



Πανεπιστήμιο Πειραιώς
University of Piraeus

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
Απαλλακτική Εργασία για το
μάθημα Εκπαιδευτικό Λογισμικό

Τεχνικό Εγχειρίδιο

Υλοποιήθηκε από τον φοιτητή

Βεργιάννη Νικόλαο - Π16170

Περιεχόμενα

1.Σημειώσεις.....	3
2. Στόχος της Εργασίας.....	3
3. Βάση Δεδομένων.....	4
4.Django Apps.....	5
4.1 settings.py.....	5
4.2 urls.py.....	6
4.UsersApp.....	7
4.1 Μοντέλο Μαθητή.....	8
4.2 Views μαθητή.....	8
4.3 Forms.....	10
5. HistoryMuseum.....	13
5.1 Μοντέλα.....	13
5.2 Views.....	14
5.3 Forms.....	17
5.4 admin.py.....	18
6. Django administration.....	18
6.1 Authentication and Authorization.....	18
6.2 Hero.....	19
6.3 Test.....	19
6.4 Student.....	20

Σημειώσεις

Η εργασία υλοποιήθηκε σε γλώσσα προγραμματισμού Python και συγκεκριμένα με το django framework το οποίο ακολουθεί **MTV(Model – Template – View)** αρχιτεκτονική και βάση δεδομένων SQLite. Υποστηρίζει επίσης κιάλλες SQL βάσεις όπως PostgreSQL, MySQL.

Για την υλοποίηση ήταν χρήσιμο το documentation του Django:

<https://docs.djangoproject.com/en/3.1/>

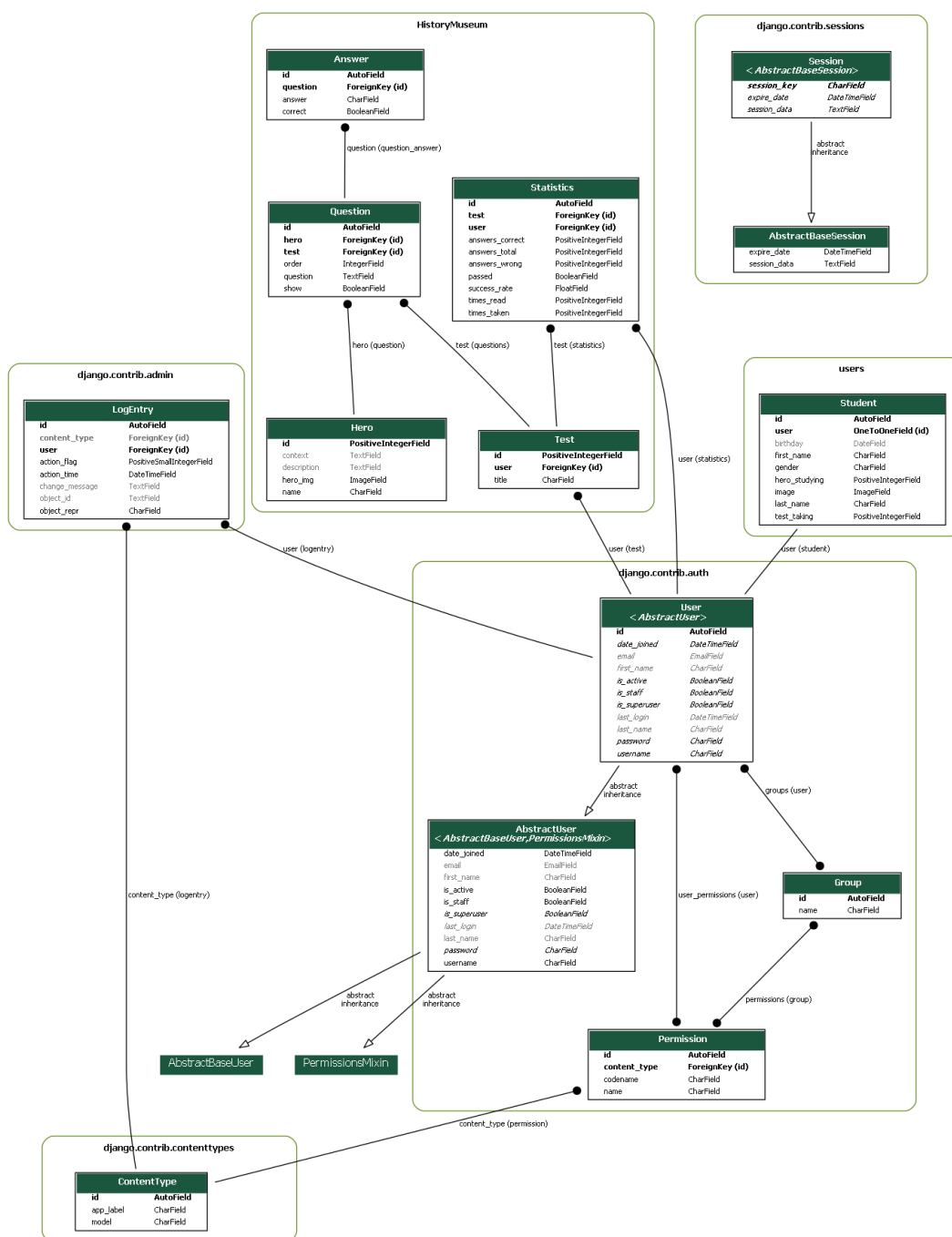
Όλα τα αρχεία και τα απαραίτητα screenshots βρίσκονται στον παραδοτέο φάκελο.

Στόχος της Εργασίας

Στόχος της εργασίας είναι η υλοποίηση ενός αλληλεπιδραστικού λογισμικού εκπαίδευσης μαθητών Πρωτοβάθμιας εκπαίδευσης για εικονικό μουσείο σχετικά με ιστορικά πρόσωπα του 1821. Η εργασία αυτή θα περιλαμβάνει τρόπους παρουσίασης του εικονικού μουσείου με στόχο να γίνει το θέμα κατανοητό και να μπορεί να απομνημονευθεί από τους μαθητές και να εμπεδωθεί η ύλη μέσω ασκήσεων. Ο κύριος σκοπός της εργασίας είναι ο καλός σχεδιασμός και υλοποίηση του εκπαιδευτικού λογισμικού (διδασκαλία – αξιολόγηση του μαθητή).

Βάση Δεδομένων

Το σχήμα και οι ή σχέση των μοντέλων της βάσης φαίνεται στην παρακάτω εικόνα. (Για πιο ευδιάκριτη ανάγνωση υπάρχει η εικόνα στον φάκελο



Django Apps

Η βάση δεδομένων δημιουργείται με τις τις εντολές python **manage.py makemigrations** και **manage.py migrate**.

- **Manage.py** : Κάθε εφαρμογή Django ξεκινά με ένα αρχείο `manage.py` στη ρίζα του. Είναι ένα script που μας επιτρέπει να εκτελούμε διαχειριστικές εργασίες, όπως το Django που περιλαμβάνεται στο `django-admin`.
- **Makemigrations**: Το **makemigrations** δημιουργεί βασικά τις εντολές SQL για προεγκατεστημένες εφαρμογές (οι οποίες μπορούν να προβληθούν σε εγκατεστημένες εφαρμογές στο `HistoricalMuseum/settings.py`) και τα νέα μοντέλα των δημιουργημένων εφαρμογών που προσθέτουμε. Δεν εκτελεί αυτές τις εντολές στο αρχείο βάσης δεδομένων.

```
INSTALLED_APPS = [  
    # own apps  
    'HistoricalMuseum.apps.HistoricalMuseumConfig',  
    'users.apps.UsersConfig',  
    'crispy_forms',  
    'nested_admin',  
    'nested_inline',  
    'django_extensions',  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

- **Migrate**: Το **migrate** εκτελεί αυτές τις εντολές SQL στο αρχείο βάσης δεδομένων. Έτσι, μετά την εκτέλεση μετεγκατάστασης, όλοι οι πίνακες των εγκατεστημένων εφαρμογών δημιουργούνται στο αρχείο της βάσης δεδομένων **db.sqlite3**

Η εφαρμογή τρέχει απο την τοπική διεύθυνση <http://127.0.0.1:8000/> έχοντας πατήσει την εντολή **python manage.py runserver**.

Settings.py

Το settings.py είναι η κεντρική διαμόρφωση για όλα τα έργα Django.

Σε αυτό το αρχείο διαφορφώνουμε πράγματα όπως εφαρμογές Django, βάσεις δεδομένων, πρότυπα κ.α.

Για την ανάπτυξη της εφαρμογής αναγκαία είναι να δηλώσουμε εκτός από τα ονόματα των εφαρμογών το static για το css αρχείο μας, το pack για της crispy forms και τα login urls,

```
STATIC_URL = '/static/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'

LOGIN_REDIRECT_URL = 'homepage'

LOGIN_URL = 'login'
```

Urls.py

Η αλληλεπίδραση μεταξύ των HTML template σελίδων γίνεται με την χρήση του αρχείου urls.py. Εκεί καθορίζεται η χαρτογράφηση(map) μεταξύ των url διευθύνσεων, μονοπατιών και προβολών.

```
from django.conf.urls.static import static
from django.contrib import admin
from HistoryMuseum import views
import nested_admin
from users.views import register, profile

urlpatterns = [
    path('', views.homepage, name='homepage'),

    #admin and auth pages
    path('admin/', admin.site.urls),
    path('register/', register, name='register'),
    path('login/', auth_views.LoginView.as_view(template_name='users/login.html'), name='login'),
    path('logout/', auth_views.LogoutView.as_view(template_name='users/logout.html'), name='logout'),
    path('profile/', profile, name='profile'),
    path('nested_admin', include('nested_admin.urls')),

    #work on theory
    path('hero/', views.HeroPageView.as_view(), name='hero'),
    path('hero/<int:hero_id>', views.TheoryPageView.as_view(), name='hero_theory'),
    path('hero/', views.HeroRedirectView.as_view(), name='hero-main'),
    path('hero/test/', views.TestRedirectView.as_view(), name='test-main'),
    path('hero/test/<int:id>', views.TestPageView.as_view(), name='test'),
    path('hero/passed/', views.PassedView.as_view(), name='passed'),
    path('hero/not_passed/', views.NotPassedView.as_view(), name='not_passed')
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Στο παρών Django project υπάρχουν 2 βασικές εφαρμογές. Μια για τους χρήστες(users) και η κεντρική για το Ιστορικό Μουσείο (HistoryMuseum).

USERS

Αρχικά πρέπει να δημιουργήσουμε έναν χρήστη που να μπορεί να συνδεθεί στην σελίδα διαχειριστή. Αυτό γίνεται με την εντολή:

```
$ python manage.py createsuper
```

Εισάγουμε

- username: admin ,
- email
- password : admin1234

Μοντέλο Μαθητή

Ο χρήστης έχει ένα βασικό μοντέλο-προφίλ Student όπου λειτουργεί σαν AbstractUser το οποίο είναι ένα πλήρες μοντέλο χρήστη ως abstract κλάση ώστε να μπορεί να κληρονομεί τα δικά μας πεδία και μεθόδους που θα εισάγουμε. Το μοντέλο Student κληρονομείται από το μοντέλο User το οποίο έχει λειτουργία ελέγχου ταυτότητας(django.contrib.auth).

```
1 from django.db import models
2 from django.contrib.auth.models import User
3 from django.template.static import static
4 from PIL import Image
5 import datetime
6
7
8
9 GENDER_CHOICES = [
10     ('M', 'Male'),
11     ('F', 'Female')
12 ]
```

```

14 class Student(models.Model):
15     user = models.OneToOneField(User, on_delete=models.CASCADE)
16     image = models.ImageField(upload_to='profile_pics', null=True)
17     first_name = models.CharField(default="", max_length=50)
18     last_name = models.CharField(default="", max_length=50)
19     birthday = models.DateField(null=True, blank=True, help_text="YYYY-MM-DD")
20     gender = models.CharField(choices=GENDER_CHOICES, max_length=1)
21
22     hero_studying = models.PositiveIntegerField(default=1)
23     test_taking = models.PositiveIntegerField(default=1)
24
25     def save(self, *args, **kwargs):
26         super(Student, self).save(*args, **kwargs)
27         img = Image.open(self.image.path)
28
29         if img.height > 300 or img.width > 300:
30             output_size = (300, 300)
31             img.thumbnail(output_size)
32             img.save(self.image.path)
33
34     def age(self):
35         if self.birthday is None:
36             pass
37         else:
38             return int((datetime.date.today() - self.birthday).days / 365.25)
39
40     def __str__(self):
41         return f'{self.user.username} Student'
42
43

```

Κάποιες λειτουργίες του είναι ο υπολογισμός της ηλικίας τους (age) βάση της ημερομηνίας που θα εισάγει ο χρήστης στο προφίλ του καθώς και την εικόνα του προφίλ η οποία αλλάζει βάση του φύλου του.

Η εντολή της αλλαγής εικόνας έρχεται απο ένα signal απο το αρχείο signals.py. Στέλνει δηλαδή ένα σήμα οτι πραγματοποιήθηκε μια ενέργεια, στην περίπτωση μας η επιλογή του φύλου του χρήστη.


```

def create_profile(sender, instance, created, **kwargs):
    if created:
        Student.objects.create(user=instance)

    |
    |
    |
    post_save.connect(create_profile, sender=User)

    #Change avatar based on gender

def set_avatar(instance):
    avatar = instance.image
    gender = instance.gender
    if gender == 'M':
        avatar = 'profile_pics/boy.png'
    elif gender == 'F':
        avatar = 'profile_pics/girl.png'
    else:
        avatar = 'profile_pics/default.jpg'
    return avatar

def save_profile(sender, instance, *args, **kwargs):
    if not instance.image:
        instance.image = set_avatar(instance)
    else:
        instance.image = set_avatar(instance)
    pre_save.connect(save_profile, sender=Student)

```

Signals στέλνονται επίσης και στην δημιουργία του χρήστη αλλά και στην ενημέρωση του Προφίλ.

Views.py

Είναι μια συνάρτηση Python που λαμβάνει ένα αίτημα Web(request) και επιστρέφει μια απόκριση Web(response). Αυτή η απόκριση μπορεί να είναι το περιεχόμενο HTML σελίδας, ή μια ανακατεύθυνση (θα δούμε στο Ιστορικό Μουσείο) ή ένα σφάλμα 404. Μπορεί να περιέχει όποια αυθαίρετη λογική είναι απαραίτητη για την επιστροφή αυτής της απόκρισης.

Για την εγγραφή αλλά και την ενημέρωση προφίλ θα καλέσουμε τις κατάλληλες φόρμες από το αρχείο forms.py (εξηγείται η χρήση τους παρακάτω) με τα κατάλληλα αιτήματα.

```

1 from django.shortcuts import render, redirect
2 from django.contrib import messages
3 from .forms import UserRegisterForm, UserUpdateForm, ProfileUpdateForm
4 from HistoryMuseum.models import Statistics
5 from django.contrib.auth.decorators import login_required
6
7
8
9 def register(request):
10     if request.method == 'POST':
11         form = UserRegisterForm(request.POST)
12         if form.is_valid():
13             form.save()
14             username = form.cleaned_data.get('username')
15             messages.success(request, f'Ο λογαριασμός δημιουργήθηκε για τον χρήστη {username}!')
16             return redirect('login')
17         else:
18             form = UserRegisterForm()
19     return render(request, 'users/register.html', {'form': form})

```

```

22 @login_required
23 def profile(request):
24     #get forms
25     if request.method == 'POST':
26         u_form = UserUpdateForm(request.POST, instance=request.user)
27         p_form = ProfileUpdateForm(request.POST,
28                                   request.FILES,
29                                   instance=request.user.student)
30         if u_form.is_valid() and p_form.is_valid():
31             user = u_form.save()
32             user.refresh_from_db()
33             p_form.save()
34             messages.success(request, f'Ο λογαριασμός σας ενημερώθηκε!')
35             return redirect('profile')
36     else:
37         u_form = UserUpdateForm(instance=request.user)
38         p_form = ProfileUpdateForm(instance=request.user.student)
39         stats = Statistics.objects.filter(user=request.user)
40
41     context = {
42         'u_form': u_form,
43         'p_form': p_form,
44         'stats': stats,
45     }
46
47     return render(request, 'users/profile.html', context)

```

Το **'POST'** αίτημα υποβάλλει δεδομένα προς επεξεργασία απο την φόρμα HTML στον προσδιορισμένο resource ή ενημερώσει το υπάρχον resource ή και τα 2 εμφανίζοντας έτσι το κατάλληλο μήνυμα για κάθε ενέργεια.

Επίσης στο προφίλ του χρήστη εμφανίζονται τα στατιστικά του κάθε Χρήστη για κάθε Τεστ τραβώντας τα αντικείμενα από το μοντέλο Statistics που βρίσκεται στο φάκελο HistoryMuseum που θα αναλύσουμε παρακάτω και τα εμφανίζει στο template view με την χρήση forloop και ενός if condition για το αν ο χρήστης έλαβε προβιβάσιμο βαθμό.

```

<legend class="border-bottom mb-4 center">Στατιστικά</legend>
<table class="centered">
  <thead>
    <tr>
      <th>Ερώτας</th>
      <th>Φορές Μελέτης Θεωρίας</th>
      <th>Περάστηκε</th>
      <th>Προσπάθειες</th>
      <th>Συνολικές Σωστές Απαντήσεις</th>
      <th>Συνολικές Λάθος Απαντήσεις</th>
      <th>% Επιτυχίας</th>
    </tr>
  </thead>
  <tbody>
    {% for stat in stats %}
      <tr>
        <td>{{ stat.test.title }}</td>
        <td>{{ stat.times_read }}</td>
        <td>{% if stat.passed %} Ναι {% else %} Όχι {% endif %}</td>
        <td>{{ stat.times_taken }}</td>
        <td>{{ stat.answers_correct }}/{{ stat.answers_total }}</td>
        <td>{{ stat.answers_wrong }}/{{ stat.answers_total }}</td>
        <td>{{ stat.success_rate|floatformat:0 }} %</td>
      </tr>
    {% endfor %}
  </tbody>
</table>
</fieldset>
</div>
</div>
{% endblock %}

```

Forms.py

Οι φόρμες Django μας γλιτώνουν πολλή δουλειά παρέχοντας ένα πλαίσιο που μας επιτρέπει να ορίσουμε φόρμες και τα πεδία τους μέσω προγραμματισμού και στη συνέχεια να χρησιμοποιήσουμε αυτά τα αντικείμενα για να δημιουργήσουμε τον κωδικά HTML της φόρμας και να χειριστούμε μεγάλο μέρος της επικύρωσης αλλά και της αλληλεπίδρασης του χρήστη.

Για την εφαρμογή μας θα χρησιμοποιήσουμε **crispy_forms** έχοντας εγκαταστήσει την κατάλληλη βιβλιοθήκη απο την εντολή `$ pip install crispy_forms` και τοποθετώντας την στις εγκατεστημένες εφαρμογές στο settings.py της εφαρμογής μας.

```

1 from django import forms
2 from django.contrib.auth.models import User
3 from django.contrib.auth.forms import UserCreationForm
4 from .models import Student
5
6 class UserRegisterForm(UserCreationForm):
7     email = forms.EmailField()
8
9     class Meta:
10         model = User
11         fields = ['username', 'email', 'password1', 'password2']
12
13 class UserUpdateForm(forms.ModelForm):
14     email = forms.EmailField()
15
16     class Meta:
17         model = User
18         fields = ['username', 'email']
19
20 class ProfileUpdateForm(forms.ModelForm):
21
22     class Meta:
23         model = Student
24         fields = ['first_name', 'last_name', 'birthday', 'gender']

```

Ένα παράδειγμα φόρμας στην ενημέρωση προφίλ:

```

1 <div class="content-section">
2   <div class="col-12 pt-4 pb-5 mb-2">
3     <form method="POST" enctype="multipart/form-data">
4       {% csrf_token %}
5       <fieldset class="form-group">
6         <legend class="border-bottom mb-4">Σπεξεργασία Λογαριασμού</legend>
7         {{ u_form|crispy }}
8         {{ p_form|crispy }}
9       </fieldset>
10      <div class="form-group">
11        <button class="btn btn-outline-info" type="submit">Ενημέρωση</button>
12      </div>
13    </form>
14  </div>

```

History Museum

Ο φάκελος περιέχει τα κατάλληλα αρχεία για την δημιουργία και την εκτέλεση της κεντρικής HTML σελίδας καθώς και την ανάπτυξη της θεωρίας, την εκτέλεση των Τεστ αυτοαξιολόγησης αλλά και την καταγραφή στατιστικών κάθε χρήστη.

Models

Οι κλάσεις μοντέλα που χρησιμοποιήθηκαν φαίνονται παρακάτω:

```
1 from django.db import models
2 from django.conf import settings
3 from django.contrib.auth.models import User
4 from django.db.models.signals import pre_save
5 from django.dispatch import receiver
6 from django.core.validators import MaxValueValidator
7
8
9
10 class Hero(models.Model):
11     id = models.PositiveIntegerField(primary_key=True)
12     name = models.CharField(max_length=50)
13     description = models.TextField(blank=True)
14     context = models.TextField(blank=True)
15     hero_img = models.ImageField(upload_to='heroes', default='')
16
17
18     def __str__(self):
19         return self.name
20
21
22 class Test(models.Model):
23     id = models.PositiveIntegerField(primary_key=True)
24     title = models.CharField(max_length=100, null=False)
25     user = models.ForeignKey(settings.AUTH_USER_MODEL, related_name='test', on_delete=models.CASCADE)
26
27     def __str__(self):
28         return f'Test: {self.title}'
29
30
```

```
31
32 class Question(models.Model):
33     question = models.TextField(max_length=255)
34     test = models.ForeignKey(Test, on_delete=models.CASCADE, related_name='questions')
35     hero = models.ForeignKey(Hero, on_delete=models.CASCADE)
36     order = models.IntegerField(default=0)
37     show = models.BooleanField(default=True)
38
39     class Meta:
40         ordering = ['order',]
41
42     def __str__(self):
43         return self.question
44
45
46 class Answer(models.Model):
47     answer = models.CharField('Answer', max_length=255)
48     question = models.ForeignKey(Question, on_delete=models.CASCADE, related_name='question_answer')
49     correct = models.BooleanField(default=False)
50
51
52     def __str__(self):
53         return f'{self.answer}'
54
55
```

```

class Statistics(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    test = models.ForeignKey(Test, on_delete=models.CASCADE)

    times_read = models.PositiveIntegerField(default=0)
    times_taken = models.PositiveIntegerField(default=0)
    answers_total = models.PositiveIntegerField(default=0)
    answers_correct = models.PositiveIntegerField(default=0)
    answers_wrong = models.PositiveIntegerField(default=0)
    success_rate = models.FloatField(default=0.0)

    passed = models.BooleanField(default=False)

    def __str__(self):
        return f'{self.user}, for {self.test}'

```

Το μοντέλο Test όπου κληρονομεί το μοντέλο ερώτηση το οποίο κληρονομεί το μοντέλο απάντηση.

Καθε τεστ δηλαδή έχει ερωτήσεις και κάθε ερώτηση που αφορά το μοντέλο Ήρωας έχει απαντήσεις (δηλώνουμε μέχρι 4 στο admins.py) και μια σωστή.

Τα Στατιστικά κάθε χρηστη κληρονομούνται όπως βλέπουμε και στο σχήμα της βάσης απο το id του Τεστ αλλά και του id του εκάστοτε Χρήστη.

Views.py

Request αιτήματα για την κεντρική ιστοσελίδα και την εμφάνιση όλων των Ηρώων για μελέτη.

```

12 def homepage(request):
13     context = {
14
15     }
16     return render(request, 'HistoryMuseum/homepage.html', context)
17
18
19
20
21
22 class HeroPageView(LoginRequiredMixin, TemplateView):
23
24     def get(self, request, *args, **kwargs):
25         user = request.user
26
27         heroes = Hero.objects.all()
28         context = {}
29         context['heroes'] = heroes
30     }
31
32     return render(request, 'HistoryMuseum/hero.html', context)
33
34

```

Οι ήρωες εμφανίζονταν σε card view μέσω ενός forloop στην hero.html σελίδα


```

<div class="row row-cols-1 row-cols-md-2">
  {% for hero in heroes %}
    <div class="col-md-4 col-lg-4">
      <div class="card">

        <!-- Card -->
        <!--Card image-->
        <div class="card-image">
          <a href="{% url 'hero_theory' hero.id %}">
            <img src= "{{ hero.hero_img.url }}"
              alt="Card image cap">
          <a href="{% url 'hero_theory' hero.id %}">
            </a>
          </div>
        </div>
        <!--Card content-->
        <div class="container">
          <!--Title-->
          <h4>{{hero.name}}</h4>
          <!--Text-->
          <p class="card-text">{{hero.description}}</p>
          <a href="{% url 'hero_theory' hero.id %}">Πάτα εδώ για να διαβάσεις την θεωρία</a>
        </div>
      </div>
    </div>
  {% endfor %}

```

Η θεωρία του κάθε ήρωα εμφανίζεται πατώντας το κουμπί που μας καθοδηγεί μέσω του αρχείου urls.py στην TheoryPageView κλάση όπου κάθε φορά που ο χρήστης διαβάζει την θεωρία αποθηκεύεται σε ένα μετρητή και καταγράφεται στα στατιστικά του.

Επίσης φορτώνεται και το τεστ του εκάστοτε ήρωα.

```

class TheoryPageView(LoginRequiredMixin, TemplateView):
    template_name = 'HistoryMuseum/theory.html'

    def get(self, request, hero_id, *args, **kwargs):
        user = request.user

        try:
            hero_theory = Hero.objects.filter(id=hero_id)
        except:
            raise Http404('Hero does not exist')

        if hero_id is None:
            hero_id = user.student.hero_studying

        try:
            t = Test.objects.get(id=hero_id)
            stats, created = Statistics.objects.get_or_create(user=user, test=t)
        except:
            raise Http404('Hero should be followed by Test')

        stats.times_read += 1
        stats.save()
        context = {
            'hero_theory': hero_theory,
            'hero_id': hero_id
        }

        return render(request, self.template_name, context)

```

Με το TestPageView εμφανίζεται η σελίδα του κάθε τέστ βάσει του id.

```
80 class TestPageView(LoginRequiredMixin, TemplateView):
81     template_name = 'HistoryMuseum/test.html'
82
83     def get(self, request, id, *args, **kwargs):
84         user = request.user
85
86         if id is None:
87             id = user.student.hero_studying
88
89         # if id > user.student.hero_studying:
90         #     return redirect(f'/hero/test/{user.student.test_taking}/')
91
92         try:
93             t = Test.objects.get(id=id)
94
95         except:
96             return Http404()
97
98         #count questions
99         limit = Question.objects.filter(test=t, show=True).count()
100
101         form = TestForm(hero=id, limit=limit)
102
103         context = {
104             'form': form,
105             'test': t
106         }
107
108         return render(request, self.template_name, context)
```

Το οποίο τέστ δημιουργείται απο την φόρμα TestForm και καλεται απο την μέθοδο post.

Limit είναι το σύνολο των ερωτήσεων το οποίο αυξάνεται για κάθε ερώτηση που προσθέτουμε.

Για κάθε απάντηση ενημερώνονται τα στατιστικά συνολικών απαντήσεων, σωστών και λάθων καθώς και το συνολικό σκορ των τεστ και το σκορ του εκάστοτε τεστ που πραγματοποιεί ο μαθητής.


```

def post(self, request, id, *args, **kwargs):
    user = request.user
    t = Test.objects.get(id=id)
    limit = Question.objects.filter(test=t, show=True).count()
    form = TestForm(request.POST, hero=id, limit=limit)

    if form.is_valid():

        stats, created = Statistics.objects.get_or_create(user=user, test=t)
        data = form.cleaned_data

        current_correct_answers = 0

        for i in range(limit):

            user_answer = data.get(f'question{i}')

            if user_answer.correct:
                current_correct_answers += 1

        stats.answers_correct += current_correct_answers
        stats.answers_wrong += limit - current_correct_answers
        stats.answers_total += limit

        stats.times_taken += 1
        stats.success_rate = round(stats.answers_correct / stats.answers_total, 2) * 100
        stats.score = round(current_correct_answers / limit, 2) * 100

        #score > 60% passed
        if current_correct_answers / limit >= 0.6:
            stats.passed = True

        else:
            stats.passed = False
            user.save()
            stats.save()
            return redirect(f'/hero/not_passed')

        user.save()
        stats.save()
        return redirect(f'/hero/passed')

    context = {
        'form': form,
        'test': t,
        'stats': stats,
    }

    return render(request, self.template_name, context)

```

Αν ο βαθμός είναι πάνω από 60% τότε ο χρήστης πέρασε επιτυχώς το τεστ ενημερώνοντας έτσι τα στατιστικά του και εμφανίζοντας του την ανάλογη σελίδα.

Η φόρμα του Τεστ φαίνεται παρακάτω:

```

class TestForm(forms.Form):
    limit= 0
    user = None

    def __init__(self, *args, **kwargs):
        hero = kwargs.pop('hero')
        self.limit = kwargs.pop('limit')
        super(TestForm, self).__init__(*args, **kwargs)

        #get test for current hero
        self.test = Test.objects.get(id=hero)

        self.questions = self.test.questions.filter(show=True)

        if self.limit < len(self.questions):
            self.questions = self.questions[:self.limit]

        widget = forms.Select(attrs={'class': 'browser-default'})

        #create question for number of limit
        for counter, q in enumerate(self.questions):
            self.fields[f'question{counter}'] = forms.ModelChoiceField(label=q,
                                                                           queryset=Answer.objects.filter(question=q,
                                                                           question__test=self.test),
                                                                           required=True,
                                                                           empty_label=None,
                                                                           widget=widget )

```

Η φόρμα δημιουργείται κατά την εκτέλεση ενημερώνοντας το limit για κάθε ερώτηση που επιλέγουμε να εμφανίζεται σε ένα πεδίο πολλαπλής επιλογής για κάθε ερώτηση.

Το widget είναι η αναπαράσταση του Django για ένα στοιχείο εισαγωγής HTML όπου κάθε ερώτηση εμφανίζει τις αντίστοιχες απαντήσεις της.

Admin.py

Το αρχείο **admin.py** χρησιμοποιείται για την εμφάνιση των μοντέλων στον πίνακα διαχείρισης Django (django admin). Έχω προσαρμόσει τον πίνακα αυτόν χρησιμοποιώντας την βιβλιοθήκη nested_admin ώστε να εμφανίζεται το κάθε Τεστ με τις ερωτήσεις και τις απαντήσεις κάνοντας έτσι την εισαγωγή και την διαχείριση ευκολότερη(θα το δούμε παρακάτω στην σελίδα admin)

```

from django.contrib import admin
from nested_admin.nested import NestedStackedInline, NestedModelAdmin, NestedTabularInline

from .models import Hero, Test, Question, Answer, Statistics
# Register your models here.

class AnswerInline(NestedTabularInline):
    model = Answer
    extra = 4
    max_num = 4

class QuestionInline(NestedTabularInline):
    model = Question
    inlines = [AnswerInline,]
    extra = 1

class TestAdmin(NestedModelAdmin):
    inlines = [QuestionInline]

admin.site.register(Statistics)

admin.site.register(Hero)
admin.site.register(Test, TestAdmin)
admin.site.register(Question)
admin.site.register(Answer)

```

Django administration

Ενα απο τα πιο ισχυρά μέρη του Django είναι η αυτόματη διεπαφή διαχειριστή. Διαβάζει τα μεταδεδομένα απο τα μοντέλα για να παρέχει μια γρήγορη διεπαφή με επίκεντρο το μοντέλο, όπου οι χρήστες με εξουσιοδότηση μπορούν να διαχειρίζονται τον ιστότοπο.

Η χρήση του είναι κυρίως σαν εργαλείο εσωτερικής διαχείρισης .

Django administration
WELCOME, ADMIN / VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	Change
Users	+ Add	Change

HISTORY/MUSEUM

Answers	+ Add	Change
Heroes	+ Add	Change
Questions	+ Add	Change
Statistics	+ Add	Change
Tests	+ Add	Change

USERS

Students	+ Add	Change
----------	-------	--------

Recent actions

My actions

- Test: Θεόδωρος Κολοκατρώνης
- Test: Θεόδωρος Κολοκατρώνης
- Test: Οδυσσέας Ανδρούτσος
- Test: Παπαφλέσσας
- Test: Οδυσσέας Ανδρούτσος
- Test: Μαντώ Μαυρογένους
- Test: Λασκαρίνα Μπουμπουλίνη
- Test: Μάρκος Μπόταρης
- Test: Αθανάσιος Διάκος
- Test: Γεώργιος Καραϊσκάκης

Authentication and authorization:

Το Django διαθέτει σύστημα ελέγχου ταυτότητας χρήστη. Διαχειρίζεται λογαριασμούς χρηστών, ομάδες, δικαιώματα και συνεδρίες χρηστών βάσει cookie.

Select user to change

Search

Action: 0 of 3 selected

<input type="checkbox"/>	USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
<input type="checkbox"/>	TestUser	test@gmail.com			✗
<input type="checkbox"/>	admin	nikosvg7@gmail.com			✔
<input type="checkbox"/>	testuser	testuser@gmail.com			✗

3 users

FILTER

By staff status

☐ All

☐ Yes

☐ No

By superuser status

☐ All

☐ Yes

☐ No

By active

☐ All

☐ Yes

☐ No

ADD USER +

Ο superuser admin που δημιουργήθηκε απο την γραμμή εντολών που αναφέραμε παραπάνω **\$python manage.py createsuper** μπορεί να διαχειριστεί τα δικαιώματα κάθε χρήστη καθώς και να προσθέσει καινούργιο χρήστη με δικαιώματα διαχειριστή.

Hero

Πατώντας το κουμπί add στην κατηγορία Hero μπορούμε να δημιουργήσουμε έναν καινούργιο ήρωα καθώς και το περιεχόμενο που θέλουμε να εμφανίζεται βάση του μοντέλου μας.

Change hero

HISTORY

id:

Name:

Description:

Ηγετική μορφή της Ελληνικής Επανάστασης, που έδρασε στην Πελοπόννησο και εξ αυτού του λόγου είναι γνωστός και ως «Γέρος του Μωριά».

Context:

Ηγετική μορφή της Ελληνικής Επανάστασης, που έδρασε στην Πελοπόννησο και εξ αυτού του λόγου είναι γνωστός και ως «Γέρος του Μωριά».

Hero img:

Currently: heros/Theodoros_Kolokotronis.jpg


Change: No file chosen



Θα δημιουργήσουμε λοιπόν τα τεστ πατώντας το Add button και συμπληρώνοντας τα κατάλληλα πεδία όπως το id του Test τον τίτλο και κάθε ερώτηση με τις απαντήσεις τις και την σειρά που θέλουμε να εμφανίζονται. Μπορούμε να προσθέσουμε νέα ερώτηση εμφανίζοντας τα ίδια πεδία.





Add test


Id:

Title:

User: 

QUESTIONS	HERO	ORDER	SHOW	DELETE?
<div style="border: 1px solid #ccc; height: 100px; width: 100%;"></div>	<input type="text"/> 	<input type="text" value="0"/>	<input checked="" type="checkbox"/>	

ANSWERS	CORRECT	DELETE?
<input type="text"/>	<input type="checkbox"/>	
<input type="text"/>	<input type="checkbox"/>	
<input type="text"/>	<input type="checkbox"/>	
<input type="text"/>	<input type="checkbox"/>	

 Add another Question

Τέλος η σελίδα διαχείρισης των μαθητών το οποίο ενημερώνεται αυτόματα κατα την ενημέρωση του προφίλ του μαθητή.

Home > Users > Students > testUser Student

Change student HISTORY

User: 

Image: Currently: profile_pics/girl.png
Change: No file chosen

First name:

Last name:

Birth-day: Today! 
YYYY-MM-DD

Note: You are 3 hours ahead of server time.

Gender:

Hero studying:

Test taking:

Όλα τα δεδομένα της σελίδας διαχείρισης Django είναι επεξεργάσιμα και διαχειρίζονται απο τον διαχειριστή

