

## RAPPORT DE PROJET

# Application Web “Blabler”

### Projet réalisé par

Ilyas BENHSSINE  
Imane ELAFADAL

### Projet encadré par

Pr. El MENDILI Fatna



## Remerciements

Tout d’abord, nous tenons à remercier tout particulièrement et à témoigner toute notre reconnaissance aux personnes suivantes, pour leur dévouement et leur soutien dans la concrétisation de ce projet :

**Mme. Fatna EL MENDILI**, formatrice, pour ses conseils éclairés, sa patience, sa disponibilité et pour la confiance qu’il nous a accordée dès l’ébauche du projet et tout au long de la formation.

**Mr. Younes EL BOUZEKRI**, responsable de la formation, pour ses conseils éclairés, sa patience, sa disponibilité et pour la confiance qu’il nous a accordée dès l’ébauche du projet et tout au long de la formation.

**L’Université Ibn Tofail**, et l’ensemble des enseignants pour leur coopération professionnelle tout au long de cette expérience et pour avoir partagé avec nous, une partie de leurs savoir-faire et de leurs expériences professionnelles.

## Introduction

Dans le cadre de notre fin de module en développement Python à ENSAK, il nous est proposé de réaliser un projet ce qui nous permettant de mettre en pratique nos connaissances et nos compétences professionnelles au travers d’un cahier des charges ayant pour finalité la conception et le développement d’une application web en utilisant le framework Django en accords avec nos intérêts professionnels.

# Description du Projet

## Contexte

Blabler est une plateforme de microblogage et de réseautage social. La plateforme permet aux utilisateurs de publier des contenus multimédias et autres sur un sujet abrégé. Les utilisateurs peuvent suivre les blogs d'autres utilisateurs.

## Technologie utilisée

### Python

Python est un langage de programmation interprété, de haut niveau et polyvalent. La philosophie de conception de Python met l'accent sur la lisibilité du code avec son utilisation notable d'une indentation significative.

### Django

Django est un framework Web Python de haut niveau qui encourage un développement rapide et une conception propre et pragmatique. Conçu par des développeurs expérimentés, il prend en charge une grande partie des tracas du développement Web, vous pouvez donc vous concentrer sur l'écriture de votre application sans avoir à réinventer la roue. C'est gratuit et open source.

### Less

Less (qui signifie Leaner Style Sheets) est une extension de langage rétrocompatible pour CSS. Ceci est la documentation officielle de Less, le langage et Less.js, l'outil JavaScript qui convertit vos styles Less en styles CSS.

## Structure des Models

### Account

Le modèle Account est le modèle principal qui identifie chaque utilisateur et qui contient toutes les informations principales sur l'utilisateur. Le modèle Account remplace le modèle USER qui est automatiquement créé par Django.

- First Name
  - Last Name
  - Email\*
  - Password\*
- Location
  - Sexe
  - Bio

### Profile

Le modèle Profile est le modèle qui contient toutes les informations secondaires sur chaque utilisateur, et qui étend le modèle Account avec une relation OneToOne, il s'auto détruit à la suppression de son modèle Account, et qui peut être étendu sans causer de problèmes.

- Account\*
- Photo de Profile

### Topic

Le modèle Topic est le modèle qui représente chaque sujet, il contient plusieurs posts où ils sont rangés selon leur propre sujet choisi par l'utilisateur.

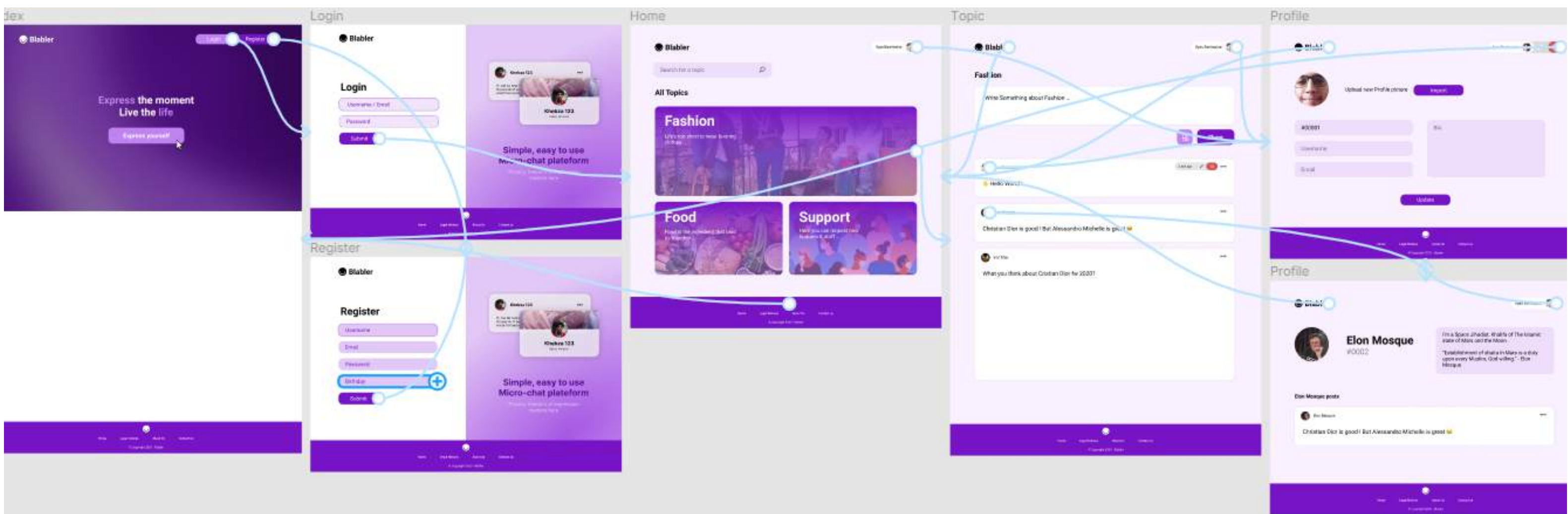
- Title
- Description
- Topic Pic

### Post

Le modèle Post est le modèle qui représente chaque Post qui appartient à un **Utilisateur** et un **Topic**. où il peut contenir un message et une photo.

- Topic\*
  - Author\*
  - Message
- Image
  - TimeStamp

## Structure des Pages





# Developement

## Qu'est-ce qu'un MVT

Le modèle-vue-modèle (MVT) est légèrement différent de MVC. En fait, la principale différence entre les deux modèles est que Django s'occupe lui-même de la partie Controller (code logiciel qui contrôle les interactions entre le modèle et la vue), nous laissant avec le modèle. Le modèle est un fichier HTML mélangé avec Django Template Language (DTL).

## Pourquoi utiliser Django

### Prise en charge du mappage objet-relationnel

Django fournit un pont entre le modèle de données et le moteur de base de données, et prend en charge un grand nombre de systèmes de base de données, y compris MySQL, Oracle, Postgres, etc. Django prend également en charge la base de données NoSQL via Django-nonrel fork. Pour l'instant, les seules bases de données NoSQL prises en charge sont MongoDB et Google App Engine.

### Prise en charge multilingue

Django prend en charge les sites Web multilingues via son système d'internationalisation intégré. Ainsi, vous pouvez développer votre site Web, qui prendrait en charge plusieurs langues.

### Support du framework

Django a un support intégré pour Ajax, RSS, Caching et divers autres frameworks.

### Interface graphique d'administration

Django fournit une belle interface utilisateur prête à l'emploi pour les activités administratives.

### Environnement de développement

Django est livré avec un serveur Web léger pour faciliter le développement et les tests d'applications de bout en bout.

## Code Source

### Remplacer un modèle utilisateur personnalisé

Pour bien profiter des options fournies par Django comme le système d'authentification pour éviter la réécriture du code, on a décidé de remplacer le modèle BaseUser par défaut avec notre modèle Account qui étend le modèle BaseUser et créer un modèle Manager "MyAccountManager" qui étend la classe BaseUserManager

```
class Account(AbstractBaseUser):
    Sexe_Choice = [
        ('Male', 'Male'),
        ('Female', 'Female'),
        ('Other', 'Other'),
        ('Not Specified', 'Not Specified'),
    ]

    id = models.AutoField(primary_key=True)
    email = models.EmailField(verbose_name='email',
    unique=True, null=False)
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    password = models.CharField(max_length=100)
    location = models.CharField(max_length=50, default='')
    sexe = models.CharField(max_length=50,
    choices=Sexe_Choice, default='Not Specified')
    bio = models.TextField(max_length=500, default='')
    date_joined = models.DateTimeField(verbose_name='date
    joined', auto_now_add=True)
    last_login = models.DateTimeField(verbose_name='last
    login', auto_now=True)
    is_admin = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)

    objects = MyAccountManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name', 'password']

    def __str__(self):
        return self.first_name + " " + self.last_name + " |
        "+self.email

    def has_perm(self, perm, obj=None):
        return self.is_admin

    def has_module_perms(self, app_label):
        return True
```

More Info on the Django Docs :  
<https://docs.djangoproject.com/en/3.1/topics/auth/customizing/#substituting-a-custom-user-model>

### Utilisation des Forms

Django fournit une gamme d'outils et de bibliothèques pour vous aider à créer des formulaires pour accepter les entrées des visiteurs du site, puis traiter et répondre à l'entrée. Voici un exemple d'utilisation des Forms.

```
def register(request):
    form = CreateAccountForm()
    if request.method == 'POST':
        form = CreateAccountForm(request.POST)
        if form.is_valid():
            form.save()
            user = Account.objects.get(email =
            form.cleaned_data.get('email'))
            profile = Profile(account = user)
            profile.save()
            first_name = form.cleaned_data.get('first_name')
            last_name = form.cleaned_data.get('last_name')
            messages.success(request, 'Account was created for
            '+ first_name + ' ' + last_name)
            return redirect('/login')

    context = {'form':form}
    return render(request, 'accounts/register.html', context)
```

```
[...]
<form action="/register" method="POST" class="v-form">
    {% csrf_token %}
    <h1>Register</h1>
    <br>
    {% for field in form %}
    {{ field }}
    {% endfor %}
    <div class="errorbox">
        {{ form.errors }}
    </div>

    <input type="submit" value="Submit">
</form>
[...]
```

More Info on the Django Docs :  
<https://docs.djangoproject.com/en/3.1/topics/forms/>

### Utilisation des Messages / Notifications Framework

Très souvent, dans les applications Web, vous devez afficher un message de notification unique (également appelé «message flash») à l'utilisateur après avoir traité un formulaire ou d'autres types d'entrée utilisateur.

Pour cela, Django fournit une prise en charge complète de la messagerie basée sur les cookies et les sessions, pour les utilisateurs anonymes et authentifiés. La structure des messages vous permet de stocker temporairement des messages dans une demande et de les récupérer pour les afficher dans une demande ultérieure (généralement la suivante). Chaque message est étiqueté avec un niveau spécifique qui détermine sa priorité (par exemple, info, warning ou error, success).

```
if int(request.POST.get('author', 'null')) ==
int(request.user.id):
    form.topic = Topic.objects.get(id =
    request.POST.get('topic', '-1'))
    if form.is_valid():
        form.save()
        messages.success(request, 'Post posted successfully')
    else:
        messages.error(request, 'The Form is not valid')
        context = {'form':form}
        return render(request, 'accounts/errors.html',
        context)
    else:
        messages.error(request, 'Go & Play with your friends
        little kid')
```

More Info on the Django Docs :  
<https://docs.djangoproject.com/en/3.1/ref/contrib/messages/>

### Utilisation de l'URL dispatcher

Un schéma d'URL propre et élégant est un détail important dans une application Web de haute qualité. Django vous permet de concevoir des URL comme vous le souhaitez, sans aucune limitation du framework.

Voir Cool URIs don't change, par le créateur du World Wide Web Tim Berners-Lee, pour d'excellents arguments sur les raisons pour lesquelles les URL doivent être propres et utilisables.

```
urlpatterns = [
    ## ---- BASE
    path('', views.index, name='index'),
    path('login', views.loginauth, name='login'),
    path('register', views.register, name='register'),
    path('home', views.home, name='home'),
    path('logout', views.disconnect, name='logout'),
    ## ---- PROFILE
    path('profile/edit', views.profileEdit,
    name='profile_edit'),
    path('profile/editprofilepic', views.profilepicEdit,
    name='profilepic_edit'),
    path('profile/<str:pkid>', views.profileShow,
    name='profile_show'),
    ## ---- TOPIC
    path('topic/<str:pkid>', views.topicShow,
    name='topic_show'),
    ## ---- POST
    path('newpost', views.newPost, name='newpost'),
    path('editpost', views.editPost, name='editpost'),
    path('deletePost/<str:pkid>', views.deletePost,
    name='deletepost'),
]
```

More Info on the Django Docs :  
<https://docs.djangoproject.com/en/3.1/topics/http/urls/>

### Utilisation des Templates

Django définit une API standard pour le chargement et le rendu des modèles quel que soit le backend. Le chargement consiste à trouver le modèle pour un identifiant donné et à le prétraiter, généralement en le compilant en une représentation en mémoire. Le rendu signifie interpoler le modèle avec des données de contexte et renvoyer la chaîne résultante.

```
{% load static %}

<DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    [...]
    <title>Blablier {% block title %}{% endblock %}</title>
</head>
<body>
    {% block top-content %}
    {% endblock %}
    <div class="wrapper">
        {% block content %}
        {% endblock %}
    </div>
    {% block bottom-content %}
    {% endblock %}
</body>
</html>
```

More Info on the Django Docs :  
<https://docs.djangoproject.com/en/3.1/topics/templates/>

```
class MyAccountManager(BaseUserManager):
    def create_user(self, email, first_name, last_name,
    password):
        if not email:
            raise ValueError("Email Address is Required")
        if not first_name:
            raise ValueError("First name is Required")
        if not last_name:
            raise ValueError("Last name is Required")
        if not password:
            raise ValueError("Password is Required")
        user = self.model(
            email = self.normalize_email(email),
            first_name = first_name,
            last_name = last_name,
        )

        user.set_password(password)
        user.save(using = self._db)
        return user

    def create_superuser(self, email, first_name, last_name,
    password):
        user = self.create_user(
            email=self.normalize_email(email),
            first_name=first_name,
            last_name=last_name,
            password=password
        )
        user.is_admin = True
        user.is_staff = True
        user.is_superuser = True
        user.save(using = self._db)
        return user
```

```
class CreateAccountForm(UserCreationForm):
    password1 =
    forms.CharField(widget=forms.PasswordInput(attrs=
    {'placeholder': 'Password'}))
    password2 =
    forms.CharField(widget=forms.PasswordInput(attrs=
    {'placeholder': 'Confirm Password'}))
    class Meta:
        model = Account
        fields = ['first_name', 'last_name', 'email',
        'password1', 'password2']

    widgets = {
        'first_name' : forms.TextInput(attrs=
        {'placeholder': 'First Name'}),
        'last_name': forms.TextInput(attrs={'placeholder':
        'Last Name'}),
        'email': forms.EmailInput(attrs={'placeholder':
        'Email'}),
    }
```

```
<div class="notiflist">
    {% for message in messages %}
    <div class="notif" id="nt{{ forloop.counter }}">
    <p>{{message}}</p>
    <p class="bx bx-x" onclick="closeNotification('#nt{{
    forloop.counter }}"')"></p>
    </div>
    {% endfor %}
</div>
```

```
@login_required(login_url="/login")
def deletePost(request, pkid):
    try:
        post = Post.objects.get(id=pkid)
        if Post.objects.filter(id=pkid).exists():
            if post.author.id == request.user.id:
                post.delete()
                messages.success(request, 'Post deleted
                successfully')
            else:
                messages.error(request, "You can't delete this
                Post ")
        except:
            messages.error(request, "This Post does not exist")
        return redirect('home')
```



## Wireframes, Filaires

Un filaire de site Web, également connu sous le nom de schéma de page ou de plan d'écran, est un guide visuel qui représente le cadre squelettique d'un site Web. Les wireframes sont créés dans le but d'organiser des éléments pour atteindre au mieux un objectif particulier.

Nous avons décidé de rester sur une structure filaire de base, facile à utiliser et à comprendre pour créer un Look & Feel digne d'un travail professionnel



## Composants

Chaque composant a été conçu et codé pour résoudre un problème d'interface utilisateur spécifique, tel que la présentation d'une liste d'options, l'activation de la soumission d'un formulaire, la fourniture de commentaires à l'utilisateur, etc. Tous les composants de Blabler ont été conçus pour fonctionner harmonieusement ensemble, en tant que parties d'un plus grand ensemble.

### Forms

Les formulaires sont utilisés pour soumettre des données afin d'être aussi concis que possible lors de la conception.

#1

First Name

Last Name

Location

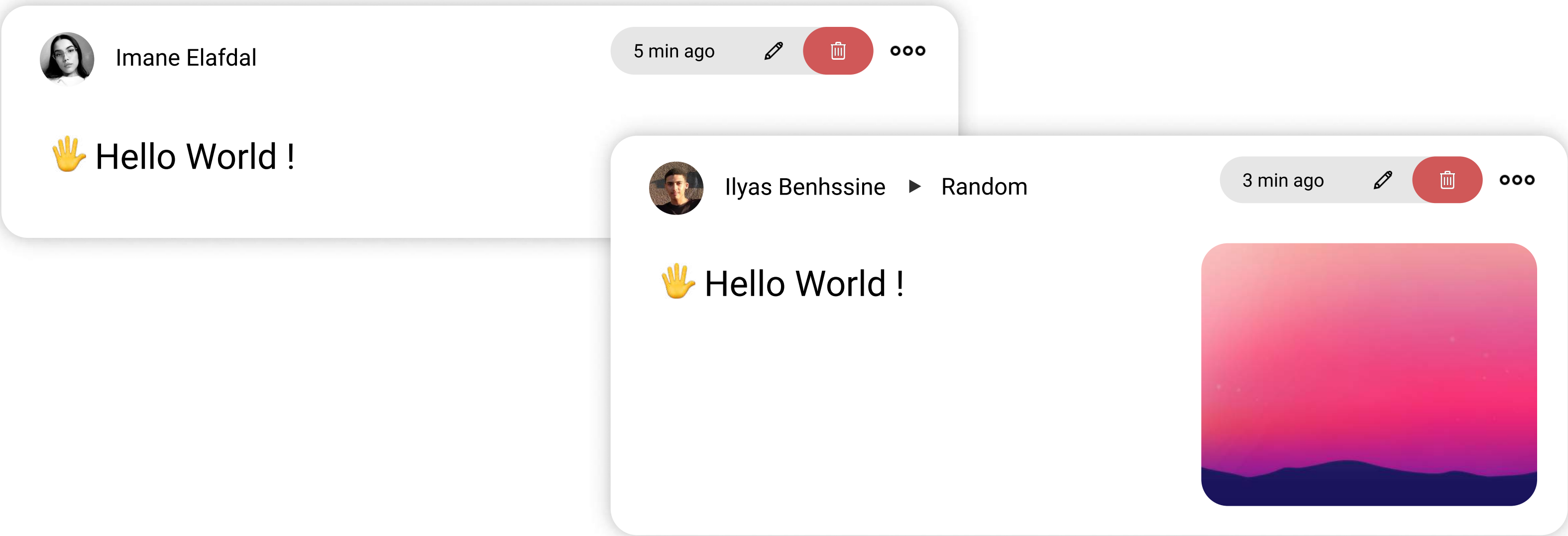
Male

Bio

Submit

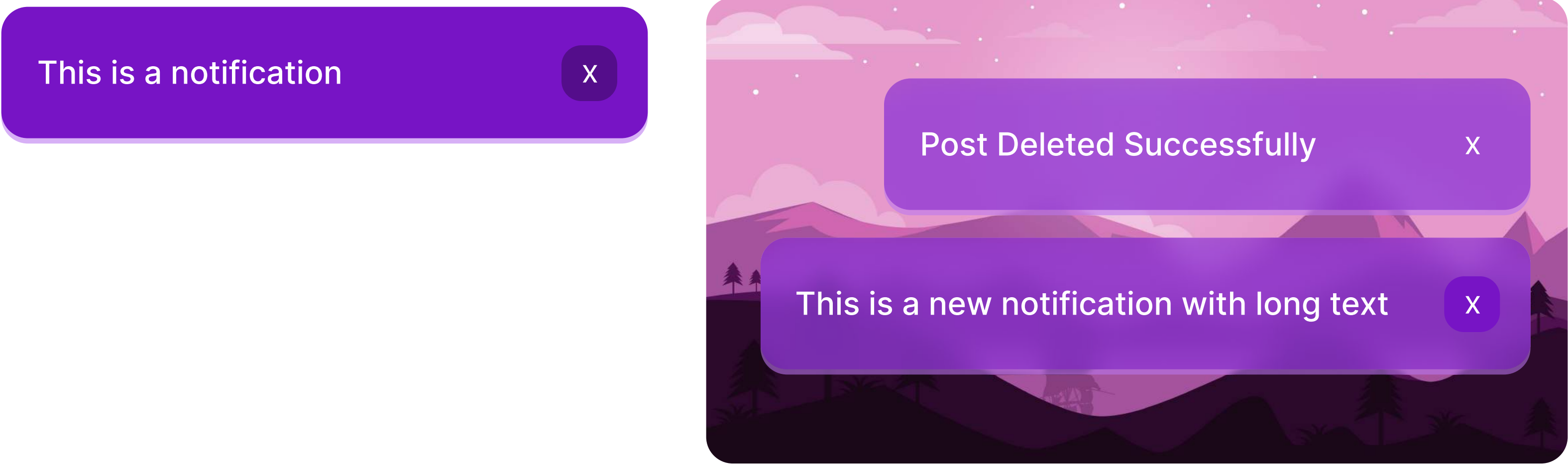
### Posts

Les publications des utilisateurs sont la principale chose sur laquelle l'utilisateur passe le plus de temps à se diriger, nous les avons donc rendues faciles à utiliser et à comprendre avec des actions minimales pour cliquer sur un bouton ou prendre une décision.



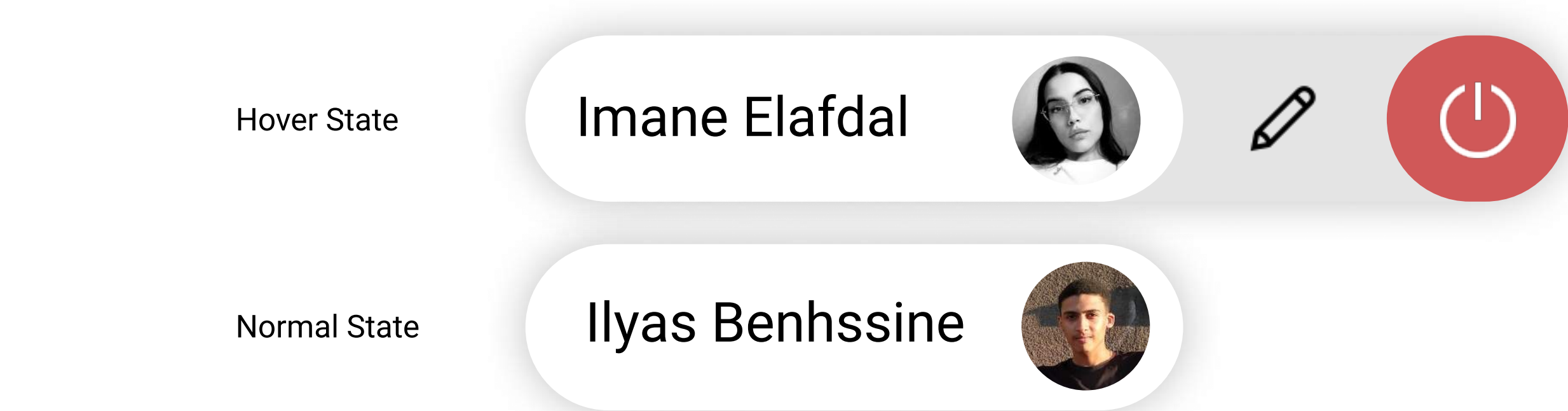
## Notifications

Les notifications sont des messages qui communiquent des informations à l'utilisateur. Les deux principaux variantes de notifications sont les notifications toast et les notifications en ligne.



## Toolbar, Bulles d'informations

Les info-bulles affichent des informations supplémentaires sur le clic, le survol ou la mise au point. Les informations doivent être contextuelles, utiles et non essentielles.



## Logo & Palette de Couleurs

Un logo est une marque graphique, un emblème, un symbole ou un nom stylisé utilisé pour identifier une entreprise, une organisation, un produit ou une marque. ... Un logo largement et instantanément reconnu est un actif incorporel précieux pour une société et est donc une marque déposée pour la protection de la propriété intellectuelle dans la majorité des situations.

Nous avons conçu le logo pour donner l'impression d'un environnement convivial où le logo représente un visage souriant «smiley» sans yeux qui représente les inconnues avec lesquelles on peut parler et discuter sans avoir à les connaître.

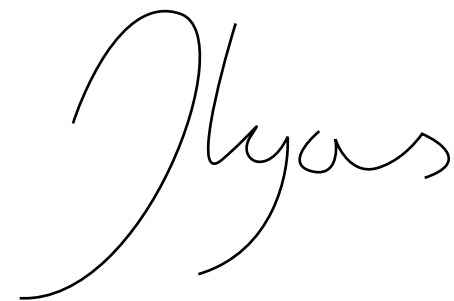


# Merci.

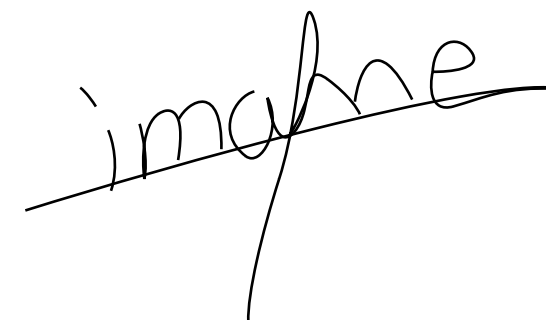
Il y a du travail et il y a du travail fait avec le cœur.

Le genre de travail que vos empreintes sont partout.  
Le genre de travail sur lequel vous ne faites jamais  
de compromis. Pour lequel vous sacrifieriez un  
week-end ou un mois.

Tout en recherchant la perfection, vous atteignez  
l'excellence.

A stylized, cursive handwritten signature in black ink, reading 'Ilyas'.

- Ilyas Benhssine

A stylized, cursive handwritten signature in black ink, reading 'Imane'.

- Imane Elafdal