Riccardo Massi, matricola 152341
UNIMORE, Università degli studi di Modena e Reggio Emilia

Web application per il negozio Dannarita Parrucchieri

IDEA DELLA WEB APPLICATION

L'idea alla base della web application è quella di digitalizzare e automatizzare la gestione dei servizi di un negozio di parrucchieri.

Attraverso la piattaforma, i clienti possono:

- Visualizzare la storia e i servizi offerti dal negozio
- Accedere a tutti i contatti, il luogo e gli orari del negozio
- Visualizzare e prenotare prodotti per la cura personale e dei capelli, acquistabili poi in negozio

Il sistema di backend è responsabile della gestione dei dati e della logica aziendale, mentre il frontend offre un'interfaccia utente moderna e reattiva per interagire con i servizi offerti.

VANTAGGI DI UTILIZZO DI DJANGO E DRF PER IL BACKEND E NEXT.JS PER IL FRONTEND

Django e Django Rest Framework

- **Rapidità di Sviluppo**: Django è noto per il suo rapido sviluppo grazie alla presenza di numerosi strumenti e librerie integrate che facilitano la creazione di applicazioni web.
- **Sicurezza**: Django include molte funzionalità di sicurezza per proteggere l'applicazione da attacchi comuni come SQL injection, cross-site scripting, e cross-site request forgery.
- **Scalabilità**: Grazie alla sua architettura modulare, Django permette di scalare facilmente l'applicazione man mano che il business cresce.
- **Gestione delle API**: Django REST Framework offre un set completo di funzionalità per creare API RESTful, inclusa la serializzazione dei dati, la gestione delle autenticazioni e delle autorizzazioni.

Next.js

- **Rendering Ibrido**: Next.js offre sia il rendering lato server (SSR) che la generazione di pagine statiche (SSG), migliorando le performance e l'ottimizzazione per i motori di ricerca (SEO).
- **Esperienza di Sviluppo**: Next.js fornisce un'ottima esperienza di sviluppo con funzionalità come il reloading automatico, la gestione integrata delle route e il supporto per CSS e Sass.
- **Performance**: Grazie al prefetching delle pagine e al supporto per la static site generation, Next.js permette di creare applicazioni web altamente performanti.

Utilizzo di JavaScript e Axios in Next.js

Per il frontend, è stato utilizzato Next.js con JavaScript per creare una Single Page Application (SPA) che offre un'esperienza utente fluida e reattiva. JavaScript è stato scelto per la sua flessibilità e la vasta comunità di sviluppatori, che fornisce un ampio supporto e numerose librerie utili.

Per le richieste API, è stata utilizzata la libreria Axios, che offre una sintassi semplice e pulita per effettuare chiamate HTTP. Axios è stato scelto per i seguenti motivi:

- **Facilità d'Uso**: La sintassi di Axios è intuitiva e permette di scrivere codice chiaro e conciso per le richieste HTTP.
- **Gestione delle Promesse**: Axios utilizza le Promises di JavaScript, che facilitano la gestione delle operazioni asincrone.
- **Intercettori di Richiesta e Risposta**: Axios permette di configurare intercettori per le richieste e le risposte, utili per aggiungere logica personalizzata come la gestione degli errori o l'inserimento di token di autenticazione.

```
import axios from 'axios';

// Effettua una richiesta GET a un'API
axios.get('http://api/data')
  .then(response => {
    // Gestisci i dati di risposta
    console.log(response.data);
  })
  .catch(error => {
    // Gestisci gli errori
    console.error('Si è verificato un errore:', error);
  });
```

SCELTE FATTE PER I MODELLI DJANGO

Gestione degli utenti

Nel progetto è stato creato un modello utente personalizzato per soddisfare meglio le esigenze specifiche dell'applicazione. Il modello `CustomUser` eredita da `AbstractBaseUser` e `PermissionsMixin` e include i seguenti campi:

- `user_id`: Identificatore unico per l'utente.
- `username`: Nome utente unico per l'autenticazione.
- `is_superuser`: Campo booleano per indicare se l'utente è un superuser.
- `is_staff`: Campo booleano per indicare se l'utente è un membro dello staff.

È stato anche implementato un manager personalizzato `CustomUserManager` per gestire la creazione degli utenti e dei superuser.

```
class CustomUserManager(BaseUserManager):
  def create_user(self, username, password=None):
    if not username:
      raise ValueError('The Username field must be set')
    if not password:
      raise ValueError('The Password field must be set')
    user = self.model(username=username)
    user.set_password(password)
    user.save()
    return user
```

Controllo del superuser

Per controllare se un utente è un superuser, è stato creato un permesso personalizzato `IsSuperUser`:

```
class IsSuperUser(BasePermission):
  def has_permission(self, request, view):
    return request.user.is_superuser
```

View per Controllare se l'Utente è Loggato

Per permettere al frontend di controllare se l'utente è loggato e decidere quali pagine mostrare in base a se l'utente è loggato o superuser, è stata implementata una view `CustomUserAPIView`:

```
class CustomUserAPIView(APIView):
    permission_classes = [permissions.IsAuthenticated]
    authentication_classes = [SessionAuthentication]

    def get(self, request):
        serializer = CustomUserSerializer(request.user)
        return Response({'user': serializer.data},
status=status.HTTP_200_OK)
```

Modelli dei Prodotti e delle Prenotazioni

Il modello `Product` rappresenta i prodotti disponibili per la vendita nel negozio:

```
class Product(models.Model):
  id = models.AutoField(primary_key=True)
  name = models.CharField(max_length=100)
  description = models.TextField()
  price = models.DecimalField(max_digits=10,
decimal_places=2)
  image = models.ImageField(upload_to='static/', null=True,
blank=True)

  def __str__(self):
    return self.name
```

Sono stati creati i modelli `Carrello` e `Prenotazione` per gestire il carrello degli acquisti e le prenotazioni dei prodotti:

```
class Carrello(models.Model):
  utente = models.OneToOneField('CustomUser',
on_delete=models.CASCADE)
  prodotti = models.ManyToManyField('Product',
through='CarrelloProdotto')

  def __str__(self):
    return f"Carrello di {self.utente.username}"

class CarrelloProdotto(models.Model):
  carrello = models.ForeignKey('Carrello',
on_delete=models.CASCADE)
```

```
  prodotto = models.ForeignKey('Product',
on_delete=models.CASCADE)
  quantita = models.PositiveIntegerField(default=1)

  class Meta:
    unique_together = ('carrello', 'prodotto')

class Prenotazione(models.Model):
  id_prenotazione = models.AutoField(primary_key=True)
  utente = models.ForeignKey(CustomUser,
on_delete=models.CASCADE)
  data_prenotazione =
models.DateTimeField(auto_now_add=True)
  prodotti = models.ManyToManyField(Product,
through='PrenotazioneProdotto')

  def __str__(self):
    return f"Prenotazione {self.id_prenotazione} di
{self.utente.username}"

class PrenotazioneProdotto(models.Model):
  prenotazione = models.ForeignKey(Prenotazione,
on_delete=models.CASCADE)
  prodotto = models.ForeignKey(Product,
on_delete=models.CASCADE)
  quantita = models.PositiveIntegerField(default=1)

  class Meta:
    unique_together = ('prenotazione', 'prodotto')
```

SERIALIZER

Vantaggi dei serializer

I serializer sono una componente fondamentale in Django REST Framework poiché permettono di trasformare i dati complessi come le query del database in formati JSON o XML che possono essere facilmente utilizzati nel frontend. I principali vantaggi dei serializer sono:

- **Facilità di Validazione**: I serializer includono meccanismi integrati per la validazione dei dati, garantendo che i dati inviati dal frontend siano corretti prima di essere salvati nel database.
- **Flessibilità**: I serializer possono essere personalizzati per gestire casi d'uso complessi, come la serializzazione di relazioni tra modelli.
- **Sicurezza**: I serializer permettono di controllare quali campi dei modelli vengono esposti al frontend, migliorando la sicurezza dell'applicazione.