un redirect alla pagina principale del sito: Main

listaHotel

```python
@login_required(login_url='/Login')
def listaHotel(request, albergatoreID):
    nessunHotel = False
    try:
        albergatore = Albergatore.objects.all().get(id=albergatoreID) #Si recupera l'albergatore da albergatore id
        # Tentativo di accesso ad una pagina di un altro albergatore tramite albergatore id
        if not (str(request.user).__eq__(str(albergatore.email))): #Se l'albergatore a cui corrisponde
                                                                   # albergatore id è diverso dall'utente loggato
            id = Albergatore.objects.all().get(email=request.user).id
            return redirect("/AggiungiHotel/" + str(id)) #L'utente viene rimandato alla sua pagina aggiungi hotel
    except Albergatore.DoesNotExist:
        albergatore = None
    if albergatore != None:
        if request.method == "POST":
            aggiungiHotelForm = AggiungiHotelForm(request.POST)

            if aggiungiHotelForm.is_valid():
                nuovoIndirizzo = Indirizzo(via=aggiungiHotelForm.cleaned_data['via'],
                                           numero=aggiungiHotelForm.cleaned_data['numeroCivico'])
                nuovoIndirizzo.save()

                nuovoHotel = Hotel(nome=aggiungiHotelForm.cleaned_data['nome'],
                                   descrizione=aggiungiHotelForm.cleaned_data['descrizione'],
                                   citta=aggiungiHotelForm.cleaned_data['citta'],
                                   indirizzo=nuovoIndirizzo,
                                   proprietario=albergatore)
                nuovoHotel.save()

        aggiungiHotelForm= AggiungiHotelForm()
        listaHotel= []
        for hotel in Hotel.objects.all():
            if hotel.proprietario.__eq__(albergatore):
                listaHotel.append(hotel)
                listaHotel.append(len(hotel.listaCamere()))
        if len(listaHotel) == 0:
            nessunHotel = True
    else:
        return HttpResponseNotFound() #Albergatore non esistente, risposta 404
    return render(request,
```

Funzioni svolte:
se l'utente risulta loggato, mostra la lista dei suoi hotel e consente l'inserimento di un nuovo hotel

dettagliHotel

```python
#Hotel
@login_required(login_url='/Login') #Questa pagina è visibile solo da un albergatore, utente registrato
#Dettagli hotel e lista camere, viene passato con POST l'id dell'hotel interessato
def dettaglioHotel(request, hotelID):
    #Inizializzazione variabile per indicare l'assenza di camere nell'hotel (nessuna camera ancora aggiunta)
    nessunCamera = False
    #Recupera l'hotel in base all'ID
    try:
        hotel = Hotel.objects.get(id=hotelID)
    except Hotel.DoesNotExist:
        hotel = None
    #Date tutte le camere registrare nel db, prende quelle presenti in hotel
    if hotel != None: #Se l'hotel esiste
        email = Hotel.objects.all().get(id=hotelID).proprietario.email
        # Tentativo di accesso ad una pagina di un altro albergatore tramite hotel id diverso
        # (da uno di quelli possibili, quindi posseduti) in barra indirizzi
        if not (str(request.user).__eq__(str(email))):
            id = Albergatore.objects.all().get(email=request.user).id
            return redirect("/Home/" + str(id)) #Utente rimandato alla sua home
            #return HttpResponseForbidden() # Variante con codice 403
        # Collegato al form aggiungi camera
        if request.method == "POST":  #In caso di richiesta POST
            aggiungiCameraForm = AggiungiCameraForm(request.POST)
            # Recupera il form e controlla che sia valido, se si crea una camera
            if aggiungiCameraForm.is_valid():
                listaServizi = [] #Vengono aggiunti i servizi in base a quelli disponibili
                if aggiungiCameraForm.cleaned_data['servizio1'] == True:
                    listaServizi.append(Servizio.objects.all().get(nome="TV"))
                if aggiungiCameraForm.cleaned_data['servizio2'] == True:
                    listaServizi.append(Servizio.objects.all().get(nome="AC"))
                if aggiungiCameraForm.cleaned_data['servizio3'] == True:
                    listaServizi.append(Servizio.objects.all().get(nome="FB"))
                #Servizi esistenti recuperati da checkbox
                nuovaCamera = Camera(id=(len(Camera.objects.all())+1),numero=aggiungiCameraForm.cleaned_data['numero'],
                                     postiLetto=aggiungiCameraForm.cleaned_data['postiLetto'],
                                     hotel=hotel)
                nuovaCamera.save()
                # Salvataggio camera nel DB
```

Funzioni svolte:
se l'utente risulta loggato, mostra i dettagli dell'hotel selezionato e consente di inserire una nuova camera con i relativi servizi

prenotazioniPerAlbergatore

```python
@login_required(login_url='/Login')
def prenotazionePerAlbergatore(request, albergatoreID):
    nessunaPrenotazione = False
    try:
        albergatore = Albergatore.objects.all().get(id=albergatoreID)  # Si recupera l'albergatore da albergatore id
        # Tentativo di accesso ad una pagina di un altro albergatore tramite albergatore id
        if not (str(request.user).__eq__(str(
                albergatore.email))):  # Se l'albergatore a cui corrisponde albergatore id è diverso dall'utente loggato
            id = Albergatore.objects.all().get(email=request.user).id
            return redirect("/Home/" + str(id)) #L'utente viene rimandato alla sua pagina prenotazioni
    except Albergatore.DoesNotExist:
        albergatore= None
    if albergatore != None:
        listaPrenotazioni = []
        for prenotazione in Prenotazione.objects.all():
            if prenotazione.camera.hotel.proprietario.__eq__(albergatore):
                listaPrenotazioni.append(prenotazione)
        if len(listaPrenotazioni) == 0:
            nessunaPrenotazione = True
    else:
        return HttpResponseNotFound() #Albergatore non trovato, errore 404
    return render(request,
                  "Home.html",{
            'albergatoreID': albergatoreID,
                  'listaPrenotazioni':listaPrenotazioni,
                  'vuoto': nessunaPrenotazione
                  })
```

Funzioni svolte:
se l'utente risulta loggato, mostra l'elenco delle prenotazioni e relative camere, hotel e periodo prenotato.

Main

```python
def Main(request):
    if request.user.is_authenticated and not request.user.is_superuser and not request.user.is_anonymous: #Se l'utente ha fatto login
        id = Albergatore.objects.all().get(email=request.user).id  # Si recupera l'id albergatore con l'email
        return redirect("/Home/"+str(id)) #Viene rimandato alla sua home (in teoria non sono previste prenotazioni per gli utenti loggati)
    if request.method == "POST":
        # Da Main se clic su cerca...
        formRicerca = FormRicerca(request.POST)
        #se il form è correttamente compilato...
        if formRicerca.is_valid():
            citta=formRicerca.cleaned_data['citta']
            dataArrivo=formRicerca.cleaned_data['dataArrivo']
            dataPartenza = formRicerca.cleaned_data['dataPartenza']
            posti = formRicerca.cleaned_data['posti']
            # Controllo che le date non siano precedenti alla data di oggi
            if dataArrivo < datetime.date.today() or dataArrivo < datetime.date.today():
                msg="Attenzione: le date di arrivo e di partenza non possono essere precedenti ad oggi"
                return render(request, "Main.html", {'form': formRicerca, 'msg': msg})

            # Controllo che la data di partenza sia successiva a quella di arrivo
            if dataArrivo >= dataPartenza:
                msg="Attenzione: la data di partenza deve essere successiva a quella di arrivo"
                return render(request, "Main.html", {'form': formRicerca, 'msg': msg})
            # recupero le camere libere
            camerelibere=(Camera.objects.exclude(id__in=Prenotazione.objects.filter(checkin__lte=dataArrivo,checkout__gt=dataArrivo\
                    )).exclude(id__in=Prenotazione.objects.filter(checkin__lt=dataPartenza,checkout__gte=dataPartenza\
                    )).exclude(id__in=Prenotazione.objects.filter(checkin__gte=dataArrivo, checkout__lte=dataPartenza\
                    ))).filter(postiLetto=posti).filter(hotel__in=Hotel.objects.filter(citta=citta))
            if len(camerelibere)==0:
                # se nessuna Camera è stata trovata...
                msg="Nessuna Camera Trovata"
                # riporta a Main.html con un messaggio per la visualizzazione di un messaggio
                return render(request, "Main.html", {'form': formRicerca,'msg':msg})
            else:
```

Funzioni svolte:
consente la ricerca delle camere nel periodo e nella città selezionata, mostrando l'elenco delle camere disponibili e relativi servizi.

Prenota

```python
def Prenota(request,nomeHotel,numeroCamera,postiLetto,idCamera,arrivo,partenza):
    if request.method == "POST":
        # Registare la prenotazione
        formPrenota = FormPrenota(request.POST)
        if formPrenota.is_valid():
            #recupero i dati della prenotazione
            nuovaPrenotazione=Prenotazione()
            utente = formPrenota.cleaned_data['prenotaUtente']
            arrivo = formPrenota.cleaned_data['prenotaCheckin']
            partenza = formPrenota.cleaned_data['prenotaCheckout']
            idCamera = formPrenota.cleaned_data['prenotaIdCamera']
            # Qui il slvataggio
            nuovaPrenotazione.utente=utente
            nuovaPrenotazione.checkin=datetime.datetime.strptime(arrivo, "%d-%m-%Y")
            nuovaPrenotazione.checkout=datetime.datetime.strptime(partenza, "%d-%m-%Y")
            nuovaPrenotazione.camera=Camera.objects.get(id=idCamera)
            nuovaPrenotazione.save()

            return render(request, "PrenotazioneEffettuata.html")
        else:
            return render(request, "Main.html")
    else:
        #imposto i valori di default nella form (sono campi hidden
        formPrenota = FormPrenota(initial={'prenotaCheckout': partenza,'prenotaCheckin': arrivo, 'prenotaIdCamera': idCamera})

        return render(request, "Prenota.html", {'form':formPrenota,'nomeHotel':nomeHotel,'numeroCamera':numeroCamera,
                                                'postiLetto':postiLetto,'idCamera':idCamera,'arrivo':arrivo,'partenza':partenza})
```

Funzioni svolte:
consente di prenotare la camera per il periodo selezionato, inserendo un indirizzo email valido

I Test di accettazione

Test prenota camera, US Prenotazione camera

- **Primo test: model**
  - *Inizializzazione oggetti tramite metodo **setUp***
    - albergatore
    - hotel
    - indirizzo
    - camera
    - servizio
    - prenotazione
  - *Verifica corretta creazione della prenotazione sul DB tramite metodo **testPrenotaCamera***
    - utilizzo del metodo **listaPrenotazioniPerAlbergatore**
      - 1 sola prenotazione per l'albergatore
    - confronto dell'uguaglianza dei parametri *camera* e *utente* della prenotazione e di quelli creati in setUp

Test prenota camera, US Prenotazione camera

- **Secondo test: view**
  - *Inizializzazione oggetti tramite metodo **setUp***
    - albergatore
    - hotel
    - indirizzo
    - camera
    - servizio
  - *Metodo **testAccettazionePrenotazione:***
    - richiesta POST con *città, posti letto, data checkin, data checkout* alla pagina di ricerca
      - si verifica che la risposta contenga i parametri della camera libera creata precedentemente
    - richiesta POST con *email* alla pagina di riepilogo prenotazione
      - si verifica che la risposta contenga la parola "Congratulazioni", conferma dell'avvenuta prenotazione
      - si verifica che esista 1 riga nella tabella Prenotazione

Test aggiungi camera, US Aggiungi camera

- **Primo test: model**
  - *Inizializzazione oggetti tramite metodo **setUp***
    - albergatore
    - hotel
    - indirizzo
    - camera
    - servizio
  - *Task 1: login*
    - corrispondenza dei dati albergatore tra oggetto creato e oggetto ottenuto tramite metodo **get** dal DB
  - *Task 2: inserimento camera*
    - diversi controlli che verificano il corretto inserimento della camera nell'hotel
      - lista delle camere ottenuta filtrando le camere dell'hotel
      - attributo *hotel* di camera

Test aggiungi camera, US Aggiungi camera

- **Secondo test: view**
  - *Task 1: login*
    - richiesta POST alla pagina login con i dati dell'albergatore creato
    - verifica del reindirizzamento alla pagina prenotazioni
  - *Task 2: visualizzazione lista hotel*
    - richiesta GET alla pagina della lista hotel
    - si controlla che il *nome* dell'hotel creato e il *numero* totale di camere ottenuto tramite **len(listaCamere)** siano presenti
  - *Task 3: inserimento camera*
    - richiesta POST alla pagina dell'hotel aggiunto con i dati della nuova camera
    - si controlla che la risposta contenga il *numero*, i *posti letto* e i *servizi* della nuova camera

Test aggiungi hotel, US Aggiungi hotel

- **Inizializzazione oggetti tramite metodo setUp**
  - *albergatore*
  - *hotel*
  - *indirizzo*
- **Task 1: registrazione**
  - *corrispondenza dei dati albergatore tra oggetto creato e oggetto ottenuto tramite metodo **get** dal DB*
- **Task 2: inserimento hotel**
  - *tramite **listaHotel** per l'albergatore creato si controlla che l'hotel sia stato aggiunto correttamente*

Test lista prenotazioni, US Lista Prenotazioni

- **Primo test: view**
  - *Inizializzazione oggetti tramite metodo **setUp***
    - albergatore
    - hotel
    - indirizzo
    - camera
    - servizio
    - prenotazione
  - *Task 1: login*
    - richiesta POST alla pagina login con i dati dell'albergatore creato
    - verifica del reindirizzamento alla pagina prenotazioni
  - *Task 2: visualizzazione lista prenotazioni*
    - richiesta GET alla pagina della lista prenotazioni
    - si controlla che il "*dati*" delle prenotazioni create siano presenti

Test lista hotel posseduti, US Lista Hotel

- **Primo test: view**
  - *Inizializzazione oggetti tramite metodo* ***setUp***
    - albergatore
    - hotel
    - indirizzo
  - *Task 1: login*
    - richiesta POST alla pagina login con i dati dell'albergatore creato
    - verifica del reindirizzamento alla pagina prenotazioni
  - *Task 2: visualizzazione lista hotel*
    - richiesta GET alla pagina della lista hotel
    - si controlla che il *nome* dell'hotel creato e il *numero* totale di camere ottenuto tramite **len(listaCamere)** siano presenti

Test elenco camere, US Elenco Camere

- **Primo test: view**
  - *Inizializzazione oggetti tramite metodo* ***setUp***
    - albergatore
    - hotel
    - indirizzo
    - camera
    - servizio
  - *Task 1: login*
    - richiesta POST alla pagina login con i dati dell'albergatore creato
    - verifica del reindirizzamento alla pagina prenotazioni
  - *Task 2: visualizzazione lista hotel*
    - richiesta GET alla pagina della lista hotel
    - si controlla che il *nome* dell'hotel creato e il *numero* totale di camere ottenuto tramite **len(listaCamere)** siano presenti

Test elenco camere, US Elenco Camere

- *Task 3: visualizzazione lista camere*
  - richiesta GET alla pagina dell'hotel
  - il *numero* totale di camere ottenuto tramite **len(listaCamere)**
  - si controlla il *numero(id)* delle camere e i relativi dati siano presenti
  - si controlla che il *nome* dei servizi creati e il *numero* totale di servizi ottenuto tramite **for()** siano presenti