UDC 004.41 DOI: 10.30748/soi.2019.157.18

Y. Parfonov

Simon Kuznets Kharkiv National University of Economics, Kharkiv

MANAGING USER DATA OF THE WEB APPLICATION FOR COMPUTER-BASED TESTING ACADEMIC PERFORMANCE OF THE STUDENTS

The paper deals with the Django-based web application for testing the level of academic achievement of the students. Considered issues related to the implementation of a subsystem for managing user data of the application. Justified the need in custom user model for the application use case. Created user profile models and a user data model that extends the AbstractUser model. Described the developed subsystem and its source code. The use of the subsystem via the Django admin application allows you to select a user profile in runtime depending on the type of user.

Keywords: *web application, Python, Django, Django admin application, user model, user profile, user type.*

Introduction

In 2018, the author registered the copyright for the Knowledge Assessment System (KnAS) software product designed for computer assessment the level of the students' academic achievement [1].

The KnAS provides such features as random selection of questions from the relevant bank, random order of questions and answers, automatic grading, setting the test duration, using an encrypted question bank, and recording results. The application is a cross-platform software based on the Java SE software platform and object-oriented programming language Java. It can be stored on a removable media, for example, a flash drive, and run from it.

However, it has certain disadvantages. In particular, the KnAS is a desktop application, so it must be deployed on each client computer. Practical use of the KnAS showed that testing without a significant reduction in its objectivity is possible in groups of up to five people simultaneously. Furthermore, there is also unproductive time spent on connecting several flash drives to computers, restarting the application after the end of the current testing session, and changing test participants at the workplaces. It can reach ten minutes. Thus, administering the test to all students of the academic group during one class in most cases is impossible. To overcome these drawbacks a new web application [2] was developed using the basic ideas previously implemented in the KnAS. In the paper, it will be called the KnAS Online.

KnAS Online created using the modern server-side Django web framework [3] and Python programming language.

One of the application requirements is the presence of the subsystem for authentication and authorization users.

Django contains a built-in user authentication and authorization system [4], which includes such elements as users, groups, access rights, as well as the necessary forms and views.

In the system, user data (username, password, first name, last name, email, and others) are described by the User model, which is a Python class. This model can be used "as is" in various web applications, especially those having only one user type for whom no additional information is required to be stored. However, in more complex cases when the application has users of the different types with specific additional data, it is recommended to develop a user data model at the very beginning of the project [5]. Also, the important problem is managing user data, which is determined by the application requirements.

Thus, **the purpose of this article** is to justify the architecture of the user data management subsystem of the KnAS Online web application.

Statement of basic materials

KnAS Online has two types of "ordinary" users: a Student and a Teacher. Teachers can create, delete, edit tests, and view the students' test results via the Django admin application. These actions permitted only for academic disciplines they assigned to. Teachers can manage users of the Student type as well.

Students can register in the system by filling out the relevant form. Registered users of the Student type allowed to take tests in some academic disciplines and view only their own results.

All the users have common data, such as username, password, last name, first name, and middle name. The Students' data, besides, contain information about the year of his or her study and the academic group code. Extra Teachers' data include his or her position and the name of the department.

So, to implement the KnAS Online user model, it is necessary to modify the built-in User model or re-

place it completely. Based on the analysis of possible approaches for solving this problem, described in [4–9], it was decided to create a custom user model. Because of its similarity to the built-in user model (class User), it inherits from the AbstractUser class.

To store extra user data of the Students and Teachers, often called “user profile”, the Student and Teacher models are implemented. You can see pseudocode of all these models below.

```
# Custom User model:
class CustomUser(AbstractUser):
   first_name=CharField(blank=False)
   middle_name=CharField(blank=False)
   last_name=CharField(blank=False)
   USER_TYPE_CHOICES= (STUDENT, TEACHER)
   user_type = Num(choices=USER_TYPE_CHOICES)
   class Meta:
      unique_together = ('last_name', 'first_name',
                                        'middle_name')
```

Here, the variables first_name and last_name override the corresponding fields of the AbstractUser class to be mandatory. The middle name field is also required. The user_type field specifies the type of user (Student or Teacher). The value of the unique together field of the internal class Meta ensures the uniqueness of the user's full name.

```
# Teacher model
class Teacher(Model):
   user = OneToOneField(CustomUser)
   TYPE_OF_POSITION = (SL, AP, FP)
   position =
      CharField(choices=TYPE_OF_POSITION,
                                          blank=False)
   department = ForeignKey('Department')

# Student model
class Student(Model):
   user = OneToOneField(CustomUser)
   year = Num()
   academic_group = ForeignKey('AcademicGroup')
```

The Teacher and Student classes both contain a user field. It defines a one-to-one relationship between the CustomUser model and Student or Teacher model. Thus, each user has a single profile and each profile belongs to the single user. The remaining fields of these classes define the data specific to the respective models.

To be able to manage user data through the Django admin application, at least, you must register the CustomUser model in the admin.py file. It might look like this:

```
admin.site.register (CustomUser)
```

But, in cases like that described in this article, apart from registering the model, we need to customize it [5; 6; 10]. That affects both the logic of the admin application and its user interface.

According to the requirements for KnAS Online, each user of the application must have an associated profile, which depends on the user type. Thus, the Django admin application should provide two abilities. Firstly, to work with user profile when creating or editing the user. Secondly, to select the current user profile dynamically at run time.

The first problem solved by using the so-called "inlines", that is, one or several dependent models within the parent model.

First, let's define “inlines” (pseudocode):

```
# “inline” based on Teacher model class Teacher-
Inline(StackedInline):
   model = Teacher
   verbose_name_plural = 'Teacher info'

# “inline” based on Student model
class StudentInline(StackedInline):
   model = Student
   verbose_name_plural = 'Student info'
```

Here, the value of the verbose_name_plural field defines the subtitle displayed in the user interface of the admin application.

Next, you need to define a class that represents the CustomUser model in the admin application. The minimum required code for this class is shown below:

```
@admin.register(CustomUser)
class CustomUserAdmin(UserAdmin):
   inlines = (TeacherInline, StudentInline,)
```

Here @admin.register is the decorator for registering the CustomUser model using our CustomUserAdmin class. The value of the inlines field is a list of dependent models.

However, considered implementation of the CustomUserAdmin class does not allow to utilize user profile forms dynamically. It leads to the user creation page contains not only form for entering regular user data but also two forms to enter profile data of the Teacher and Student types. In the literature on the Django framework, information about ways to solve this problem is practically absent, except for [10]. The source mentions that it is necessary to override the generator method get_formsets_with_inlines of the class UserAdmin and gives a small example.

As a result of analyzing possible options to implement this method, the following code was added to the CustomUserAdmin class. Thanks to that the profile

form of the newly created user corresponding to its type appears only on the editing page.

```
# overridden generator function (pseudocode)
def get_formsets_with_inlines():
   if custom_user created:
      for inline in inline_instances():
         if custom_user_type  equals TEACHER and
                        inline is instance of TeacherInline:
            custom_user_is_staff = True
            yield (custom_user, inline)
         elif custom_user_type equals STUDENT and
                        inline is instance of StudentInline:
            yield (custom_user, inline)
```

Now, the page of the admin application for adding a new user will display only form for entering the username, password, and password confirmation.

But, as defined in the CustomUser class, each user must also have the last name, first name, and middle name. In addition, to select the user profile form shown on the edit page, you must be able to enter its type beforehand. To do this, override the add_fieldsets field in the CustomUserAdmin class as follows:

```
add_fieldsets = UserAdmin.add_fieldsets + (
      (None, {'fields': ('user_type',)}),
      ('Personal info', {'fields': ('last_name', 'first_name',
                                    'middle_name')}),)
```

The appearance of this page is shown in fig. 1.

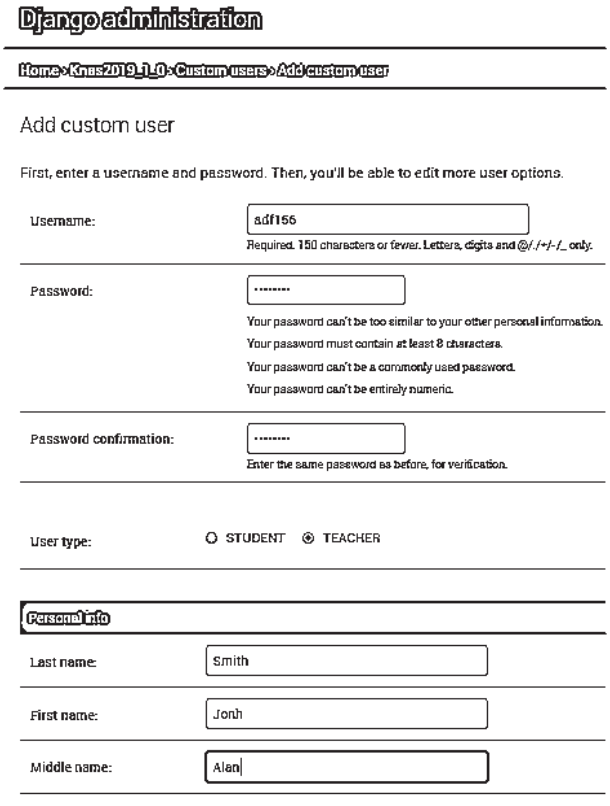

Fig. 1. The Add user page

In order for the edit page to display only the username and form for editing user profile, you must also override the fieldsets field in the CustomUserAdmin class:

```
fieldsets = (
      (None, {'fields': ('username',)}),
      ('Personal info', {'fields': ('last_name', 'first_name',
                                    'middle_name')}),)
```

As a result, the user edit page will have the look shown in fig. 2.

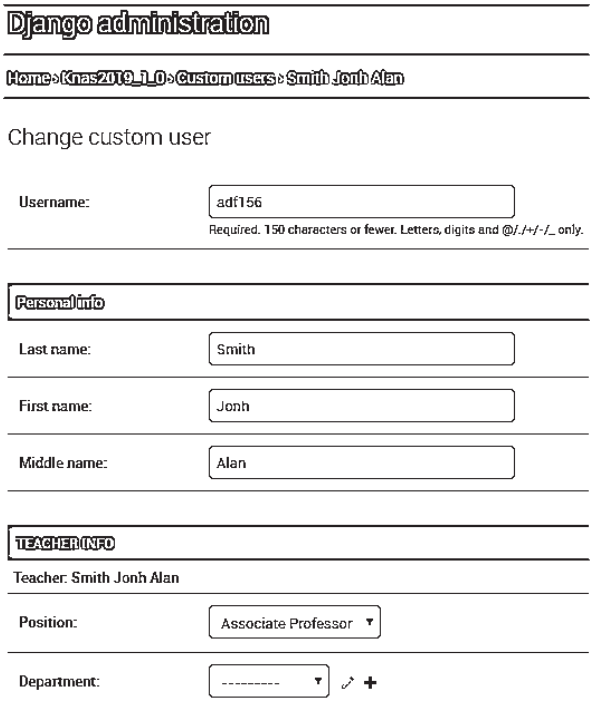

Fig. 2. The Edit user page

Conclusion

Thus, the article discusses the particularities of the implementation user data management subsystem of the KnAS Online web application and its utilization with the Django admin site.

Developed the custom user model of the application based on the abstract user model of the Django framework. The created model takes into account the specifics of the domain, as well as the application requirements. For each user type designed model of their profile.

Proposed the structure of the user data management subsystem, presented the software implementation of its elements, and interfaces of human-machine interaction.

The results can be applied in the development of user management subsystems for Django-based web applications with several types of users when the profile of each of them must be dynamically selected in the application runtime.

In the future, it is planned to improve the developed subsystem by adding two-factor user authentication and implementing the ability to import user data.

Список літератури

1. Парфьонов Ю.Е. Комп'ютерна програма "Knowledge Assessment System" (KnAS): свідоцтво про реєстрацію авторського права на твір № 77613, Авторське право і суміжні права: офіційний бюлетень, 2018. – № 48. – С. 743.

2. Parfonov Y.E. Use of the Django framework in the web application for computer-based testing of higher education institution students / Y.E. Parfonov, Y.V. Zmiievska // Матеріали X-ї Ювілейної Міжнародної науково-практичної конференції "Free and Open Source Software", 20 – 22 листопада 2018 р. – Харків: ХНУБА, 2018. – С. 47.

3. The official site of the Django Software Foundation [Електронний ресурс]. – Режим доступу: www.djangoproject.com/foundation.

4. The official site of the Django documentation. User authentication in Django [Електронний ресурс]. – Режим доступу: docs.djangoproject.com/en/2.2/topics/auth/.

5. Rubio D. Beginning Django: Web Application Development and Deployment with Python / D. Rubio. – Apress, Berkeley, CA. – 2017/ – 593 p. https://doi.org/10.1007/978-1-4842-2787-9.

6. Mele A. Django by Example / A. Mele. – Packt Publishing Limited, Birmingham, 2015. – 474 p.

7. Freitas V. How to Extend Django User Model [Електронний ресурс] / V. Freitas. – Режим доступу: simpleisbetterthancomplex.com/tutorial/2016/07/22/how-to-extend-django-user-model.html.

8. Vincent W.S. Django: How to Extend The User Model (aka Custom User Model) [Електронний ресурс] / W.S. Vincent. – Режим доступу: wsvincent.com/django-custom-user-model-tutorial/.

9. Freitas V. How to Implement Multiple User Types with Django [Електронний ресурс] / V. Freitas. – Режим доступу: simpleisbetterthancomplex.com/tutorial/2018/01/18/how-to-implement-multiple-user-types-with-django.html.

10. The official site of the Django documentation. The Django admin site [Електронний ресурс]. – Режим доступу: docs.djangoproject.com/en/2.1/ref/contrib/admin/.

References

1. Parfonov, Y.E. (2018), "Komp'juterna proghrama "Knowledge Assessment System" (KnAS): svidoctvo pro rejestraciju avtorsjkogho prava na tvir No. 77613" [Computer program "Knowledge Assessment System" (KnAS): certificate of registration of copyright for work No. 77613], *Avtors'ke pravo i sumizhni prava: ofitsiynyy bulletin*, No. 48. pp. 743.

2. Parfonov, Y.E. and Zmiievska, Y.V. (2018), Use of the Django framework in the web application for computer-based testing of higher education institution students, *Materialy 10-i Juvilejnoji Mizhnarodnoji naukovo-praktychnoji konferenciji "Free and Open Source Software", 20-22 lystopada*, KhNUBA, Kharkiv, pp. 47.

3. The official site of the Django Software Foundation (2019), *About the Django Software Foundation*, available at: www.djangoproject.com/foundation (accessed 1 February 2019).

4. The official site of the Django documentation (2019), *User authentication in Django*, available at: docs.djangoproject.com/en/2.2/topics/auth/ (accessed 4 February 2019).

5. Rubio, D. (2017), *Beginning Django: Web Application Development and Deployment with Python*, Apress, Berkeley, CA, 593 p. https://doi.org/10.1007/978-1-4842-2787-9.

6. Mele, A. (2015), *Django by Example*, Packt Publishing Limited, Birmingham, 474 p.

7. Freitas, V. (2016), *How to Extend Django User Model*, available at: simpleisbetterthancomplex.com/tutorial/2016/07/22/how-to-extend-django-user-model.html (accessed 15 January 2019).

8. Vincent, W.S. (2018), *Django: How to Extend The User Model (aka Custom User Model)*, available at: wsvincent.com/django-custom-user-model-tutorial/ (accessed 21 January 2019).

9. Freitas, V. (2018), *How to Implement Multiple User Types with Django*, available at: simpleisbetterthancomplex.com/tutorial/2018/01/18/how-to-implement-multiple-user-types-with-django.html (accessed 8 February 2019).

10. The official site of the Django documentation (2019), *The Django admin site*, available at: docs.djangoproject.com/en/2.1/ref/contrib/admin/ (accessed 3 January 2019).

Received by Editorial Board 26.03.2019
Signed for Printing 23.04.2019

Відомості про автора:

Парфьонов Юрій Едуардович
кандидат технічних наук старший науковий співробітник
доцент кафедри
Харківського національного економічного університету
ім. С. Кузнеця,
Харків, Україна
https://orcid.org/0000-0003-3943-3201

Information about the author:

Yurii Parfonov
Candidate of Technical Sciences Senior Research
Senior Lecturer
of Simon Kuznets Kharkiv National
University of Economics,
Kharkiv, Ukraine
https://orcid.org/0000-0003-3943-3201

УПРАВЛІННЯ ДАНИМИ КОРИСТУВАЧІВ ВЕБ-ЗАСТОСУНКУ ДЛЯ КОМП'ЮТЕРНОГО ТЕСТУВАННЯ НАВЧАЛЬНИХ ДОСЯГНЕНЬ СТУДЕНТІВ

Ю.Е. Парфьонов

У 2018 році автором зареєстровано авторське право на програмний продукт "Knowledge Assessment System" (KnAS), призначений для комп'ютерного оцінювання рівня навчальних досягнень студентів.

Однак, йому притаманні певні недоліки. Зокрема, KnAS – це настільна програма, тому вона має бути розгорнута на кожному клієнтському комп'ютері. Практичне використання KnAS показало, що тестування без значного зменшення його об'єктивності можливе в групах до п'яти осіб одночасно. Крім того, існують непродуктивні витрати часу на підключення декількох флеш-накопичувачів до комп'ютерів, перезапуск програми після закінчення поточного сеансу тестування і зміну учасників тесту на робочих місцях. Вони можуть досягати десяти хвилин. Таким чином, тестування всіх студентів академічної групи протягом одного заняття в більшості випадків неможливо.

Для подолання цих недоліків пропонується створити веб-застосунок KnAS Online. на базі фреймворку Django. KnAS Online має є два типи "звичайних" користувачів: студент і викладач Для забезпечення можливості управління даними користувачів застосунку з використанням сайту адміністратора Django, по-перше, доцільно при створенні нового користувача і подальших операціях з ним мати можливість роботи з його профілем. По-друге, профіль поточного користувача повинен вибиратися динамічна під час виконання.

Розглянуті проблеми, пов'язані з реалізацією підсистеми управління даними користувачів застосунку. Обґрунтовано необхідність створення власної моделі користувача. Створені моделі профілів користувачів, а також модель даних користувача, яка розширює модель AbstractUser. Описана розроблена підсистема та її вихідний код. Використання даної підсистеми за допомогою сайту адміністратора Django дозволяє вибирати профіль користувача під час виконання в залежності від типу користувача.

Ключові слова: *веб-застосунок, Python, Django, сайт адміністратора, модель користувача, профіль користувача, тип користувача.*

УПРАВЛЕНИЕ ДАННЫМИ ПОЛЬЗОВАТЕЛЕЙ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ КОМПЬЮТЕРНОГО ТЕСТИРОВАНИЯ УЧЕБНЫХ ДОСТИЖЕНИЙ СТУДЕНТОВ

Ю.Э. Парфенов

Статья посвящена вопросам разработки веб-приложения на базе фреймворка Django для тестирования уровня успеваемости студентов. Рассмотрены проблемы, связанные с реализацией подсистемы управления данными пользователей приложения. Обоснована необходимость создания собственной модели пользователя. Созданы модели профилей пользователей, а также модель данных пользователя, которая расширяет модель AbstractUser. Описана разработанная подсистема и ее исходный код. Использование данной подсистемы посредством сайта администратора Django позволяет выбирать профиль пользователя во время выполнения в зависимости от типа пользователя.

Ключевые слова: *веб-приложение, Python, Django, сайт администратора, модель пользователя, профиль, тип пользователя.*