نظام إدارة الإيجارات في سلطنة عمان

متكامل Django دليل تعليمي شامل لتطوير نظام

المؤلف: Manus AI
التاريخ: يونيو 2025
الإصدار: 1.0

فهرس المحتويات

مقدمة المشروع

يُعد نظام إدارة الإيجارات في سلطنة عمان مشروعاً تعليمياً شاملاً يهدف إلى تعليم تطوير تطبيقات الويب Django باستخدام إطار العمل. يجمع هذا المشروع بين التقنيات الحديثة والتصميم الثقافي الأصيل، مما يجعله مثالاً مثالياً لتعلم تطوير التطبيقات الحقيقية التي تخدم السوق المحلي.

أهداف المشروع التعليمية

يسعى هذا المشروع إلى تحقيق عدة أهداف تعليمية مهمة تشمل فهم أساسيات تطوير تطبيقات الويب باستخدام Django، وتعلم كيفية تصميم قواعد البيانات المعقدة، وإتقان تقنيات الترجمة والتدويل، وتطوير واجهات مستخدم متجاوبة وجذابة. كما يهدف إلى تعليم أفضل الممارسات في تطوير البرمجيات وإدارة المشاريع التقنية.

إن اختيار موضوع إدارة الإيجارات ليس عشوائياً، بل يأتي من كونه يمثل حاجة حقيقية في السوق العماني، حيث يشهد القطاع العقاري نمواً مستمراً ويحتاج إلى حلول تقنية متطورة. هذا الواقع يجعل المشروع ذا قيمة عملية حقيقية بالإضافة إلى قيمته التعليمية، مما يحفز المتعلمين على الاستمرار والتطوير.

الخصائص الرئيسية للنظام

يتميز النظام بمجموعة شاملة من الخصائص التي تغطي جميع جوانب إدارة العقارات والإيجارات. يشمل ذلك إدارة المباني بتفاصيلها الكاملة من الطوابق والوحدات، وإدارة المستأجرين مع معلوماتهم الشخصية ووثائقهم، وإدارة عقود الإيجار مع إمكانيات التجديد والإنهاء، وإدارة الدفعات والإيصالات، ونظام شامل لطلبات الصيانة، بالإضافة إلى تقارير مالية وإدارية متقدمة.

ما يميز هذا النظام عن غيره هو التركيز على الخصائص المحلية العمانية، حيث يدعم اللغة العربية بشكل كامل مع نظام RTL (من اليمين إلى اليسار)، ويستخدم التصميم العماني الأصيل بالألوان الرسمية للسلطنة، ويتضمن التحقق من صحة أرقام الهوية العمانية وأرقام الهواتف المحلية، ويراعي القوانين والأنظمة المحلية في إدارة العقارات.

التقنيات والأدوات المستخدمة

يعتمد المشروع على مجموعة متنوعة من التقنيات الحديثة والمجربة في تطوير تطبيقات الويب. في الخلفية (Backend)، يستخدم Python كلغة البرمجة الأساسية مع إطار العمل Django الذي يوفر بنية قوية ومرنة لتطوير التطبيقات. قاعدة البيانات المستخدمة هي SQLite للتطوير مع إمكانية التطوير إلى MySQL أو PostgreSQL للإنتاج.

لضمان Bootstrap 5 مع إطار العمل CSS3 و HTML5 يتم استخدام، (Frontend) في الواجهة الأمامية التصميم المتجاوب والحديث. JavaScript يُستخدم لإضافة التفاعلية والحركة، بينما تُستخدم أيقونات Bootstrap Icons لتحسين تجربة المستخدم البصرية.

متطلبات النظام والتقنيات المستخدمة

متطلبات النظام الأساسية

لتشغيل هذا المشروع بنجاح، يجب توفر مجموعة من المتطلبات الأساسية على النظام المستخدم. أولاً، يجب أن يكون نظام التشغيل متوافقاً مع Linux (Ubuntu 20.04 أو أحدث)، macOS 10.15 أو أحدث، و Windows 10 أو أحدث). هذه الأنظمة توفر البيئة المناسبة لتشغيل Python، وهذا يشمل أنظمة Python وجميع المكتبات المطلوبة.

ثانياً، يجب توفر Python 3.8 أو أحدث، حيث أن Django 4.x يتطلب هذا الإصدار كحد أدنى. يُنصح بشدة باستخدام Python 3.11 أو أحدث للحصول على أفضل أداء وأحدث المميزات. يمكن التحقق من إصدار Python المثبت باستخدام الأمر `python --version` في سطر الأوامر.

ثالثاً، يجب توفر مساحة تخزين كافية على القرص الصلب، حيث يحتاج المشروع إلى حوالي 500 ميجابايت للملفات الأساسية والمكتبات، بالإضافة إلى مساحة إضافية لقاعدة البيانات والملفات المرفوعة. يُنصح بتوفير 2 جيجابايت على الأقل لضمان عمل النظام بسلاسة.

المكتبات والحزم المطلوبة

يعتمد المشروع على مجموعة من المكتبات الأساسية التي يجب تثبيتها قبل البدء. Django 4.2 هو إطار العمل الأساسي الذي يوفر جميع الأدوات اللازمة لتطوير تطبيق الويب، بما في ذلك ORM لإدارة قاعدة البيانات، ونظام القوالب، ونظام التوجيه، وأدوات الأمان.

Pillow هي مكتبة معالجة الصور المطلوبة لدعم ImageField في Django، والتي تُستخدم لرفع وعرض صور المباني والمستأجرين. هذه المكتبة توفر إمكانيات متقدمة لمعالجة الصور مثل تغيير الحجم والقص والتحويل بين التنسيقات المختلفة.

Django-extensions هي مجموعة من الأدوات الإضافية التي تسهل عملية التطوير، مثل أوامر إدارة إضافية وأدوات تصحيح الأخطاء. Django-debug-toolbar أداة مفيدة جداً أثناء التطوير لمراقبة أداء الاستعلامات وتحليل الأخطاء.

أدوات التطوير المُوصى بها

لتحسين تجربة التطوير وزيادة الإنتاجية، يُنصح باستخدام مجموعة من الأدوات المتخصصة. Visual Studio Code هو محرر النصوص المُوصى به لهذا المشروع، حيث يوفر دعماً ممتازاً لـ Python و Django مع إضافات مفيدة مثل Python extension و Django extension.

Git هو نظام إدارة الإصدارات الذي يجب استخدامه لتتبع التغييرات في الكود وإدارة الإصدارات المختلفة. يُنصح بإنشاء مستودع Git منذ بداية المشروع وعمل commits منتظمة لحفظ التقدم.

SQLite يُنصح بهما كقواعد بيانات للإنتاج، رغم أن MySQL أو PostgreSQL تكفي للتطوير والتعلم. هذه القواعد توفر أداءً أفضل وميزات متقدمة للتطبيقات الكبيرة.

إعداد البيئة الافتراضية

إنشاء بيئة افتراضية منفصلة للمشروع خطوة أساسية لضمان عدم تداخل المكتبات مع مشاريع أخرى. يتم إنشاء البيئة الافتراضية باستخدام الأمر `python -m venv venv`، والذي ينشئ مجلداً جديداً يحتوي على نسخة منفصلة من Python وجميع المكتبات.

تفعيل البيئة الافتراضية يختلف حسب نظام التشغيل. في أنظمة macOS و Linux، يتم استخدام الأمر `source venv/bin/activate`، بينما في Windows يُستخدم `venv\Scripts\activate`. عند تفعيل البيئة بنجاح، سيظهر اسم البيئة في بداية سطر الأوامر.

بعد تفعيل البيئة الافتراضية، يمكن تثبيت جميع المكتبات المطلوبة باستخدام pip. يُنصح بإنشاء ملف requirements.txt يحتوي على قائمة بجميع المكتبات وإصداراتها، مما يسهل إعادة إنشاء البيئة على أجهزة أخرى.

هيكل المشروع والتصميم

فلسفة التصميم المعماري

يتبع هذا المشروع نمط MVC (Model-View-Controller) بشكل طبيعي، والذي Django الذي يطبقه MVT (Model-View-Template). هذا النمط يفصل بين منطق البيانات (Models) ومنطق العرض (Views) وطبقة العرض (Templates)، مما يجعل الكود أكثر تنظيماً وقابلية للصيانة. Django يُعرف في

تمثل هيكل البيانات وتتعامل مع قاعدة البيانات مباشرة. في مشروعنا، لدينا نماذج (Models) للمباني والمستأجرين والعقود والدفعات وطلبات الصيانة. كل نموذج يحتوي على الحقول اللازمة والعلاقات مع النماذج الأخرى، بالإضافة إلى الدوال المساعدة والتحقق من صحة البيانات.

تحتوي على منطق التطبيق وتتعامل مع طلبات المستخدمين. تستقبل البيانات من (Views) العروض النماذج وتمررها إلى القوالب للعرض. في مشروعنا، لدينا عروض لعرض قوائم البيانات وتفاصيل العناصر وإضافة وتعديل البيانات.

القوالب (Templates) تحتوي على كود HTML مع Django template language لعرض البيانات للمستخدم. تتضمن القوالب التصميم والتخطيط والتفاعلات الأساسية، وتدعم الترجمة والتدويل بشكل كامل.

تنظيم التطبيقات

المشروع مقسم إلى عدة تطبيقات منفصلة، كل منها يركز على جانب معين من النظام. هذا التقسيم يجعل الكود أكثر تنظيماً ويسهل الصيانة والتطوير المستقبلي.

تطبيق `core` هو التطبيق الأساسي الذي يحتوي على النموذج المخصص للمستخدمين والصفحة الرئيسية والإعدادات المشتركة. هذا التطبيق يعمل كنقطة مركزية للمشروع ويحتوي على العناصر التي تُستخدم عبر التطبيقات الأخرى.

تطبيق `buildings` يدير كل ما يتعلق بالمباني والطوابق والوحدات. يحتوي على نماذج Building و Floor و Unit و UnitType، بالإضافة إلى العروض والقوالب اللازمة لإدارة هذه البيانات. هذا التطبيق يوفر واجهات لإضافة وتعديل وعرض المباني وتفاصيلها.

تطبيق `tenants` يدير معلومات المستأجرين ووثائقهم. يحتوي على نموذج Tenant و TenantDocument، مع دعم كامل لرفع وإدارة الوثائق المختلفة. هذا التطبيق يتضمن أيضاً التحقق من صحة أرقام الهوية العمانية وأرقام الهواتف.

تطبيق `leases` يدير عقود الإيجار وتجديدها وإنهائها. يحتوي على نماذج Lease و LeaseRenewal و LeaseTermination، مع منطق معقد لحساب المبالغ والتواريخ والتنبيهات.

تطبيق `payments` يدير الدفعات والإيصالات والجدولة. يحتوي على نماذج Payment و Receipt، و PaymentSchedule مع إمكانيات متقدمة لتتبع الدفعات المتأخرة وإنشاء التقارير المالية.

تطبيق `maintenance` يدير طلبات الصيانة ومقدمي الخدمة. يحتوي على نماذج ServiceProvider، و MaintenanceRequest مع نظام لتتبع حالة الطلبات وإدارة التكاليف.

تطبيق `reports` يوفر التقارير والإحصائيات المختلفة. يحتوي على عروض متخصصة لإنشاء تقارير مالية وإدارية مع إمكانية التصدير إلى PDF و Excel.

قاعدة البيانات والعلاقات

تصميم قاعدة البيانات يتبع أفضل الممارسات في التطبيع والعلاقات. الجداول الرئيسية مترابطة بعلاقات واضحة ومنطقية تضمن سلامة البيانات وتجنب التكرار.

جدول المباني (Buildings) هو نقطة البداية، حيث يحتوي على معلومات أساسية مثل الاسم والعنوان والوصف وعدد الطوابق والمرافق المتاحة. كل مبنى يرتبط بعدة طوابق من خلال علاقة One-to-Many.

جدول الطوابق (Floors) يحتوي على معلومات كل طابق مثل الرقم والوصف ونوع الطابق. كل طابق يرتبط بعدة وحدات، وهذا التصميم يوفر مرونة في إدارة المباني المعقدة.

جدول الوحدات (Units) يحتوي على تفاصيل كل وحدة مثل الرقم والمساحة والسعر والحالة ونوع الوحدة. الوحدات ترتبط بالمستأجرين من خلال عقود الإيجار، مما يسمح بتتبع تاريخ الإيجار لكل وحدة.

جدول المستأجرين (Tenants) يحتوي على المعلومات الشخصية الكاملة مع دعم للوثائق المرفقة. التصميم يراعي الخصوصية والأمان، مع تشفير البيانات الحساسة وإمكانية التحكم في الوصول.

جدول العقود (Leases) يربط بين المستأجرين والوحدات مع تفاصيل العقد مثل تاريخ البداية والنهاية والمبلغ والشروط. هذا الجدول يدعم تجديد العقود وتتبع التغييرات.

جدول الدفعات (Payments) يتتبع جميع المعاملات المالية مع ربطها بالعقود والمستأجرين. التصميم يدعم أنواع دفع متعددة ويوفر تتبعاً دقيقاً للحالة المالية.

أمان البيانات والخصوصية

الأمان جانب أساسي في تصميم النظام، خاصة مع التعامل مع بيانات شخصية حساسة. يتم تطبيق عدة طبقات من الحماية لضمان سلامة البيانات وخصوصية المستخدمين.

يتم استخدام نظام المصادقة والتفويض المدمج مع تخصيصات إضافية. كل Django، على مستوى عملية تتطلب تسجيل دخول، ويتم التحقق من صلاحيات المستخدم قبل السماح بالوصول إلى البيانات أو تعديلها.

على مستوى قاعدة البيانات، يتم تشفير البيانات الحساسة مثل أرقام الهوية وتفاصيل الحسابات البنكية. كما يتم تطبيق قيود على مستوى قاعدة البيانات لضمان سلامة البيانات ومنع التلاعب.

على مستوى التطبيق، يتم التحقق من صحة جميع البيانات المدخلة وتنظيفها لمنع هجمات الحقن. كما يتم تطبيق XSS protection و CSRF protection لحماية المستخدمين من الهجمات الشائعة.

يتم أيضاً تطبيق مبدأ الحد الأدنى من الصلاحيات، حيث يحصل كل مستخدم على الصلاحيات اللازمة فقط لأداء مهامه. هذا يقلل من مخاطر تسريب البيانات أو سوء الاستخدام.

إعداد البيئة التطويرية

إنشاء المشروع من الصفر

تتطلب إعداداً دقيقاً للبيئة التطويرية. الخطوة الأولى هي إنشاء Django البداية الصحيحة لأي مشروع مجلد للمشروع في مكان مناسب على النظام، ويُفضل أن يكون في مجلد منفصل للمشاريع البرمجية. حيث يعكس طبيعة ، `oman_rental_management` اختيار اسم واضح ومعبر للمجلد مهم، مثل المشروع وهدفه.

بعد إنشاء المجلد، الخطوة التالية هي إنشاء البيئة الافتراضية داخل هذا المجلد. البيئة الافتراضية تضمن عزل مكتبات المشروع عن النظام الأساسي، مما يمنع تضارب الإصدارات ويسهل إدارة التبعيات. يتم والذي ينشئ مجلداً فرعياً يحتوي على نسخة ، `python -m venv venv` إنشاء البيئة باستخدام الأمر منفصلة من Python.

Unix تفعيل البيئة الافتراضية خطوة أساسية يجب تكرارها في كل جلسة عمل. في أنظمة (Linux/macOS)، يتم التفعيل باستخدام `source venv/bin/activate` ، بينما في Windows يُستخدم `venv\Scripts\activate` . عند نجاح التفعيل، سيظهر اسم البيئة `(venv)` في بداية سطر الأوامر، مما يؤكد أن جميع العمليات ستتم داخل البيئة المعزولة.

تثبيت Django والمكتبات الأساسية

بعد تفعيل البيئة الافتراضية، يأتي دور تثبيت Django والمكتبات الأساسية. يُنصح بتثبيت أحدث إصدار مستقر من Django، والذي يمكن تثبيته باستخدام `pip install django` . هذا الأمر سيقوم بتحميل وتثبيت Django مع جميع تبعياته الأساسية.

Pillow هي المكتبة التالية المطلوبة، وهي ضرورية لدعم ImageField في Django. يتم تثبيتها باستخدام `pip install Pillow` . هذه المكتبة توفر إمكانيات معالجة الصور المتقدمة وتدعم تنسيقات متعددة مثل GIF. و PNG و JPEG.

للتطوير، يُنصح بتثبيت مكتبات إضافية تسهل عملية التطوير وتحسن الإنتاجية. `django-extensions` تساعد في تحليل الأداء وتصحيح الأخطاء `django-debug-toolbar` توفر أوامر إدارة إضافية مفيدة، و أثناء التطوير.

خطوة مهمة لتوثيق جميع المكتبات المستخدمة وإصداراتها. يمكن إنشاء ملف `requirements.txt` والذي سيحفظ قائمة بجميع ، `pip freeze > requirements.txt` إنشاء هذا الملف باستخدام المكتبات المثبتة. هذا الملف يسهل إعادة إنشاء البيئة على أجهزة أخرى باستخدام `pip install -r requirements.txt` .

إنشاء مشروع Django الأساسي

إنشاء مشروع Django الجديد يتم باستخدام الأمر `django-admin startproject oman_rental_system .` بإنشاء المشروع في Django النقطة في نهاية الأمر مهمة، حيث تخبر المجلد الحالي بدلاً من إنشاء مجلد فرعي جديد. هذا يحافظ على تنظيم أفضل لهيكل المشروع.

في المجلد الرئيسي، وهو الأداة الأساسية لإدارة `manage.py` بعد إنشاء المشروع، ستجد ملف `settings.py` المشروع. كما ستجد مجلداً يحمل اسم المشروع يحتوي على الإعدادات الأساسية مثل `wsgi.py` و `urls.py` و.

الخطوة التالية هي إنشاء التطبيقات المختلفة للمشروع. كل تطبيق يركز على جانب معين من النظام، مما يجعل الكود أكثر تنظيماً وقابلية للصيانة. يتم إنشاء التطبيقات باستخدام `python manage.py startapp app_name`.

للمشروع الحالي، نحتاج إلى إنشاء عدة تطبيقات: `core` للوظائف الأساسية، `buildings` لإدارة المباني، `tenants` لإدارة المستأجرين، `leases` لإدارة العقود، `payments` لإدارة الدفعات، `reports` للتقارير. `maintenance` لإدارة الصيانة، و.

تكوين الإعدادات الأساسية

ملف `settings.py` هو قلب مشروع Django، أول حيث يحتوي على جميع الإعدادات والتكوينات. تعديل مطلوب هو إضافة التطبيقات المُنشأة إلى قائمة `INSTALLED_APPS`. هذا يخبر Django بوجود هذه التطبيقات ويمكنه من استخدام نماذجها وعروضها.

إعداد قاعدة البيانات خطوة مهمة، حيث يمكن استخدام SQLite أو تكوين (الإعداد الافتراضي) للتطوير مناسبة للتطوير والتعلم لأنها لا تتطلب إعداد SQLite. للإنتاج PostgreSQL قاعدة بيانات أخرى مثل خادم منفصل.

إعداد المنطقة الزمنية واللغة مهم للمشاريع المحلية. يجب تغيير `TIME_ZONE` إلى `'Asia/Muscat'` لدعم اللغة العربية. كما يجب التأكد من أن `USE_TZ` و `USE_I18N` إلى `'ar'` و `LANGUAGE_CODE` مفعلان لدعم الترجمة والمناطق الزمنية.

إعداد الملفات الثابتة والوسائط ضروري لعرض JavaScript و CSS والصور. يجب تحديد `STATIC_URL` للملفات الثابتة، و `MEDIA_URL` للملفات المرفوعة من `MEDIA_ROOT` و `STATIC_ROOT` المستخدمين.

إعداد نظام الترجمة

أولاً، يجب إضافة دعم اللغة العربية يتطلب إعدادات خاصة في Django. في الإعدادات. هذا `MIDDLEWARE` إلى قائمة `django.middleware.locale.LocaleMiddleware` الـ middleware يتعامل مع تحديد لغة المستخدم وتطبيق الترجمة المناسبة.

في جذر المشروع ضروري لحفظ ملفات الترجمة. يجب إضافة مسار هذا المجلد `locale` إنشاء مجلد أين يبحث عن ملفات الترجمة Django هذا يخبر . `settings.py` في `LOCALE_PATHS` إلى إعداد.

للمشروع الحالي، . `settings.py` في `LANGUAGES` تكوين اللغات المدعومة يتم من خلال إعداد نحتاج إلى دعم العربية والإنجليزية، لذا يجب تحديد هاتين اللغتين مع أكوادهما الصحيحة.

للعربية و `python manage.py makemessages -l ar` إنشاء ملفات الترجمة يتم باستخدام أمر للإنجليزية. هذه الأوامر تبحث في الكود عن النصوص `python manage.py makemessages -l en` القابلة للترجمة وتنشئ ملفات `.po` يمكن تعديلها لإضافة الترجمات.

إعداد قاعدة البيانات الأولية

بعد تكوين الإعدادات الأساسية، يجب إنشاء قاعدة البيانات وتطبيق الهجرات الأولية. Django يأتي مع مجموعة من النماذج المدمجة للمصادقة والجلسات والإدارة، والتي تحتاج إلى جداول في قاعدة البيانات.

الأمر `python manage.py migrate` ينشئ جميع الجداول المطلوبة للنماذج المدمجة. هذا الأمر يجب تشغيله في كل مرة يتم فيها تعديل النماذج أو إضافة تطبيقات جديدة.

`python manage.py createsuperuser` إنشاء مستخدم إداري خطوة مهمة للوصول إلى واجهة الإدارة. يتم ذلك باستخدام والذي سيطلب اسم المستخدم والبريد الإلكتروني وكلمة المرور. هذا المستخدم سيكون له صلاحيات كاملة على النظام.

والذي يبدأ خادم التطوير ، `python manage.py runserver` اختبار التشغيل الأولي يتم باستخدام للتأكد من أن كل `http://localhost:8000` على المنفذ 8000. يمكن الوصول إلى الموقع من خلال شيء يعمل بشكل صحيح.

إعداد نظام إدارة الإصدارات

في بداية Git لإدارة الإصدارات أمر ضروري لأي مشروع برمجي. يجب إنشاء مستودع Git استخدام الذي يحتوي على تاريخ المشروع وجميع `.git` هذا ينشئ مجلد . `git init` المشروع باستخدام التغييرات.

مهم لتجنب رفع ملفات غير ضرورية إلى المستودع. هذا الملف يجب أن `.gitignore` إنشاء ملف المؤقتة Python وملفات (`*.sqlite3`)، وملفات قاعدة البيانات (`venv/`) يتضمن البيئة الافتراضية (`__pycache__/`, `*.pyc`)، وملفات الوسائط المرفوعة (`media/`).

أولي للمشروع يحفظ الحالة الأولية ويوفر نقطة مرجعية للعودة إليها عند الحاجة. يتم commit عمل ذلك باستخدام `git add .` لإضافة جميع الملفات، ثم `git commit -m "Initial commit"` لحفظ التغييرات.

إنشاء فروع للميزات المختلفة يساعد في تنظيم العمل وتجنب التضارب. يمكن إنشاء فرع جديد والعمل عليه بشكل منفصل قبل دمجه مع الفرع باستخدام `git checkout -b feature-name` الرئيسي.

تطوير النماذج (Models)

فهم نماذج Django والـ ORM

نماذج Django هي الطبقة التي تتعامل مع قاعدة البيانات وتمثل هيكل البيانات في التطبيق. كل نموذج وتحتوي على حقول تمثل أعمدة الجدول في `django.db.models.Model` ترث من Python هو فئة إلى Python يترجم عمليات (Object-Relational Mapping) Django ORM. قاعدة البيانات بدلاً من Python مما يسمح للمطورين بالتعامل مع قاعدة البيانات باستخدام كود SQL، استعلامات SQL المباشر.

أهمية النماذج تكمن في كونها تحدد بنية البيانات وقواعد العمل والعلاقات بين الجداول المختلفة. كل حقل في النموذج له نوع محدد مثل CharField للنصوص، IntegerField للأرقام الصحيحة، DateField للتواريخ، و ForeignKey للعلاقات. هذه الأنواع تحدد كيفية تخزين البيانات والتحقق من صحتها.

العلاقات بين النماذج تُعرّف باستخدام حقول خاصة مثل ForeignKey للعلاقة واحد-إلى-كثير، OneToOneField للعلاقة واحد-إلى-واحد. هذه العلاقات تسمح بربط البيانات بطريقة منطقية وتسهل الاستعلامات المعقدة. ManyToManyField للعلاقة كثير-إلى-كثير، و

نموذج المستخدم المخصص

في مشروعنا، نبدأ بإنشاء نموذج مستخدم مخصص في تطبيق `core`. هذا النموذج يوسع نموذج المستخدم الافتراضي في Django ليشمل حقولاً إضافية مناسبة للسياق العماني. النموذج المخصص يرث من `AbstractUser` ويضيف حقولاً مثل رقم الهاتف والصورة الشخصية والمنطقة.

```python
 from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.translation import gettext_lazy as _

class User(AbstractUser):
    phone_number = models.CharField(
        _('رقم الهاتف'),
        max_length=15,
        blank=True,
        help_text=_('رقم الهاتف مع رمز البلد')
    )

    profile_picture = models.ImageField(
        _('الصورة الشخصية'),
        upload_to='profiles/',
        blank=True,
        null=True
    )

    region = models.CharField(
        _('المنطقة'),
        max_length=100,
        blank=True,
        help_text=_('المنطقة أو المحافظة')
    )
```

مهم لدعم الترجمة، حيث يؤجل ترجمة النصوص حتى وقت العرض. هذا `gettext_lazy` استخدام `verbose_name` يضمن أن النصوص ستُترجم حسب لغة المستخدم الحالية. كل حقل يحتوي على `help_text` باللغة العربية و لتوضيح الغرض من الحقل.

نماذج المباني والوحدات

`Building` نماذج المباني تشكل العمود الفقري للنظام، حيث تمثل الهيكل الفيزيائي للعقارات. نموذج يحتوي على المعلومات الأساسية للمبنى مثل الاسم والعنوان والوصف وعدد الطوابق والمرافق المتاحة.

```python
class Building(models.Model):
    name = models.CharField(
        _('اسم المبنى'),
        max_length=200,
        help_text=_('اسم المبنى التجاري')
    )

    address = models.TextField(
        _('العنوان'),
        help_text=_('العنوان الكامل للمبنى')
    )

    total_floors = models.PositiveIntegerField(
        _('إجمالي الطوابق'),
        default=1,
        help_text=_('العدد الإجمالي للطوابق في المبنى')
    )

    has_mosque = models.BooleanField(
        _('يحتوي على مصلى'),
        default=False,
        help_text=_('هل يحتوي المبنى على مصلى؟')
    )

    has_elevator = models.BooleanField(
        _('يحتوي على مصعد'),
        default=False,
        help_text=_('هل يحتوي المبنى على مصعد؟')
    )

    has_parking = models.BooleanField(
        _('يحتوي على مواقف'),
        default=False,
        help_text=_('هل يحتوي المبنى على مواقف سيارات؟')
    )
```

نموذج `Floor` يمثل الطوابق داخل المبنى ويرتبط بالمبنى من خلال ForeignKey. هذا التصميم يسمح بمرونة في إدارة المباني المعقدة التي قد تحتوي على طوابق بخصائص مختلفة.

```
class Floor(models.Model):
    FLOOR_TYPES = [
        ('ground', _('الطابق الأرضي')),
        ('first', _('الطابق الأول')),
        ('second', _('الطابق الثاني')),
        ('basement', _('الطابق السفلى')),
        ('roof', _('السطح')),
    ]

    building = models.ForeignKey(
        Building,
        on_delete=models.CASCADE,
        related_name='floors',
        verbose_name=_('المبنى')
    )

    floor_number = models.IntegerField(
        _('رقم الطابق'),
        help_text=_('رقم الطابق في المبنى')
    )

    floor_type = models.CharField(
        _('نوع الطابق'),
        max_length=20,
        choices=FLOOR_TYPES,
        default='ground'
    )
```

نموذج `Unit` يمثل الوحدات الفردية داخل كل طابق. هذا النموذج يحتوي على تفاصيل مهمة مثل المساحة والسعر والحالة ونوع الوحدة.

```python
class Unit(models.Model):
    STATUS_CHOICES = [
        ('vacant', _('شاغرة')),
        ('rented', _('مؤجرة')),
        ('maintenance', _('تحت الصيانة')),
        ('reserved', _('محجوزة')),
    ]

    floor = models.ForeignKey(
        Floor,
        on_delete=models.CASCADE,
        related_name='units',
        verbose_name=_('الطابق')
    )

    unit_number = models.CharField(
        _('رقم الوحدة'),
        max_length=10,
        help_text=_('رقم الوحدة في الطابق')
    )

    area = models.DecimalField(
        _('المساحة'),
        max_digits=8,
        decimal_places=2,
        help_text=_('المساحة بالمتر المربع')
    )

    rent_price = models.DecimalField(
        _('سعر الإيجار'),
        max_digits=10,
        decimal_places=3,
        help_text=_('سعر الإيجار الشهري بالريال العماني')
    )

    status = models.CharField(
        _('الحالة'),
        max_length=20,
        choices=STATUS_CHOICES,
        default='vacant'
    )
```

نماذج المستأجرين والوثائق

نموذج `Tenant` يحتوي على المعلومات الشخصية الكاملة للمستأجرين مع مراعاة الخصوصية والقوانين المحلية. النموذج يتضمن التحقق من صحة البيانات مثل رقم الهوية العمانية ورقم الهاتف.

```python
class Tenant(models.Model):
    GENDER_CHOICES = [
        ('male', _('ذكر')),
        ('female', _('أنثى')),
    ]

    MARITAL_STATUS_CHOICES = [
        ('single', _('أعزب')),
        ('married', _('متزوج')),
        ('divorced', _('مطلق')),
        ('widowed', _('أرمل')),
    ]

    first_name = models.CharField(
        _('الاسم الأول'),
        max_length=50
    )

    last_name = models.CharField(
        _('اسم العائلة'),
        max_length=50
    )

    civil_id = models.CharField(
        _('رقم الهوية المدنية'),
        max_length=8,
        unique=True,
        help_text=_('(8 أرقام) رقم الهوية المدنية العمانية')
    )

    phone_number = models.CharField(
        _('رقم الهاتف'),
        max_length=15,
        help_text=_('رقم الهاتف العماني')
    )

    email = models.EmailField(
        _('البريد الإلكتروني'),
        blank=True,
        null=True
    )

    date_of_birth = models.DateField(
        _('تاريخ الميلاد')
    )

    gender = models.CharField(
        _('الجنس'),
        max_length=10,
        choices=GENDER_CHOICES
    )

    marital_status = models.CharField(
        _('الحالة الاجتماعية'),
        max_length=20,
        choices=MARITAL_STATUS_CHOICES
    )

    nationality = models.CharField(
        _('الجنسية'),
        max_length=50,
```

```
        default='عماني'
    )
```

يدير الوثائق المرفقة لكل مستأجر مثل نسخة الهوية وعقد العمل وكشف `TenantDocument` نموذج الراتب. هذا النموذج يدعم أنواع وثائق متعددة ويتتبع تاريخ الرفع وحالة التحقق.

```
class TenantDocument(models.Model):
    DOCUMENT_TYPES = [
        ('civil_id', _('نسخة الهوية المدنية')),
        ('passport', _('نسخة جواز السفر')),
        ('work_contract', _('عقد العمل')),
        ('salary_certificate', _('شهادة راتب')),
        ('bank_statement', _('كشف حساب بنكي')),
        ('other', _('أخرى')),
    ]

    tenant = models.ForeignKey(
        Tenant,
        on_delete=models.CASCADE,
        related_name='documents',
        verbose_name=_('المستأجر')
    )

    document_type = models.CharField(
        _('نوع الوثيقة'),
        max_length=20,
        choices=DOCUMENT_TYPES
    )

    document_file = models.FileField(
        _('ملف الوثيقة'),
        upload_to='tenant_documents/'
    )

    uploaded_at = models.DateTimeField(
        _('تاريخ الرفع'),
        auto_now_add=True
    )

    is_verified = models.BooleanField(
        _('تم التحقق'),
        default=False
    )
```

نماذج العقود والدفعات

يمثل عقود الإيجار ويربط بين المستأجرين والوحدات. هذا النموذج يحتوي على جميع `Lease` نموذج تفاصيل العقد مثل تواريخ البداية والنهاية والمبلغ والشروط الخاصة.

```python
class Lease(models.Model):
    STATUS_CHOICES = [
        ('active', _('نشط')),
        ('expired', _('منتهى')),
        ('terminated', _('مُنهى')),
        ('renewed', _('مُجدد')),
    ]

    tenant = models.ForeignKey(
        Tenant,
        on_delete=models.CASCADE,
        related_name='leases',
        verbose_name=_('المستأجر')
    )

    unit = models.ForeignKey(
        Unit,
        on_delete=models.CASCADE,
        related_name='leases',
        verbose_name=_('الوحدة')
    )

    start_date = models.DateField(
        _('تاريخ بداية العقد')
    )

    end_date = models.DateField(
        _('تاريخ نهاية العقد')
    )

    monthly_rent = models.DecimalField(
        _('الإيجار الشهرى'),
        max_digits=10,
        decimal_places=3,
        help_text=_('المبلغ الشهري بالريال العماني')
    )

    security_deposit = models.DecimalField(
        _('التأمين'),
        max_digits=10,
        decimal_places=3,
        help_text=_('مبلغ التأمين بالريال العماني')
    )

    status = models.CharField(
        _('حالة العقد'),
        max_length=20,
        choices=STATUS_CHOICES,
        default='active'
    )
```

نموذج `Payment` يتتبع جميع الدفعات المرتبطة بالعقود. هذا النموذج يدعم أنواع دفع متعددة ويوفر تتبعاً دقيقاً للحالة المالية لكل مستأجر.

```python
class Payment(models.Model):
    PAYMENT_TYPES = [
        ('rent', _('إيجار')),
        ('deposit', _('تأمين')),
        ('maintenance', _('صيانة')),
        ('utilities', _('خدمات')),
        ('penalty', _('غرامة')),
    ]

    PAYMENT_METHODS = [
        ('cash', _('نقداً')),
        ('bank_transfer', _('تحويل بنكي')),
        ('check', _('شيك')),
        ('card', _('بطاقة')),
    ]

    STATUS_CHOICES = [
        ('pending', _('معلق')),
        ('completed', _('مكتمل')),
        ('failed', _('فاشل')),
        ('refunded', _('مُسترد')),
    ]

    lease = models.ForeignKey(
        Lease,
        on_delete=models.CASCADE,
        related_name='payments',
        verbose_name=_('العقد')
    )

    amount = models.DecimalField(
        _('المبلغ'),
        max_digits=10,
        decimal_places=3
    )

    payment_type = models.CharField(
        _('نوع الدفعة'),
        max_length=20,
        choices=PAYMENT_TYPES
    )

    payment_method = models.CharField(
        _('طريقة الدفع'),
        max_length=20,
        choices=PAYMENT_METHODS
    )

    payment_date = models.DateField(
        _('تاريخ الدفع')
    )

    status = models.CharField(
        _('الحالة'),
        max_length=20,
        choices=STATUS_CHOICES,
        default='pending'
    )
```

التحقق من صحة البيانات والقيود

يوفر Django نظاماً قوياً للتحقق من صحة البيانات على مستوى النموذج. يمكن إضافة دوال تحقق مخصصة لضمان سلامة البيانات وتطبيق القواعد التجارية المعقدة.

للتحقق من صحة رقم الهوية العمانية، يمكن إضافة دالة تحقق في نموذج المستأجر:

```
from django.core.exceptions import ValidationError
import re

def validate_omani_civil_id(value):
    """التحقق من صحة رقم الهوية العمانية"""
    if not re.match(r'^\d{8}$', value):
        raise ValidationError(
            _('رقم الهوية العمانية يجب أن يكون 8 أرقام')
        )

def validate_omani_phone(value):
    """التحقق من صحة رقم الهاتف العمانى"""
    if not re.match(r'^(968)?\d{8}$', value.replace('+', '').replace(' ', '')):
        raise ValidationError(
            _('رقم الهاتف غير صحيح')
        )
```

يمكن أيضاً إضافة دوال `clean()` مخصصة في النماذج للتحقق من القواعد المعقدة التي تتضمن عدة حقول:

```
def clean(self):
    cleaned_data = super().clean()
    start_date = cleaned_data.get('start_date')
    end_date = cleaned_data.get('end_date')

    if start_date and end_date:
        if start_date >= end_date:
            raise ValidationError(
                _('تاريخ نهاية العقد يجب أن يكون بعد تاريخ البداية')
            )

    return cleaned_data
```

الفهرسة وتحسين الأداء

يوفر Django لتحسين أداء الاستعلامات، يجب إضافة فهارس على الحقول التي يتم البحث فيها بكثرة. عدة خيارات للفهرسة من خلال Meta class:

```
class Meta:
    verbose_name = _('مستأجر')
    verbose_name_plural = _('المستأجرون')
    ordering = ['first_name', 'last_name']
    indexes = [
        models.Index(fields=['civil_id']),
        models.Index(fields=['phone_number']),
        models.Index(fields=['last_name', 'first_name']),
    ]
    constraints = [
        models.UniqueConstraint(
            fields=['civil_id'],
            name='unique_civil_id'
        ),
    ]
```

في الاستعلامات يقلل من عدد `prefetch_related()` و `select_related()` استخدام استعلامات قاعدة البيانات ويحسن الأداء بشكل كبير، خاصة عند التعامل مع العلاقات المعقدة بين النماذج.

نظام الترجمة والتدويل

مفهوم التدويل في Django

التدويل (Internationalization) أو i18n هو عملية تصميم التطبيق ليدعم لغات وثقافات متعددة دون الحاجة لتغييرات في الكود الأساسي. Django يوفر نظاماً متكاملاً للتدويل يشمل ترجمة النصوص، وتنسيق التواريخ والأرقام، واتجاه النص (RTL/LTR). هذا النظام ضروري جداً للمشاريع التي تستهدف أسواقاً متعددة اللغات مثل السوق العماني الذي يتطلب دعم اللغتين العربية والإنجليزية.

أهمية التدويل تكمن في توفير تجربة مستخدم محلية وطبيعية لكل مستخدم حسب لغته وثقافته. في السياق العماني، هذا يعني دعم اللغة العربية كلغة أساسية مع الحفاظ على الإنجليزية كلغة ثانوية، ودعم اتجاه النص من اليمين إلى اليسار، وتنسيق التواريخ والأرقام حسب المعايير المحلية.

Django يستخدم مكتبة GNU gettext لترجمة الصناعي المعيار تعتبر والتي ترجمات، لإدارة التطبيقات. هذا النظام يفصل النصوص القابلة للترجمة عن الكود، مما يسمح للمترجمين بالعمل على الترجمات دون الحاجة لفهم البرمجة.

إعداد الترجمة في الإعدادات

الخطوة الأولى في إعداد نظام الترجمة هي تكوين الإعدادات المناسبة في ملف `settings.py`. يجب لتفعيل التوطين `USE_L10N = True` و لتفعيل نظام الترجمة، `USE_I18N = True` التأكد من أن (localization) الذي يتعامل مع تنسيق التواريخ والأرقام.

```
# إعدادات الترجمة والتدويل
USE_I18N = True
USE_L10N = True
USE_TZ = True

LANGUAGE_CODE = 'ar'  # اللغة الافتراضية
TIME_ZONE = 'Asia/Muscat'  # المنطقة الزمنية العمانية

# اللغات المدعومة
LANGUAGES = [
    ('ar', _('العربية')),
    ('en', _('English')),
]

# مسارات ملفات الترجمة
LOCALE_PATHS = [
    BASE_DIR / 'locale',
]
```

إضافة `LocaleMiddleware` إلى قائمة MIDDLEWARE. هذا الـ middleware يحدد لغة المستخدم من عدة مصادر مثل URL أو session أو cookies أو Accept-Language header.

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',  # مهم: يجب أن يكون بعد SessionMiddleware
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

ترجمة النصوص في النماذج والعروض

استخدام النسخة `_` (المختصر) لـ `gettext_lazy` في النماذج، يتم استخدام "lazy" لترجمة النصوص. مهم لأنها تؤجل الترجمة حتى وقت العرض، مما يضمن أن النص سيُترجم حسب لغة المستخدم الحالية وليس لغة الخادم.

```python
from django.utils.translation import gettext_lazy as _

class Building(models.Model):
    name = models.CharField(
        verbose_name=_('اسم المبنى'),
        max_length=200,
        help_text=_('اسم المبنى التجاري')
    )

    class Meta:
        verbose_name = _('مبنى')
        verbose_name_plural = _('المباني')
```

في العروض، يمكن استخدام `gettext` العادي للنصوص التي تُترجم في وقت التنفيذ:

```python
from django.utils.translation import gettext as _
from django.contrib import messages

def building_create(request):
    if request.method == 'POST':
        # معالجة النموذج
        messages.success(request, _('تم إنشاء المبنى بنجاح'))

    return render(request, 'buildings/create.html')
```

ترجمة القوالب

في القوالب، يتم استخدام template tags خاصة للترجمة. أهم هذه العلامات هي `{% trans %}` للنصوص القصيرة و `{% blocktrans %}` للنصوص الطويلة التي تحتوي على متغيرات.

```
{% load i18n %}

<h1>{% trans "قائمة المباني" %}</h1>

<p>{% blocktrans count counter=buildings.count %}
    يوجد مبنى واحد في النظام.
{% plural %}
    يوجد {{ counter }} مبنى في النظام.
{% endblocktrans %}</p>

<!-- للنصوص مع متغيرات -->
{% blocktrans with name=building.name %}
    مرحباً بك في مبنى {{ name }}
{% endblocktrans %}
```

إنشاء وإدارة ملفات الترجمة

بعد إضافة النصوص القابلة للترجمة في الكود، يجب إنشاء ملفات الترجمة باستخدام أمر `makemessages`. هذا الأمر يبحث في جميع ملفات المشروع عن النصوص المحددة للترجمة وينشئ

ملفات `.po` لكل لغة.

```
# إنشاء ملفات الترجمة للعربية
python manage.py makemessages -l ar

# إنشاء ملفات الترجمة للإنجليزية
python manage.py makemessages -l en

# تحديث ملفات الترجمة الموجودة
python manage.py makemessages -a
```

ملفات `.po` تحتوي على النصوص الأصلية والترجمات المقابلة لها. كل إدخال يحتوي على `msgid` (النص الأصلي) و `msgstr` (الترجمة):

```
#: buildings/models.py:15
msgid "اسم المبنى"
msgstr "Building Name"

#: buildings/models.py:16
msgid "اسم المبنى التجاري"
msgstr "Commercial building name"
```

بعد إضافة الترجمات، يجب تجميع ملفات `.po` إلى ملفات `.mo` باستخدام أمر `compilemessages`:

```
python manage.py compilemessages
```

دعم اتجاه النص RTL

يوفر Django دعماً مدمجاً لهذا من اليمين إلى اليسار (RTL) دعم اتجاه النص ضروري للغة العربية. خلال template tags ومتغيرات خاصة.

```
{% load i18n %}
<!DOCTYPE html>
<html lang="{% get_current_language as LANGUAGE_CODE %}{{ LANGUAGE_CODE }}"
      dir="{% if LANGUAGE_CODE == 'ar' %}rtl{% else %}ltr{% endif %}">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% trans "نظام إدارة الإيجارات" %}{% endblock %}
</title>

    <!-- CSS للغة العربية -->
    {% if LANGUAGE_CODE == 'ar' %}
        <link rel="stylesheet" href="{% static 'css/rtl.css' %}">
    {% else %}
        <link rel="stylesheet" href="{% static 'css/ltr.css' %}">
    {% endif %}
</head>
```

في CSS، يجب إنشاء قواعد خاصة لدعم RTL:

```
/* ملف rtl.css */
body {
    direction: rtl;
    text-align: right;
}

.navbar-brand {
    margin-right: 0;
    margin-left: 1rem;
}

.dropdown-menu {
    right: 0;
    left: auto;
}

/* عكس الهوامش والحشو */
.me-2 { margin-left: 0.5rem !important; margin-right: 0 !important; }
.ms-2 { margin-right: 0.5rem !important; margin-left: 0 !important; }
```

تبديل اللغات

توفير إمكانية تبديل اللغات للمستخدمين يحسن تجربة الاستخدام. يمكن إنشاء نموذج بسيط لتبديل اللغة:

```
 {% load i18n %}
<form action="{% url 'set_language' %}" method="post" class="d-inline">
    {% csrf_token %}
    <input name="next" type="hidden" value="{{ redirect_to }}">
    <select name="language" onchange="this.form.submit()" class="form-select
form-select-sm">
        {% get_current_language as LANGUAGE_CODE %}
        {% get_available_languages as LANGUAGES %}
        {% for lang_code, lang_name in LANGUAGES %}
            <option value="{{ lang_code }}"{% if lang_code == LANGUAGE_CODE %}
selected{% endif %}>
                {{ lang_name }}
            </option>
        {% endfor %}
    </select>
</form>
```

يجب إضافة URL pattern لمعالجة تبديل اللغة:

```python
# في urls.py الرئيسي
from django.conf.urls.i18n import i18n_patterns

urlpatterns = [
    path('i18n/', include('django.conf.urls.i18n')),
]

# إضافة i18n_patterns للمسارات التي تحتاج ترجمة
urlpatterns += i18n_patterns(
    path('admin/', admin.site.urls),
    path('', include('core.urls')),
    path('buildings/', include('buildings.urls')),
    # باقي المسارات...
)
```

تنسيق التواريخ والأرقام

يوفر Django تنسيقاً تلقائياً للتواريخ والأرقام حسب اللغة المحددة. يمكن تخصيص هذا التنسيق من خلال إعدادات FORMAT_MODULE_PATH:

```python
# في settings.py
FORMAT_MODULE_PATH = [
    'myproject.formats',
]
```

إنشاء ملفات تنسيق مخصصة للغة العربية:

```python
# في ملف formats/ar/formats.py
DATE_FORMAT = 'j F Y'  # ١٥ يناير ٢٠٢٤
TIME_FORMAT = 'H:i'    # ١٤:٣٠
DATETIME_FORMAT = 'j F Y H:i'
SHORT_DATE_FORMAT = 'j/n/Y'

# تنسيق الأرقام
USE_THOUSAND_SEPARATOR = True
THOUSAND_SEPARATOR = '،'
DECIMAL_SEPARATOR = '.'
```

اختبار نظام الترجمة

اختبار نظام الترجمة يتطلب التأكد من أن جميع النصوص تُترجم بشكل صحيح وأن التخطيط يعمل مع كلا الاتجاهين. يمكن إنشاء اختبارات آلية للتحقق من الترجمات:

```python
from django.test import TestCase
from django.utils.translation import activate
from django.urls import reverse

class TranslationTestCase(TestCase):
    def test_arabic_translation(self):
        activate('ar')
        response = self.client.get(reverse('buildings:list'))
        self.assertContains(response, 'قائمة المباني')

    def test_english_translation(self):
        activate('en')
        response = self.client.get(reverse('buildings:list'))
        self.assertContains(response, 'Buildings List')
```

تطوير العروض (Views)

فهم أنماط العروض في Django

العروض في Django هي الطبقة التي تتعامل مع منطق التطبيق وتربط بين النماذج والقوالب. Django يوفر عدة أنماط للعروض، بدءاً من Function-Based Views (FBV) البسيطة إلى Class-Based Views (CBV) المتقدمة. كل نمط له مميزاته واستخداماته المناسبة.

Function-Based Views هي الطريقة التقليدية والأبسط لكتابة العروض. هذه العروض عبارة عن دوال عادية تستقبل HttpRequest وترجع HttpResponse. هذا النمط مناسب للعروض البسيطة والمخصصة التي تحتاج منطق معقد.

Class-Based Views توفر طريقة أكثر تنظيماً وقابلية لإعادة الاستخدام. Django يأتي مع مجموعة من CBVs الجاهزة مثل ListView و DetailView و CreateView التي تغطي الاحتياجات الشائعة. هذه العروض تقلل من تكرار الكود وتوفر بنية موحدة.

في مشروعنا، نستخدم مزيجاً من النمطين حسب الحاجة. العروض البسيطة مثل عرض قائمة المباني تستخدم CBVs، بينما العروض المعقدة مثل إنشاء عقد إيجار جديد تستخدم FBVs للمرونة الإضافية.

عروض إدارة المباني

عروض إدارة المباني تشمل عرض القائمة، والتفاصيل، والإضافة، والتعديل، والحذف. هذه العروض تستخدم مزيجاً من CBVs و FBVs لتوفير وظائف شاملة.

```python
 from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.core.paginator import Paginator
from django.db.models import Q, Count
from django.utils.translation import gettext as _

@login_required
def building_list(request):
    """عرض قائمة المبانى مع البحث والفلترة"""
    buildings = Building.objects.all()

    # البحث
    search_query = request.GET.get('search')
    if search_query:
        buildings = buildings.filter(
            Q(name__icontains=search_query) |
            Q(address__icontains=search_query)
        )

    # الفلترة حسب المرافق
    has_elevator = request.GET.get('has_elevator')
    if has_elevator:
        buildings = buildings.filter(has_elevator=True)

    has_parking = request.GET.get('has_parking')
    if has_parking:
        buildings = buildings.filter(has_parking=True)

    # الترتيب
    sort_by = request.GET.get('sort', 'name')
    if sort_by in ['name', 'total_floors', 'created_at']:
        buildings = buildings.order_by(sort_by)

    # الترقيم
    paginator = Paginator(buildings, 12)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)

    context = {
        'page_obj': page_obj,
        'search_query': search_query,
        'has_elevator': has_elevator,
        'has_parking': has_parking,
        'sort_by': sort_by,
    }

    return render(request, 'buildings/building_list.html', context)
```

عرض تفاصيل المبنى يوفر معلومات شاملة مع إحصائيات مفيدة:

```python
@login_required
def building_detail(request, building_id):
    """عرض تفاصيل المبنى مع الإحصائيات"""
    building = get_object_or_404(Building, id=building_id)

    # حساب الإحصائيات
    total_units = building.floors.aggregate(
        total=Count('units')
    )['total'] or 0

    occupied_units = building.floors.aggregate(
        occupied=Count('units', filter=Q(units__status='rented'))
    )['occupied'] or 0

    vacant_units = total_units - occupied_units
    occupancy_rate = (occupied_units / total_units * 100) if total_units > 0
else 0

    # الإيرادات الشهرية
    monthly_revenue = building.floors.aggregate(
        revenue=models.Sum('units__rent_price',
filter=Q(units__status='rented'))
    )['revenue'] or 0

    context = {
        'building': building,
        'total_units': total_units,
        'occupied_units': occupied_units,
        'vacant_units': vacant_units,
        'occupancy_rate': round(occupancy_rate, 1),
        'monthly_revenue': monthly_revenue,
    }

    return render(request, 'buildings/building_detail.html', context)
```

عروض إدارة المستأجرين

عروض المستأجرين تتطلب معالجة خاصة للبيانات الشخصية والوثائق المرفقة. هذه العروض تتضمن التحقق من صحة البيانات وإدارة الملفات.

```python
@login_required
def tenant_create(request):
    """إنشاء مستأجر جديد"""
    if request.method == 'POST':
        form = TenantForm(request.POST, request.FILES)
        if form.is_valid():
            tenant = form.save()

            # معالجة الوثائق المرفقة
            documents = request.FILES.getlist('documents')
            document_types = request.POST.getlist('document_types')

            for doc, doc_type in zip(documents, document_types):
                TenantDocument.objects.create(
                    tenant=tenant,
                    document_type=doc_type,
                    document_file=doc
                )

            messages.success(request, _('تم إنشاء المستأجر بنجاح'))
            return redirect('tenants:detail', tenant_id=tenant.id)
    else:
        form = TenantForm()

    context = {
        'form': form,
        'document_types': TenantDocument.DOCUMENT_TYPES,
    }

    return render(request, 'tenants/tenant_create.html', context)
```

عرض البحث المتقدم للمستأجرين يوفر فلترة متعددة المعايير:

```python
@login_required
def tenant_search(request):
    """البحث المتقدم في المستأجرين"""
    tenants = Tenant.objects.all()

    # البحث النصي
    search_query = request.GET.get('search')
    if search_query:
        tenants = tenants.filter(
            Q(first_name__icontains=search_query) |
            Q(last_name__icontains=search_query) |
            Q(civil_id__icontains=search_query) |
            Q(phone_number__icontains=search_query)
        )

    # الفلترة حسب الجنس
    gender = request.GET.get('gender')
    if gender:
        tenants = tenants.filter(gender=gender)

    # الفلترة حسب الحالة الاجتماعية
    marital_status = request.GET.get('marital_status')
    if marital_status:
        tenants = tenants.filter(marital_status=marital_status)

    # الفلترة حسب الجنسية
    nationality = request.GET.get('nationality')
    if nationality:
        tenants = tenants.filter(nationality=nationality)

    # الفلترة حسب النشاط
    is_active = request.GET.get('is_active')
    if is_active:
        tenants = tenants.filter(is_active=is_active == 'true')

    # الترتيب
    sort_by = request.GET.get('sort', 'first_name')
    if sort_by in ['first_name', 'last_name', 'created_at']:
        tenants = tenants.order_by(sort_by)

    # الترقيم
    paginator = Paginator(tenants, 15)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)

    # الحصول على قائمة الجنسيات للفلترة
    nationalities = Tenant.objects.values_list('nationality',
flat=True).distinct()

    context = {
        'page_obj': page_obj,
        'search_query': search_query,
        'selected_gender': gender,
        'selected_marital_status': marital_status,
        'selected_nationality': nationality,
        'selected_is_active': is_active,
        'sort_by': sort_by,
        'nationalities': nationalities,
    }
```

```
    return render(request, 'tenants/tenant_search.html', context)
```

عروض إدارة العقود

عروض العقود تتطلب منطق معقد لحساب التواريخ والمبالغ والتحقق من توفر الوحدات:

```python
@login_required
def lease_create(request):
    """إنشاء عقد إيجار جديد"""
    if request.method == 'POST':
        form = LeaseForm(request.POST)
        if form.is_valid():
            lease = form.save(commit=False)

            # التحقق من توفر الوحدة
            if lease.unit.status != 'vacant':
                messages.error(request, _('الوحدة المختارة غير متاحة'))
                return render(request, 'leases/lease_create.html', {'form':
form})

            # حساب تاريخ النهاية إذا لم يتم تحديده
            if not lease.end_date:
                lease.end_date = lease.start_date + timedelta(days=365)

            lease.save()

            # تحديث حالة الوحدة
            lease.unit.status = 'rented'
            lease.unit.save()

            # إنشاء جدولة الدفعات
            create_payment_schedule(lease)

            messages.success(request, _('تم إنشاء العقد بنجاح'))
            return redirect('leases:detail', lease_id=lease.id)
    else:
        form = LeaseForm()

    # الحصول على الوحدات المتاحة فقط
    available_units =
Unit.objects.filter(status='vacant').select_related('floor__building')

    context = {
        'form': form,
        'available_units': available_units,
    }

    return render(request, 'leases/lease_create.html', context)

def create_payment_schedule(lease):
    """إنشاء جدولة الدفعات للعقد"""
    current_date = lease.start_date
    payment_number = 1

    while current_date <= lease.end_date:
        PaymentSchedule.objects.create(
            lease=lease,
            due_date=current_date,
            amount=lease.monthly_rent,
            payment_number=payment_number,
            status='pending'
        )

        # الانتقال للشهر التالي
        if current_date.month == 12:
            current_date = current_date.replace(year=current_date.year + 1,
month=1)
```

```
        else:
            current_date = current_date.replace(month=current_date.month + 1)

        payment_number += 1
```

عروض التقارير والإحصائيات

عروض التقارير توفر تحليلات مفيدة لإدارة العقارات:

```python
@login_required
def financial_report(request):
    """تقرير مالي شامل"""
    # فترة التقرير
    start_date = request.GET.get('start_date')
    end_date = request.GET.get('end_date')

    if not start_date or not end_date:
        # افتراضياً: الشهر الحالي
        today = timezone.now().date()
        start_date = today.replace(day=1)
        end_date = (start_date + timedelta(days=32)).replace(day=1) -
timedelta(days=1)
    else:
        start_date = datetime.strptime(start_date, '%Y-%m-%d').date()
        end_date = datetime.strptime(end_date, '%Y-%m-%d').date()

    # الإيرادات
    payments = Payment.objects.filter(
        payment_date__range=[start_date, end_date],
        status='completed'
    )

    total_revenue = payments.aggregate(total=models.Sum('amount'))['total'] or
0
    rent_revenue = payments.filter(payment_type='rent').aggregate(
        total=models.Sum('amount')
    )['total'] or 0

    # المصروفات (الصيانة)
    maintenance_costs = MaintenanceRequest.objects.filter(
        completion_date__range=[start_date, end_date],
        status='completed'
    ).aggregate(total=models.Sum('cost'))['total'] or 0

    # صافي الربح
    net_profit = total_revenue - maintenance_costs

    # إحصائيات الإشغال
    total_units = Unit.objects.count()
    occupied_units = Unit.objects.filter(status='rented').count()
    occupancy_rate = (occupied_units / total_units * 100) if total_units > 0
else 0

    # الدفعات المتأخرة
    overdue_payments = PaymentSchedule.objects.filter(
        due_date__lt=timezone.now().date(),
        status='pending'
    ).count()

    context = {
        'start_date': start_date,
        'end_date': end_date,
        'total_revenue': total_revenue,
        'rent_revenue': rent_revenue,
        'maintenance_costs': maintenance_costs,
        'net_profit': net_profit,
        'total_units': total_units,
        'occupied_units': occupied_units,
        'occupancy_rate': round(occupancy_rate, 1),
        'overdue_payments': overdue_payments,
```

```
    }

    return render(request, 'reports/financial_report.html', context)
```

معالجة الأخطاء والاستثناءات

معالجة الأخطاء بشكل صحيح تحسن تجربة المستخدم وتساعد في تصحيح المشاكل:

```python
 from django.core.exceptions import ValidationError
from django.db import transaction
import logging

logger = logging.getLogger(__name__)

@login_required
def lease_terminate(request, lease_id):
    """إنهاء عقد الإيجار"""
    lease = get_object_or_404(Lease, id=lease_id)

    if request.method == 'POST':
        try:
            with transaction.atomic():
                # التحقق من إمكانية الإنهاء
                if lease.status != 'active':
                    raise ValidationError(_('لا يمكن إنهاء عقد غير نشط'))

                # إنشاء سجل الإنهاء
                termination = LeaseTermination.objects.create(
                    lease=lease,
                    termination_date=timezone.now().date(),
                    reason=request.POST.get('reason', ''),
                    early_termination=lease.end_date > timezone.now().date()
                )

                # تحديث حالة العقد والوحدة
                lease.status = 'terminated'
                lease.save()

                lease.unit.status = 'vacant'
                lease.unit.save()

                # إلغاء الدفعات المستقبلية
                PaymentSchedule.objects.filter(
                    lease=lease,
                    due_date__gt=termination.termination_date,
                    status='pending'
                ).update(status='cancelled')

                messages.success(request, _('تم إنهاء العقد بنجاح'))
                return redirect('leases:detail', lease_id=lease.id)

        except ValidationError as e:
            messages.error(request, str(e))
            logger.warning(f'Lease termination failed: {e}')
        except Exception as e:
            messages.error(request, _('حدث خطأ أثناء إنهاء العقد'))
            logger.error(f'Unexpected error in lease termination: {e}')

    context = {
        'lease': lease,
    }

    return render(request, 'leases/lease_terminate.html', context)
```

التصميم العماني والهوية البصرية

فلسفة التصميم العماني

التصميم العماني يجمع بين الأصالة والمعاصرة، حيث يستمد عناصره من التراث العماني الغني مع تطبيق المعايير الحديثة لتجربة المستخدم. الألوان الرسمية لسلطنة عمان تشمل الأحمر والأخضر والأبيض، والتي تحمل دلالات تاريخية وثقافية عميقة. الأحمر يرمز للقوة والشجاعة، والأخضر يرمز للخصوبة والازدهار، والأبيض يرمز للسلام والنقاء.

العمارة العمانية التقليدية تتميز بالأقواس والزخارف الهندسية والخط العربي الجميل. هذه العناصر يمكن تطبيقها في التصميم الرقمي من خلال استخدام أنماط هندسية في الخلفيات، وخطوط عربية أنيقة، وتخطيطات تحترم اتجاه القراءة من اليمين إلى اليسار.

الهوية البصرية للمشروع تعكس هذه القيم من خلال استخدام لوحة ألوان متناسقة، وتايبوغرافيا واضحة ومقروءة، وعناصر بصرية تستمد من التراث العماني دون إفراط. الهدف هو خلق تجربة مستخدم مألوفة ومريحة للمستخدمين العمانيين مع الحفاظ على المعايير الحديثة للتصميم.

تطبيق الألوان العمانية

لوحة الألوان المستخدمة في المشروع تستمد من العلم العماني والتراث المحلي:

```
:root {
    /* الألوان الأساسية العمانية */
    --primary-color: #C41E3A;      /* الأحمر العمانى */
    --secondary-color: #228B22;    /* الأخضر العمانى */
    --accent-color: #FFD700;       /* الذهبى للتمييز */
    --white-color: #FFFFFF;        /* الأبيض النقي */

    /* ألوان مساعدة */
    --light-red: #E8B4B8;         /* أحمر فاتح */
    --light-green: #90EE90;       /* أخضر فاتح */
    --dark-red: #8B0000;          /* أحمر داكن */
    --dark-green: #006400;        /* أخضر داكن */

    /* ألوان النظام */
    --success-color: var(--secondary-color);
    --danger-color: var(--primary-color);
    --warning-color: #FFA500;
    --info-color: #17A2B8;

    /* ألوان النص */
    --text-primary: #2C3E50;
    --text-secondary: #6C757D;
    --text-light: #ADB5BD;
}
```

تطبيق هذه الألوان يتم بطريقة متوازنة لضمان الوضوح والقابلية للقراءة:

```css
/* شريط التنقل */
.navbar {
    background: linear-gradient(135deg, var(--primary-color), var(--dark-red));
    box-shadow: 0 2px 10px rgba(196, 30, 58, 0.3);
}

/* الأزرار الأساسية */
.btn-primary {
    background: linear-gradient(135deg, var(--primary-color), var(--dark-red));
    border: none;
    transition: all 0.3s ease;
}

.btn-primary:hover {
    background: linear-gradient(135deg, var(--dark-red), var(--primary-color));
    transform: translateY(-2px);
    box-shadow: 0 4px 15px rgba(196, 30, 58, 0.4);
}

/* الأزرار الثانوية */
.btn-success {
    background: linear-gradient(135deg, var(--secondary-color), var(--dark-
green));
    border: none;
}

/* البطاقات */
.card {
    border: none;
    border-radius: 15px;
    box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
    transition: all 0.3s ease;
}

.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);
}
```

الخطوط والتايبوغرافيا

اختيار الخطوط المناسبة للغة العربية أمر بالغ الأهمية لضمان الوضوح والجمالية:

```css
/* خطوط عربية احترافية */
@import url('https://fonts.googleapis.com/css2?
family=Noto+Sans+Arabic:wght@300;400;500;600;700&display=swap');
@import url('https://fonts.googleapis.com/css2?
family=Amiri:wght@400;700&display=swap');

body {
    font-family: 'Noto Sans Arabic', 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
    font-size: 16px;
    line-height: 1.6;
    color: var(--text-primary);
}

/* العناوين */
h1, h2, h3, h4, h5, h6 {
    font-family: 'Amiri', 'Noto Sans Arabic', serif;
    font-weight: 700;
    color: var(--primary-color);
    margin-bottom: 1rem;
}

h1 { font-size: 2.5rem; }
h2 { font-size: 2rem; }
h3 { font-size: 1.75rem; }
h4 { font-size: 1.5rem; }
h5 { font-size: 1.25rem; }
h6 { font-size: 1rem; }

/* النصوص الخاصة */
.brand-text {
    font-family: 'Amiri', serif;
    font-weight: 700;
    font-size: 1.5rem;
    color: var(--white-color);
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
}
```

الأنماط الهندسية والزخارف

استخدام الأنماط الهندسية الإسلامية يضفي طابعاً أصيلاً على التصميم:

```css
/* خلفية بنمط هندسي عماني */
.geometric-bg {
    background-image:
        radial-gradient(circle at 25% 25%, var(--accent-color) 2px, transparent 2px),
        radial-gradient(circle at 75% 75%, var(--light-green) 2px, transparent 2px),
        linear-gradient(45deg, transparent 49%, var(--light-red) 49%, var(--light-red) 51%, transparent 51%);
    background-size: 50px 50px, 50px 50px, 20px 20px;
    opacity: 0.1;
}

/* حدود مزخرفة */
.decorated-border {
    border: 3px solid var(--primary-color);
    border-image: linear-gradient(45deg, var(--primary-color), var(--accent-color), var(--secondary-color)) 1;
    position: relative;
}

.decorated-border::before {
    content: '';
    position: absolute;
    top: -6px;
    left: -6px;
    right: -6px;
    bottom: -6px;
    background: linear-gradient(45deg, var(--accent-color), transparent, var(--accent-color));
    z-index: -1;
    border-radius: inherit;
}

/* أيقونات مخصصة */
.icon-oman {
    color: var(--primary-color);
    filter: drop-shadow(2px 2px 4px rgba(196, 30, 58, 0.3));
}
```

التأثيرات الحركية والتفاعلية

إضافة تأثيرات حركية مناسبة تحسن تجربة المستخدم:

```css
/* تأثيرات الظهور */
@keyframes fadeInUp {
    from {
        opacity: 0;
        transform: translateY(30px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

.fade-in-up {
    animation: fadeInUp 0.6s ease-out;
}

/* تأثيرات التحويم */
.hover-lift {
    transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

.hover-lift:hover {
    transform: translateY(-8px);
    box-shadow: 0 15px 35px rgba(0, 0, 0, 0.1);
}

/* تأثيرات الأزرار */
.btn-animated {
    position: relative;
    overflow: hidden;
    transition: all 0.3s ease;
}

.btn-animated::before {
    content: '';
    position: absolute;
    top: 0;
    left: -100%;
    width: 100%;
    height: 100%;
    background: linear-gradient(90deg, transparent, rgba(255, 255, 255, 0.2),
transparent);
    transition: left 0.5s;
}

.btn-animated:hover::before {
    left: 100%;
}

/* تأثيرات النبض للعناصر المهمة */
@keyframes pulse {
    0% { box-shadow: 0 0 0 0 rgba(196, 30, 58, 0.7); }
    70% { box-shadow: 0 0 0 10px rgba(196, 30, 58, 0); }
    100% { box-shadow: 0 0 0 0 rgba(196, 30, 58, 0); }
}

.pulse-effect {
    animation: pulse 2s infinite;
}
```

التصميم المتجاوب

ضمان عمل التصميم على جميع الأجهزة:

```
/* نقاط التوقف للشاشات المختلفة */
@media (max-width: 768px) {
    .navbar-brand {
        font-size: 1.2rem;
    }

    .card {
        margin-bottom: 1rem;
    }

    .btn {
        width: 100%;
        margin-bottom: 0.5rem;
    }

    h1 { font-size: 2rem; }
    h2 { font-size: 1.75rem; }
    h3 { font-size: 1.5rem; }
}

@media (max-width: 576px) {
    .container-fluid {
        padding-left: 10px;
        padding-right: 10px;
    }

    .card-body {
        padding: 1rem;
    }

    .table-responsive {
        font-size: 0.875rem;
    }
}

/* تحسينات للشاشات الكبيرة */
@media (min-width: 1200px) {
    .container {
        max-width: 1140px;
    }

    .card-columns {
        column-count: 4;
    }
}
```

الخلاصة والتوصيات

ملخص المشروع

لقد تم تطوير نظام إدارة الإيجارات في سلطنة عمان كمشروع تعليمي شامل يجمع بين أفضل الممارسات في تطوير تطبيقات الويب والخصائص المحلية العمانية. المشروع يغطي جميع جوانب إدارة العقارات من إدارة المباني والمستأجرين والعقود والدفعات والصيانة والتقارير.

النظام يتميز بدعم كامل للغة العربية مع نظام RTL، وتصميم عماني أصيل يستخدم الألوان الرسمية للسلطنة، وواجهات مستخدم حديثة ومتجاوبة، ونظام شامل للترجمة والتدويل. كما يتضمن ميزات متقدمة مثل التحقق من صحة البيانات العمانية، وإدارة الوثائق، والتقارير المالية المفصلة.

المشروع يستخدم Python 3.11 مع Django 4.2 من الناحية التقنية، ويطبق أفضل الممارسات في تصميم قواعد البيانات والأمان وتجربة المستخدم. الكود منظم ومُوثق بشكل جيد، مما يجعله مرجعاً ممتازاً للتعلم والتطوير.

الدروس المستفادة

خلال تطوير هذا المشروع، تم تعلم العديد من الدروس المهمة في تطوير التطبيقات المحلية. أولاً، أهمية فهم السياق الثقافي والقانوني للسوق المستهدف، حيث أن دعم اللغة العربية والتصميم العماني ليس مجرد إضافة جمالية بل ضرورة لقبول المستخدمين للنظام.

ثانياً، أهمية التخطيط الجيد لهيكل قاعدة البيانات والعلاقات بين النماذج. التصميم الصحيح من البداية يوفر الكثير من الوقت والجهد في المراحل اللاحقة، ويضمن قابلية التوسع والصيانة.

ثالثاً، قيمة استخدام أنماط التصميم المُجربة مثل MVC وDRY (Don't Repeat Yourself). هذه الأنماط تجعل الكود أكثر تنظيماً وقابلية للفهم والصيانة.

رابعاً، أهمية الاختبار المستمر والتحسين التدريجي. بدلاً من محاولة إنجاز كل شيء مرة واحدة، التطوير التدريجي مع الاختبار المستمر يؤدي إلى نتائج أفضل وأكثر استقراراً.

التوصيات للتطوير المستقبلي

لتطوير النظام أكثر، يُنصح بإضافة عدة ميزات متقدمة. أولاً، تطوير تطبيق جوال مصاحب باستخدام Flutter أو React Native لتوفير وصول أسهل للمستأجرين وأصحاب العقارات. هذا التطبيق يمكن أن يتضمن إشعارات الدفعات وطلبات الصيانة والتواصل المباشر.

ثانياً، إضافة نظام دفع إلكتروني متكامل يدعم البنوك العمانية المحلية وطرق الدفع الشائعة. هذا يسهل على المستأجرين دفع الإيجارات ويقلل من العمل الإداري.

ثالثاً، تطوير نظام ذكي للتنبيهات والتذكيرات باستخدام الذكاء الاصطناعي لتحليل أنماط الدفع والتنبؤ بالمشاكل المحتملة. هذا يساعد في الإدارة الاستباقية للعقارات.

رابعاً، إضافة تكامل مع الخدمات الحكومية العمانية مثل نظام الهوية المدنية للتحقق الآلي من بيانات المستأجرين، ونظام الأراضي والمساحة للتحقق من ملكية العقارات.

خامساً، تطوير نظام تحليلات متقدم يوفر رؤى عميقة حول أداء العقارات واتجاهات السوق. هذا يساعد أصحاب العقارات في اتخاذ قرارات استثمارية مدروسة.

نصائح للمطورين الجدد

من خلال هذا المشروع، يُنصح بالبدء بفهم المفاهيم الأساسية Django للمطورين الذين يرغبون في تعلم وكيفية تصميم قواعد البيانات، ثم انتقل إلى Django قبل الغوص في التفاصيل. ابدأ بدراسة نماذج العروض والقوالب.

لا تتردد في تجربة وتعديل الكود. أفضل طريقة للتعلم هي الممارسة العملية. جرب إضافة ميزات جديدة أو تعديل الموجود لفهم كيفية عمل الأجزاء المختلفة معاً.

لفهم ما يحدث خلف الكواليس. هذه Django Debug Toolbar استخدم أدوات التطوير المناسبة مثل الأدوات تساعد في تحسين الأداء وفهم سلوك التطبيق.

بانتظام. الوثائق شاملة ومُحدثة وتحتوي على أمثلة عملية مفيدة. كما Django اقرأ الوثائق الرسمية لـ أن مجتمع Django نشط ومفيد للحصول على المساعدة.

أخيراً، لا تنس أهمية الأمان. تعلم أفضل الممارسات في أمان تطبيقات الويب وطبقها من البداية. الأمان ليس شيئاً يُضاف لاحقاً بل يجب أن يكون جزءاً من التصميم الأساسي.

الخاتمة

نظام إدارة الإيجارات في سلطنة عمان يمثل مثالاً شاملاً على كيفية تطوير تطبيقات ويب حديثة ومحلية باستخدام Django. المشروع يجمع بين التقنيات المتقدمة والاحتياجات المحلية، مما يجعله مرجعاً قيماً للمطورين والطلاب.

الهدف من هذا المشروع ليس فقط تعليم Django، بل أيضاً إظهار كيفية تطوير حلول تقنية تخدم المجتمع المحلي وتحترم الثقافة والتقاليد. هذا النهج ضروري لنجاح أي مشروع تقني في الأسواق المحلية.

نأمل أن يكون هذا المشروع نقطة انطلاق لمشاريع أكثر تطوراً وإبداعاً تخدم المجتمع العماني والعربي. التقنية أداة قوية للتطوير والتقدم، وعندما تُستخدم بحكمة وفهم للسياق المحلي، يمكنها أن تحدث تأثيراً إيجابياً حقيقياً.

المؤلف: Manus AI
التاريخ: يونيو 2025
الإصدار: 1.0

هذا المشروع مفتوح المصدر ومتاح للتطوير والتحسين من قبل المجتمع التقني العماني والعربي.