

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



دانشگاه سمنان

دانشکده مهندسی برق و کامپیوتر

پایان نامه کارشناسی مهندسی کامپیوتر

عنوان:

طراحی و پیاده سازی سامانه اینترنتی اجاره ویلا و آپارتمان

نگارش:

ایلیا جعفری چمازکتی

استاد راهنما:

جناب آقای دکتر محمد رحمانی منش

بهمن ۱۴۰۱

تعهدنامه

اینجانب ایلیا جعفری چمازکتی بدین وسیله اظهار می دارم که محتوای علمی این نوشتار با عنوان "طراحی و پیاده سازی سامانه اینترنتی اجاره ویلا و آپارتمان" که به عنوان پایان نامه کارشناسی مهندسی کامپیوتر به دانشکده برق و کامپیوتر دانشگاه سمنان ارائه شده، دارای اصالت پژوهشی بوده و حاصل فعالیت های این جانب است.

این جانب می دانم که اگر خلاف ادعای بالا در هر زمانی محرز شود، کلیه حقوق مرتبط بر این نوشتار، از این جانب سلب شده و مراتب قانونی مرتبط با آن نیز از طرف مراجع ذیربط قابل پیگیری است.

ایلیا جعفری چمازکتی ۹۷۱۱۱۲۶۰۲۹

تاریخ و امضا



دانشگاه سمنان

دانشکده مهندسی برق و کامپیوتر

تایید دفاع از پایان نامه کارشناسی

پایان نامه ایلیا جعفری چمازکتی

با عنوان:

طراحی و پیاده سازی سامانه اینترنتی اجاره ویلا و آپارتمان

در تاریخ دفاع شد و مورد تایید قرار گرفت.

تایید کنندگان:

استاد محترم راهنما: امضا

استاد محترم داور: امضا

مدیر گروه محترم کامپیوتر: امضا

پاسکزاری

اکنون که بیاری پروردگار و راهمائی استید محترم موفق به انجام این پروژه شده ام، وظیفه خود دانستم که نهایت پاسکزاری را از تمامی عزیزانی که بنده را در این مسیر یاری کرده اند به عمل آورم:

در آغاز از استاد بزرگوارم جناب آقای دکتر رحمانی نش که راهمائی این پروژه را به عهده داشته اند کمال تشکر و قدردانی را دارم و از خداوند متعال پیروزی و موفقیت روز افزون ایشان را خواستارم.

پس از استید ارجمند دانشگاه سمنان که شاکردی محضرشان از بزرگترین افتخارات زندگی علمی ام می باشد، برای تمام حمایت ها و زحمات بی دریغشان پاسکزارم.

و در پایان، از مادر مهربانم و پدر بزرگوارم، آن دو فرشته ای که از خواسته هایشان گذشتند و سختی ها را به جان خریدند تا من به جایگاهی که اکنون در آن ایستاده ام برسم، کمال قدردانی و سپاس را دارم.

چکیده

امروزه اغلب کسب و کارها در حال خروج از فضای سنتی و در حال ورود به فضای دیجیتال هستند که برای انجام چنین تحولاتی ابزارهای مختلفی وجود دارد؛ مثلاً استفاده از شبکه‌های اجتماعی مختلف و تاسیس یک صفحه ویژه برای کسب و کار یا استفاده از پلتفرم‌های مختلف مجازی برای ارائه خدمات و محصولات به طیف گسترده‌ای از مشتریان.

در همین راستا، راه‌اندازی یک وبسایت منحصر به فرد برند و نام تجاری شاید یکی از بهترین تصمیماتی باشد که باید برای مهاجرت از فضای سنتی به فضای آنلاین اجرایی شود. صنعت هتلداری نیز از این قاعده مستثنی نیست. در این پروژه سعی شده که نمونه‌ای از یک سامانه اینترنتی اجاره ویلا و آپارتمان، با فریم‌ورک جنگو پیاده‌سازی شود.

واژه‌های کلیدی:

کسب و کار دیجیتال، صنعت هتلداری، سامانه اینترنتی، جنگو، فریم‌ورک جنگو، Django، پایتون، Python

فهرست مطالب

فصل اول مقدمه	۱
۱-۱ جنگو چیست؟	۲
۲-۱ چرا جنگو؟	۲
۱-۲-۱ ویژگی های جنگو:	۲
۳-۱ توسعه دهنده کیست؟	۳
۴-۱ توسعه دهنده وب کیست؟	۳
۱-۴-۱ توسعه دهنده وب در سمت کاربر:	۳
۲-۴-۱ توسعه دهنده وب در سمت سرور:	۴
۳-۴-۱ توسعه دهنده وب کامل:	۴
۵-۱ ابزار های استفاده شده در این پروژه	۴
۶-۱ تکنولوژی های استفاده شده در این پروژه	۵
۷-۱ داده، پایگاه داده و ORM	۵
۸-۱ معماری MVT	۶
Model ۱-۸-۱:	۶
View ۲-۸-۱:	۶
Template ۳-۸-۱:	۶
فصل دوم آماده سازی محیط پروژه	۷
۱-۲ ایجاد مخزن کد یا ریپازیتوری	۸
۲-۲ محیط مجازی پایتون	۹
۳-۲ چه زمانی باید از محیط مجازی استفاده کرد؟	۹

۴-۲ نحوه عملکرد محیط مجازی	۱۰
۵-۲ نسخه جنگو مورد استفاده در این پروژه	۱۰
۶-۲ شروع ساخت پروژه جنگو	۱۱
۷-۲ فایل های ایجاد شده پس از ساخت پروژه	۱۱
۱-۷-۲ فایل manage.py:	۱۲
۲-۷-۲ فایل __init__.py:	۱۲
۳-۷-۲ فایل settings.py:	۱۲
۴-۷-۲ فایل urls.py:	۱۲
۸-۲ ساخت اپلیکیشن	۱۲
۹-۲ ایجاد پایگاه داده و superuser	۱۴
فصل سوم توسعه مدل ها	۱۵
۱-۳ فیلد های مدل جنگو	۱۶
۲-۳ رابطه ها در جداول	۱۷
۳-۳ مدل Meta	۱۷
۴-۳ توسعه مدل ها از طریق فایل models.py	۱۸
۱-۴-۳ توسعه مدل users:	۱۸
۲-۴-۳ توسعه مدل rooms و core:	۲۰
۳-۴-۳ توسعه مدل reviews	۲۴
۴-۴-۳ توسعه مدل reservation:	۲۶
۵-۴-۳ توسعه مدل lists:	۲۸
۶-۴-۳ توسعه مدل conversation:	۲۸
۵-۳ اضافه کردن اطلاعات مدل ها به ادمین	۳۰

فصل چهارم توسعه View ها و URL ها	۳۱
۱-۴ توسعه ویو از طریق فایل views.py	۳۲
۱-۱-۴ توسعه ویو users	۳۲
۲-۱-۴ توسعه ویو rooms	۳۳
۳-۱-۴ توسعه ویو reviews	۳۴
۴-۱-۴ توسعه ویو reservations	۳۵
۵-۱-۴ توسعه ویو lists	۳۶
۶-۱-۴ توسعه ویو conversations	۳۷
۲-۴ توسعه URL ها از طریق فایل urls.py	۳۷
۱-۲-۴ توسعه آدرس های users	۳۸
۲-۲-۴ توسعه آدرس های rooms	۳۸
۳-۲-۴ توسعه آدرس های reviews	۳۹
۴-۲-۴ توسعه آدرس های reservations	۳۹
۵-۲-۴ توسعه آدرس های lists	۳۹
۶-۲-۴ توسعه آدرس های conversations	۳۹
۷-۲-۴ توسعه آدرس های core	۴۰
۸-۲-۴ توسعه آدرس های config	۴۰
فصل پنجم توسعه Template ها	۴۱
۱-۵ تفاوت فایل های Media و فایل های Static	۴۲
۲-۵ نحوه تنظیم Static و Media در پروژه	۴۲
۳-۵ نحوه تنظیم Template ها در پروژه	۴۲
۴-۵ پیاده سازی تمپلیت ها	۴۳

۴۴.....	۱-۴-۵ توسعه base.html :
۴۴.....	۲-۴-۵ توسعه 404.html :
۴۵.....	۳-۴-۵ توسعه تمپلیت های گروه users :
۴۶.....	۴-۴-۵ توسعه تمپلیت های گروه rooms :
۴۶.....	۵-۴-۵ توسعه تمپلیت های گروه reservation :
۴۷.....	۶-۴-۵ توسعه تمپلیت های گروه partials :
۴۷.....	۷-۴-۵ توسعه تمپلیت های گروه mixins :
۴۸.....	۷-۴-۵ توسعه تمپلیت های گروه lists :
۴۸.....	۸-۴-۵ توسعه تمپلیت های گروه conversations :
۵۰.....	فصل ششم توسعه فرم ها :
۵۱.....	۱-۶ تعریف فرم ها :
۵۱.....	۱-۱-۶ فرم جنگو.....
۵۲.....	۲-۶ توسعه فرم ها از طریق فایل forms.py :
۵۲.....	۱-۲-۶ فرم users :
۵۳.....	۲-۲-۶ فرم rooms :
۵۵.....	۳-۲-۶ فرم reviews :
۵۵.....	۴-۲-۶ فرم conversations :
۵۶.....	مراجع :

فهرست تصاویر

تصویر ۱-۱ نمایی از یک پایگاه داده.....	۵
تصویر ۲-۱ نحوه ارتباط کاربر با مدل MVT و نقشه ORM.....	۶
تصویر ۱-۲ منوی ساخت رپازیتوری در GitHub Desktop.....	۸
تصویر ۲-۲ کامیت کردن تغییرات در GitHub Desktop.....	۸
تصویر ۳-۲ نقش محیط مجازی در مدیریت نسخه های مختلف.....	۹
تصویر ۴-۲ صفحه اصلی وب سایت پس از ساخت موفق پروژه جنگو.....	۱۱
تصویر ۵-۲ اضافه کردن نام پروژه به لیست PROJECT_APPS.....	۱۳
تصویر ۱-۳ انواع فیلد ها در مدل جنگو.....	۱۶
تصویر ۲-۳ آپشن های موجود برای فیلد های مدل.....	۱۷
تصویر ۳-۳ مدل users.....	۱۹
تصویر ۴-۳ فایل managers.py.....	۲۰
تصویر ۵-۳ مدل core.....	۲۰
تصویر ۶-۳ مدل rooms.....	۲۲
تصویر ۷-۳ توابع مدل rooms.....	۲۳
تصویر ۸-۳ مدل reviews.....	۲۵
تصویر ۹-۳ توابع مدل reviews.....	۲۵
تصویر ۱۰-۳ مدل reservations.....	۲۶
تصویر ۱۱-۳ توابع مدل reservations.....	۲۷
تصویر ۱۲-۳ مدل lists.....	۲۸
تصویر ۱۳-۳ مدل conversations.....	۲۹
تصویر ۱۴-۳ فایل ادمین users.....	۳۰
تصویر ۱-۴ ویو users.....	۳۳
تصویر ۲-۴ ویو rooms.....	۳۴
تصویر ۳-۴ ویو reviews.....	۳۵

۳۶.....	تصویر ۴-۴ ویو reservations
۳۶.....	تصویر ۵-۴ ویو lists
۳۷.....	تصویر ۶-۴ ویو conversations
۳۸.....	تصویر ۷-۴ URL های users
۳۸.....	تصویر ۸-۴ URL های rooms
۳۹.....	تصویر ۹-۴ URL های reviews
۳۹.....	تصویر ۱۰-۴ URL های reservations
۳۹.....	تصویر ۱۱-۴ URL های lists
۳۹.....	تصویر ۱۲-۴ URL های conversations
۴۰.....	تصویر ۱۳-۴ URL های core
۴۰.....	تصویر ۱۴-۴ URL های config
۴۳.....	تصویر ۱-۵ تنظیم تمپلیت ها در پروژه
۴۳.....	تصویر ۲-۵ سازماندهی تمپلیت ها
۴۵.....	تصویر ۳-۵ فایل 404.html
۴۵.....	تصویر ۴-۵ تمپلیت لاگین
۴۶.....	تصویر ۵-۵ تمپلیت سرچ
۴۶.....	تصویر ۶-۵ تمپلیت جزئیات رزرواسیون
۴۷.....	تصویر ۷-۵ تمپلیت فوتر
۴۸.....	تصویر ۸-۵ تمپلیت لیست ها
۴۹.....	تصویر ۹-۵ تمپلیت مکالمه
۵۱.....	تصویر ۱-۶ دیاگرام روش کار فرم ها
۵۲.....	تصویر ۲-۶ فرم لاگین کاربر
۵۳.....	تصویر ۳-۶ فرم ثبت نام کاربر
۵۴.....	تصویر ۴-۶ فرم جستجو بین اتاق ها
۵۵.....	تصویر ۵-۶ فرم reviews
۵۵.....	تصویر ۶-۶ فرم conversations

فصل اول

مقدمه

در این فصل به شرح بعضی از مقدمات پروژه پرداخته خواهد شد.

۱-۱ جنگو^۱ چیست؟

جنگو یکی از وب فریم‌ورک^۲ های قدرتمند، منبع آزاد^۳ و رایگان پایتون^۴ است که توسعه سریع، کاربردی و زیبایی یک وب سایت را میسر می‌کند. جنگو در سال ۲۰۰۸ میلادی توسط بنیاد نرم‌افزار جنگو^۵ خلق شد که هدفش آسان سازی توسعه وب با پایتون بود که پیشرفت چشمگیر این فریم‌ورک از آن سال تا کنون ادله ای بر موفقیت این بنیاد است.

جنگو ساختاری ماژولار دارد و می‌توان بر اساس نیاز پروژه در آن عملکردهای مختلفی را اعمال کرد که به کار توسعه‌دهنده سهولت بیشتری می‌بخشد.

۱-۲ چرا جنگو؟

جنگو از لحاظ عملکرد بسیار سریع است و قابلیت انعطاف زیادی در پروژه‌های مختلف را دارد. همچنین با جنگو، با صرف کم‌ترین زمان ممکن می‌توان بخش بک‌اند یک وب سایت را توسعه داد.

۱-۲-۱ ویژگی های جنگو:

ویژگی های جنگو بر اساس اسناد وب سایت جنگو [1]:

۱. به طرز شگفت آوری سریع است! جنگو برای کمک به توسعه دهندگان ای طراحی شده که می‌خواهند یک ایده را در اسرع وقت به یک محصول نهایی تبدیل کنند.

۲. بسیار امن! جنگو امنیت را جدی می‌گیرد و به توسعه دهندگان کمک می‌کند تا از بسیاری از اشتباهات رایج امنیتی جلوگیری نمایند. البته لازم به ذکر است که این هشدارها توسط توسعه دهنده قابل چشم پوشی می‌باشند.

۳. بسیار مقیاس پذیر! تعدادی از وب سایت‌های مشهور و پرترافیک (مانند اینستاگرام، یوتیوب و اسپاتیفای) از توانایی جنگو در مقیاس‌پذیری و انعطاف پذیر برای پاسخگویی به سنگین ترین درخواست ها استفاده می‌کنند.

¹ Django

² Web Framework

³ Open Source

⁴ Python

⁵ Django Software Foundation (DSF)

۴. جامع و تنوع عملکرد! شرکت‌ها، سازمان‌ها و دولت‌ها از جنگو برای ساخت انواع وب سایت‌ها - از سیستم‌های مدیریت محتوا تا شبکه‌های اجتماعی - استفاده می‌کنند.

۱-۳ توسعه دهنده^۶ کیست؟

توسعه‌دهنده فردی است که کد یا سورس یک نرم افزار کاربردی را می‌نویسد، اشکال‌زدایی و اجرا می‌کند. توسعه‌دهنده همچنین به عنوان توسعه‌دهنده نرم‌افزار، برنامه‌نویس کامپیوتر، برنامه‌نویس یا مهندس نرم‌افزار نیز شناخته می‌شود.

۱-۴ توسعه دهنده وب^۷ کیست؟

توسعه‌دهنده وب، برنامه‌نویسی است که در توسعه برنامه‌های وب جهانی با استفاده از مدل کلاینت - سرور^۸ تخصص دارد. توسعه دهنده های وب معمولاً به سه دسته تقسیم می‌شوند: توسعه دهنده وب در سمت کاربر، توسعه دهنده وب در سمت سرور و توسعه دهنده وب کامل.

۱-۴-۱ توسعه دهنده وب در سمت کاربر^۹:

توسعه‌دهنده وب در سمت کاربر وظیفه‌ی ساخت نمای یک وب‌سایت را بر عهده دارد، یا به عبارتی دیگر چگونه رنگ‌ها، نمادها و تصاویر در کنار یکدیگر ظاهر شوند. این توسعه دهنده باید ظاهر وب‌سایت در همه دستگاه‌ها، از دسکتاپ گرفته تا تبلت و تلفن همراه را در نظر بگیرد.

این توسعه دهنده معمولاً از HTML، JavaScript و CSS استفاده می‌کند.

^۶ Developer

^۷ Web Developer

^۸ Client-Server

^۹ Front-End Developer

۱-۴-۲ توسعه دهنده وب در سمت سرور^{۱۰}:

توسعه دهنده وب در سمت سرور مسئول ایجاد و نگهداری کدی است که منطق وبسایت را اجرا می‌کند. این کد وبسایت را به سرور متصل می‌کند و اطمینان حاصل می‌کند که داده‌ها به‌درستی به وبسایت منتقل می‌شوند و تراکنش‌ها به‌درستی پردازش می‌شوند. همچنین این توسعه دهنده به حفظ و نگهداری داده در پایگاه داده^{۱۱} نیز رسیدگی می‌کنند و اتصال سایت با این پایگاه را برقرار می‌نماید.

زبان‌های برنامه‌نویسی این حوزه معمولاً شامل PHP، Java و تکنولوژی‌های جدیدتری مثل Django و Golang است.

۱-۴-۳ توسعه دهنده وب کامل^{۱۲}:

توسعه‌دهنده کامل، هم مسئولیت‌های سمت کاربر و هم سرور را پوشش می‌دهد. بسته به پیچیدگی یک وبسایت، یک توسعه‌دهنده کامل می‌تواند مسئول همه‌ی جوانب توسعه آن، از سمت سرور تا رابط کاربری باشد. بسیاری از توسعه دهندگان کامل وب، معمولاً بیشتر در یک جنبه از توسعه وب تخصص دارند.

۱-۵ ابزار های استفاده شده در این پروژه

در مسیر فرآیند توسعه این وب سایت از ۳ ابزار زیر استفاده شده است:

۱. Visual Studio Code: ویرایشگر کد توسعه داده شده توسط مایکروسافت که اجازه مدیریت، ساخت

پروژه و برنامه‌نویسی در یک محیط مجتمع به همراه ابزارهای اولیه را می‌دهد. [2]

۲. GitHub Desktop: نسخه گرافیکی Git می‌باشد. [3]

۳. Git: یک ابزار بسیار کاربردی برای توسعه دهندگان در سطوح مختلف می‌باشد که امکان کنترل و مدیریت

نسخه‌های پروژه^{۱۳} را در طی مسیر ایجاد می‌کند. [4]

¹⁰ Back-End Developer

¹¹ Database

¹² Full-Stack Web Developer

¹³ Version Control

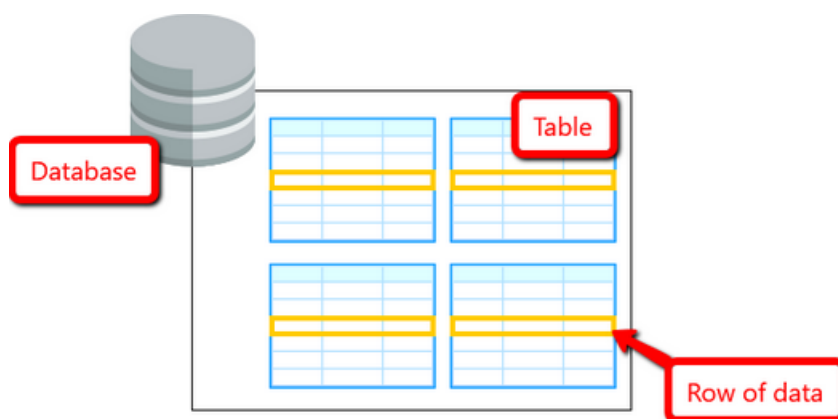
۱-۶ تکنولوژی های استفاده شده در این پروژه

- جنگو نسخه ۲.۲.۵
- پایتون
- HTML
- CSS
- JavaScript
- TailwindCSS

۱-۷ داده، پایگاه داده و ORM

داده‌ها^{۱۴} را می‌توان به عنوان مجموعه ای از حقایق و سوابق تعریف کرد که مقابلیت پردازش شدن دارند. داده‌ها می‌توانند به صورت گرافیکی، گزارش، جدول، متن و غیره وجود داشته باشند.

اطلاعات جمع آوری شده به صورت سازمان یافته برای دسترسی آسان، مدیریت و به روزرسانی‌های مختلف، به عنوان پایگاه داده شناخته می‌شود.



تصویر ۱-۱ نمایی از یک پایگاه داده

یکی از قدرتمندترین ویژگی‌های جنگو، نقشه Object-Relational Mapper (ORM) است که این امکان را میسر می‌کند که مانند SQL با پایگاه داده تعامل برقرار شود. در واقع ORM جنگو یک روش برای ساخت و دستکاری پایگاه داده با محوریت زبان پایتون است که بدون نیاز به تغییر نوع query ها با یک پایگاه داده های متفاوت ارتباط برقرار می‌شود.

¹⁴ Data

۸-۱ معماری MVT

معماری MVT یک الگوی طراحی نرم افزاری^{۱۵} است که از سه بخش مهم Model، View و Template تشکیل شده است.

۸-۱-۱ Model:

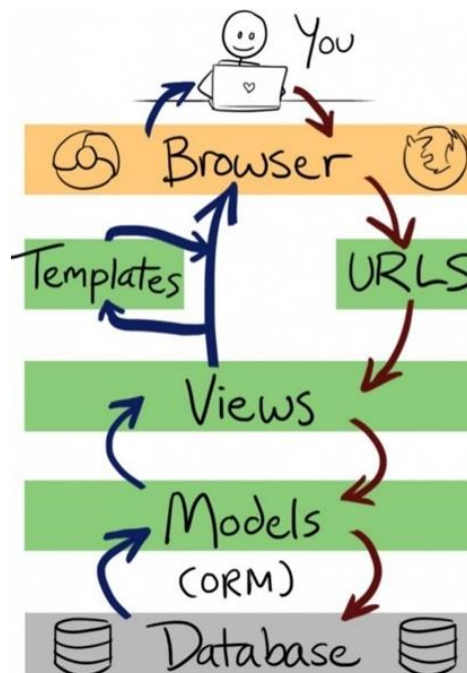
این بخش در جنگو مسئول مدیریت ساختار اطلاعات ذخیره شده در پایگاه داده است.

۸-۱-۲ View:

این بخش مسئول پردازش درخواست‌ها است. این بخش رابطی است که بخش‌های model و template را به هم وصل میکند. درخواست‌های کاربران در این بخش پردازش شده و پاسخ مناسب به آن‌ها نشان داده خواهد شد.

۸-۱-۳ Template:

در این بخش نتیجه نهایی به کاربر نشان داده میشود. همچنین میتوان در این بخش از کاربر اطلاعات و ورودی دریافت کرد و به دیگر بخش‌های برنامه ارسال کرد. نمایش templates جنگو وظیفه بخش view است. تمام فایل‌های استاتیک^{۱۶} مانند فایل‌های HTML یا CSS در این بخش مدیریت می‌شوند.



تصویر ۲-۱ نحوه ارتباط کاربر با مدل MVT و نقشه ORM

¹⁵ Software Pattern Design

¹⁶ Static

فصل دوم

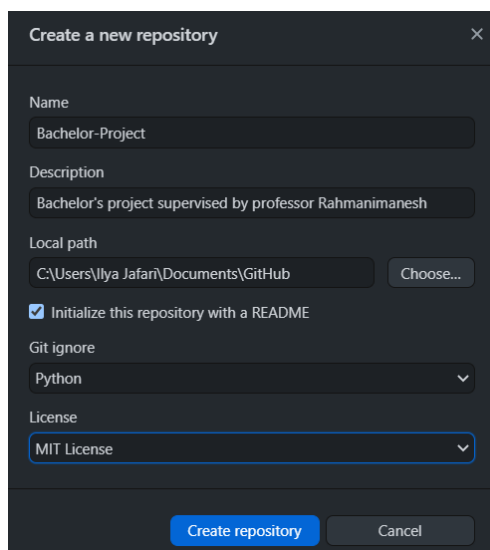
آماده سازی محیط پروژه

در این فصل به اقدامات لازم برای آماده سازی محیط پروژه پرداخته خواهد شد.

۱-۲ ایجاد مخزن کد یا ریپازیتوری^{۱۷}

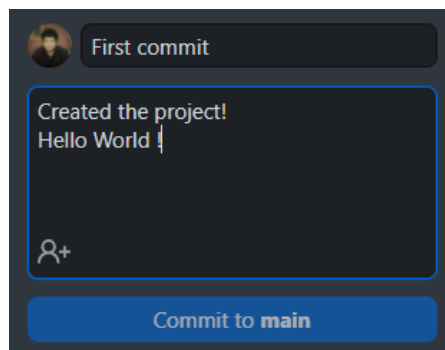
همانطور که در فصل اول اشاره شد، در این پروژه برای کنترل نسخه از نسخه دسکتاپ گیت‌هاب^{۱۸} استفاده شده است که نیاز به وارد کردن دستورات گیت از طریق ترمینال^{۱۹} را برطرف کرده است و با استفاده از یک رابط کاربری^{۲۰} ساده، می‌توان نسخه‌ها را مدیریت کرد.

برای ایجاد مخزن کد یا ریپازیتوری وارد محیط نرم افزار شده و مشخصات ریپازیتوری را وارد می‌کنیم. [5]



تصویر ۱-۲ منوی ساخت ریپازیتوری در GitHub Desktop

از این پس تمام تغییرات در ستون سمت چپ محیط نرم افزار نمایش داده می‌شود که می‌توان با افزودن نام و توصیفات لازمه، آن را Commit و سپس Push کرد.



تصویر ۲-۲ کامیت کردن تغییرات در GitHub Desktop

¹⁷ Repository

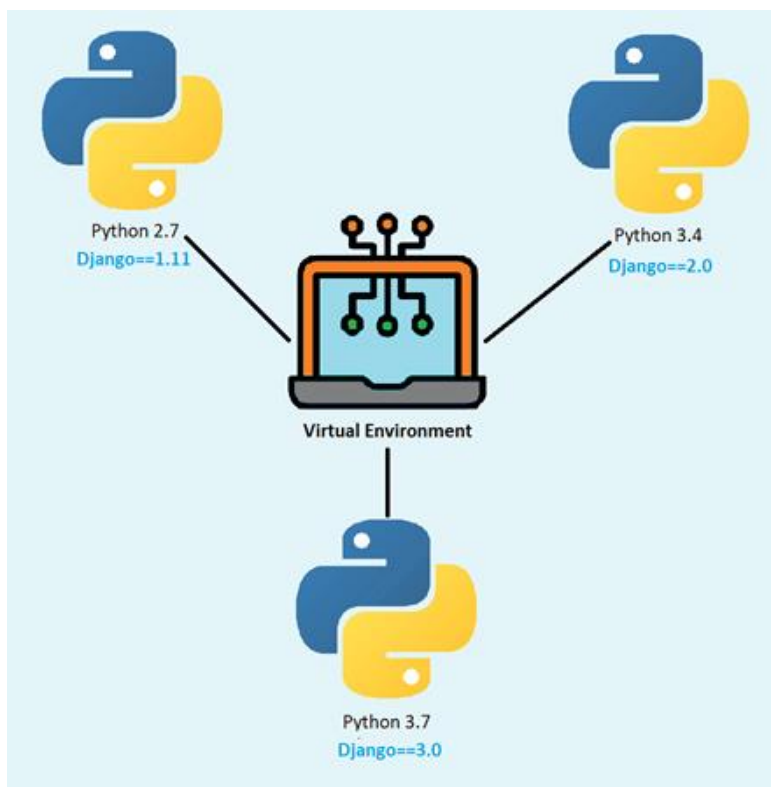
¹⁸ GitHub Desktop

¹⁹ Terminal

²⁰ User Interface (UI)

۲-۲ محیط مجازی پایتون^{۲۱}

محیط مجازی ابزاری است که به کمک آن می‌توان وابستگی^{۲۲}های مورد نیاز پروژه‌های مختلف را با ایجاد محیط مجازی پایتون جدا برای آنها ساده‌تر و مجزا کرد. با ساخت یک محیط مجازی برای پروژه، توسعه دهنده می‌تواند نسخه‌های دلخواه تکنولوژی‌های مورد استفاده را نصب و استفاده کند. [6]



تصویر ۳-۲ نقش محیط مجازی در مدیریت نسخه‌های مختلف

۳-۲ چه زمانی باید از محیط مجازی استفاده کرد؟

یک سناریو را تصور کنید که توسعه دهنده همزمان روی دو پروژه پایتون مبتنی بر وب کار می‌کند و یکی از آنها Django 2.2 و دیگری از Django 3.2 استفاده می‌کند. این موضوع برای پایتون مشکل ایجاد می‌کند زیرا که پایتون نمی‌تواند بین این دو نسخه تفاوت قائل شود.

در چنین شرایطی محیط مجازی می‌تواند برای حفظ وابستگی‌های هر دو پروژه بسیار مفید باشد.

²¹ Python Virtual Environment (env)

²² Dependencies

برای حل این مشکل کفایت دو محیط مجازی مجزا برای هر دو پروژه ایجاد کرد. نکته قابل توجه در این مورد این است که هیچ محدودیتی در تعداد محیط‌هایی که می‌توان ایجاد کرد، وجود ندارد.

۲-۴ نحوه عملکرد محیط مجازی

در این پروژه از ماژول `virtualenv` استفاده شده است که ابزاری برای ایجاد محیط‌های جداگانه پایتون است. نصب این ماژول از اجرای دستور زیر میسر است:

```
pip install virtualenv
```

سپس در پوشه مورد نظر برای ساخت پروژه، دستور زیر اجرا می‌شود:

```
python -m venv Bachelor_Project
```

که `Bachelor_Project` نام محیط مجازی و نام پروژه است.

و در مرحله آخر باید محیط مجازی فعال شود که از دستور زیر استفاده می‌شود:

```
venv\Scripts\activate.bat
```

حالا که محیط مجازی فعال شده است، می‌توان تکنولوژی‌ها و وابستگی‌های مورد نظر پروژه را نصب کرد.

۲-۵ نسخه جنگو مورد استفاده در این پروژه

در این پروژه از نسخه ۲.۲.۵ جنگو استفاده شده است که یک نسخه ^{۲۳}LTS می‌باشد. دلیل استفاده از نسخه های LTS، تضمین طولانی مدت پشتیبانی پروژه می‌باشد.

برای نصب این نسخه از دستور زیر استفاده می‌شود.

```
pip install django==2.2.5
```

دقت شود که هنگام اجرای این دستور، باید محیط مجازی فعال باشد.

²³ Long-Term Supported

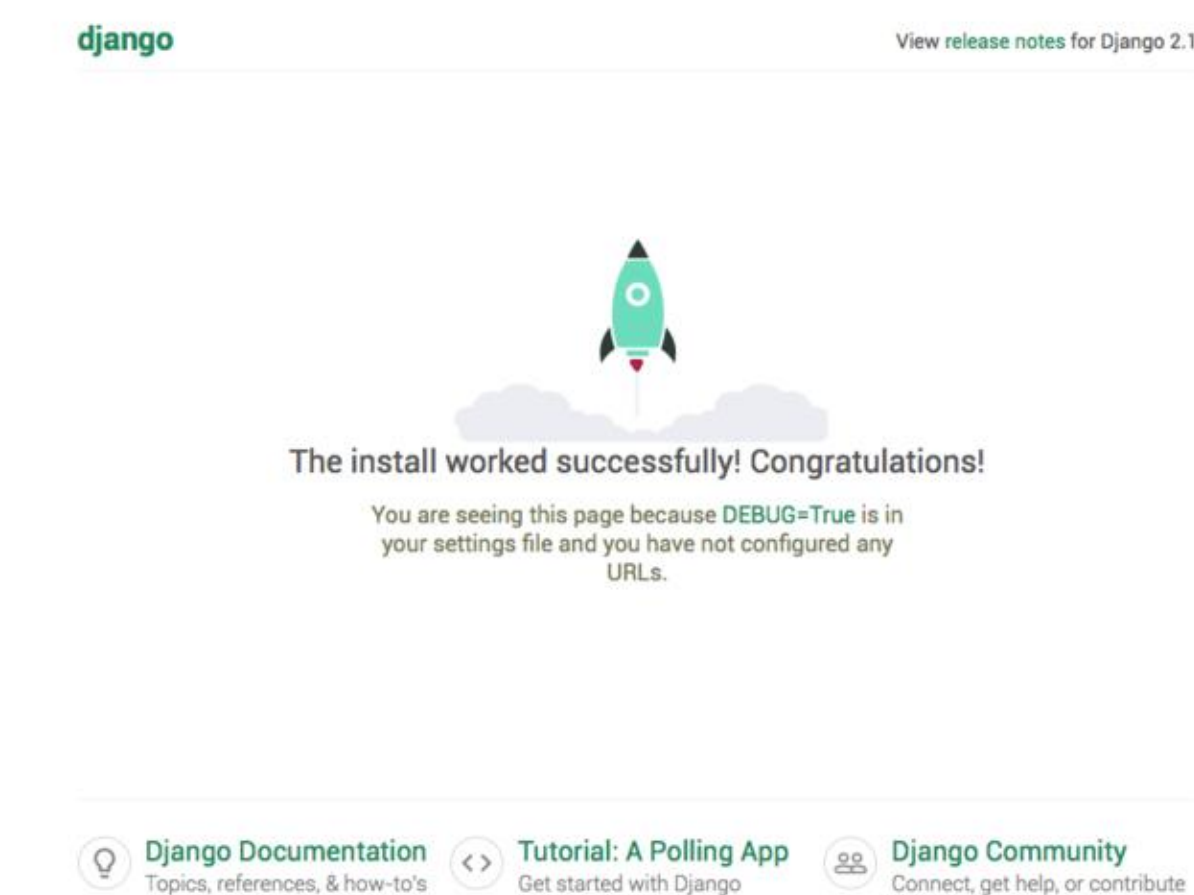
۲-۶ شروع ساخت پروژه جنگو

برای شروع هر پروژه لازم است که ابتدا فایل‌های اصلی پروژه ایجاد شود که این امر با اجرای دستور زیر میسر می‌شود:

```
django-admin startproject Bachelor_Project
```

سپس با اجرای دستور زیر می‌توان اطمینان حاصل کرد که پروژه به درستی ایجاد شده است:

```
python manage.py runserver
```



تصویر ۲-۴ صفحه اصلی وب سایت پس از ساخت موفق پروژه جنگو

۲-۷ فایل‌های ایجاد شده پس از ساخت پروژه

فایل‌ها و پوشه‌های موجود در دایرکتوری^{۲۴} ریشه، اهداف خاص خود را دارند و پس از درک آنها، جنگو بسیار منطقی به نظر می‌رسد.

²⁴ Root Directory

۲-۷-۱ فایل `manage.py`:

این فایل ابزار خط فرمان پروژه است و از این فایل فقط برای استقرار، اشکال زدایی و آزمایش با پروژه استفاده خواهد شد.

۲-۷-۲ فایل `__init__.py`:

این فایل یک فایل خالی است و فقط در پروژه وجود دارد که به مترجم پایتون بفهماند که محتویات این پوشه را نیز در نظر بگیرد. وجود این فایل از قوانین استاندارد بسته‌های پایتون است.

۲-۷-۳ فایل `settings.py`:

همانطور که از نام این فایل مشخص است، این فایل تنظیمات اصلی پروژه جنگو و اطلاعات میان افزارها^{۲۵} را نگهداری می‌کند.

۲-۷-۴ فایل `urls.py`:

این فایل حاوی اطلاعات URL در سطح پروژه است. این فایل یک آدرس یاب کلی در پروژه است و آدرس منابع (تصاویر ، صفحات وب، برنامه‌های وب) را فراهم می‌کند.

۲-۸ ساخت اپلیکیشن

برای اینکه توسعه و نگهداری پروژه‌ها ساده تر باشد، جنگو پروژه را به موجودیت‌هایی کوچکتر به نام App^{۲۶} تقسیم می‌کند. هر App یک وظیفه مخصوص را بر عهده دارد.

^{۲۵} Middlewares

^{۲۶} Application

در این پروژه ۷ اپلیکیشن ساخته شده است:

- conversations
- core
- lists
- reservations
- reviews
- rooms
- users

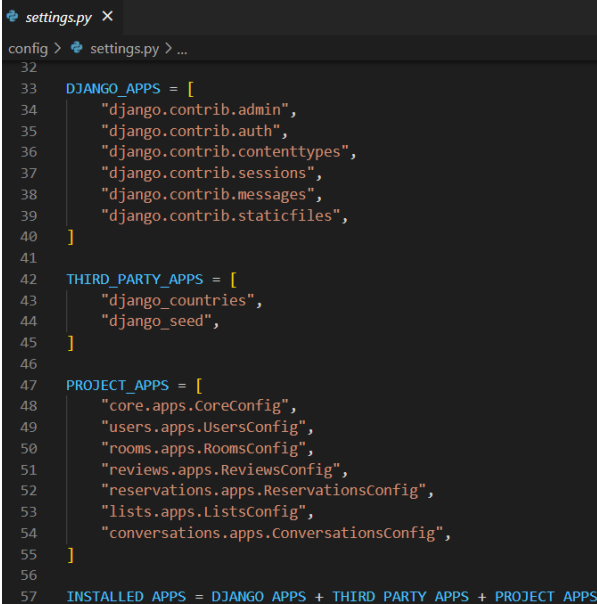
برای ساخت یک اپلیکیشن از دستور زیر استفاده می‌شود:

```
django-admin startapp name
```

در این حالت یک پوشه جدید با نام اپلیکیشن با محتویات زیر ساخته خواهد شد:

```
name/  
  admin.py  
  apps.py  
  models.py  
  tests.py  
  views.py  
  __init__.py  
  migrations/
```

سپس می‌بایست نام اپلیکیشن ساخته شده را در بخش installed apps در فایل settings.py به پروژه اضافه کرد تا در دسترس کل برنامه قرار بگیرد.



```
32  
33 DJANGO_APPS = [  
34     "django.contrib.admin",  
35     "django.contrib.auth",  
36     "django.contrib.contenttypes",  
37     "django.contrib.sessions",  
38     "django.contrib.messages",  
39     "django.contrib.staticfiles",  
40 ]  
41  
42 THIRD_PARTY_APPS = [  
43     "django_countries",  
44     "django_seed",  
45 ]  
46  
47 PROJECT_APPS = [  
48     "core.apps.CoreConfig",  
49     "users.apps.UsersConfig",  
50     "rooms.apps.RoomsConfig",  
51     "reviews.apps.ReviewsConfig",  
52     "reservations.apps.ReservationsConfig",  
53     "lists.apps.ListsConfig",  
54     "conversations.apps.ConversationsConfig",  
55 ]  
56  
57 INSTALLED_APPS = DJANGO_APPS + THIRD_PARTY_APPS + PROJECT_APPS
```

تصویر ۲-۵ اضافه کردن نام پروژه به لیست PROJECT_APPS

۲-۹ ایجاد پایگاه داده و superuser

در ابتدای هر پروژه جنگو که ساخته می‌شود اطلاعاتی وجود دارند که نیاز به نگهداری در پایگاه داده‌ها دارند. برای اینکه این ساختارها شکل بگیرند می‌بایست پردازش اولیه توسط دستور makemigrations صورت بگیرد. این دستور یک فایل اولیه را ایجاد می‌کند که اطلاعات لازم برای شکل‌گیری جداول و فیلدهای آن را شامل می‌شود. و سپس به صورت یک فرمت قابل اجرا برای سیستم جنگو تعریف و در پوشه migrations هر اپلیکیشن نگهداری می‌شود. اما برای اجرا و اعمال تمامی این دستورات می‌بایست تایید نهایی صورت گیرد که با استفاده از دستور migrate انجام می‌شود:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

نکته: ممکن است که اجرای فرمان makemigrations فایل db.sqlite3 را ایجاد کند که حاوی پایگاه داده SQLite پروژه است. هنگامی در نبود پایگاه داده، این دستور اجرا شود، به صورت خودکار پایگاه داده sqlite ایجاد می‌شود. اگر پایگاه داده دیگری مد نظر برنامه نویس باشد، باید ابتدا فایل دیتابیس ایجاد شود، سپس دستور makemigrations را اجرا شود.

جنگو پنل مدیریت (ادمین)^{۲۷} را بصورت پیش‌فرض^{۲۸} در اختیار کاربران خود قرار می‌دهد. بنابراین لازم نیست نگران ایجاد یک صفحه مدیریت جداگانه یا ارائه ویژگی احراز هویت^{۲۹} باشیم. قبل از استفاده از این ویژگی، باید پروژه migrate شده باشد. در غیر این صورت پایگاه داده superuser را ایجاد نمی‌کند.

برای ایجاد superuser دستور زیر باید اجرا شود:

```
python manage.py createsuperuser
```

سپس نام کاربری، ایمیل، رمز عبور و تایید رمز عبور وارد می‌شود. اگر تمامی این فیلدها درست وارد شده باشند، superuser با موفقیت ایجاد می‌شود. در اینصورت، می‌توان با اجرای دستور زیر و ورود به صفحه Admin و وارد کردن مشخصات کاربری، وارد پنل مدیریت شد.

```
python manage.py runserver
```

Admin panel link: <http://127.0.0.1:8000/admin/>

²⁷ Admin Panel

²⁸ Default

²⁹ Authentication

فصل سوم

توسعه مدل ها

همانطور که در فصل قبل اشاره شد، پایه کاری جنگو بر مبنای مدل MVT می باشد. بنابراین در پروژه های جنگو، اکثر وقت برنامه نویس صرف توسعه این سه بخش می شود. در این فصل توسعه مدل ها^{۳۰} بررسی شده است.

۳-۱ فیلد^{۳۱} های مدل جنگو

به زبانی ساده، مدل ها خصوصیات و صفات^{۳۲} یک اپلیکیشن را بیان می کنند. اینکه هر اپلیکیشن چه فیلد هایی دارد و آن فیلد ها چه مقادیری می پذیرند، همه و همه بر عهده مدل ها می باشد. نام های موجود در مدل های جنگو که به صورت attribute های یک کلاس هستند در واقع نام ستون های جداول می باشند که می توانند موارد مختلفی را نگهداری کنند.

جنگو انواع فیلد های زیر را ارائه می دهد:

Field Name	Class	Particular
AutoField	class AutoField(**options)	It An IntegerField that automatically increments.
BigAutoField	class BigAutoField(**options)	It is a 64-bit integer, much like an AutoField except that it is guaranteed to fit numbers from 1 to 9223372036854775807.
BigIntegerField	class BigIntegerField(**options)	It is a 64-bit integer, much like an IntegerField except that it is guaranteed to fit numbers from -9223372036854775808 to 9223372036854775807.
BinaryField	class BinaryField(**options)	A field to store raw binary data.
BooleanField	class BooleanField(**options)	A true/false field. The default form widget for this field is a CheckboxInput.
CharField	class DateField(auto_now=False, auto_now_add=False, **options)	It is a date, represented in Python by a datetime.date instance.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **options)	It is a date, represented in Python by a datetime.date instance.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **options)	It is used for date and time, represented in Python by a datetime.datetime instance.
DecimalField	class DecimalField(max_digits=None, decimal_places=None, **options)	It is a fixed-precision decimal number, represented in Python by a Decimal instance.
DurationField	class DurationField(**options)	A field for storing periods of time.
EmailField	class EmailField(max_length=254, **options)	It is a CharField that checks that the value is a valid email address.
FileField	class FileField(upload_to=None, max_length=100, **options)	It is a file-upload field.
FloatField	class FloatField(**options)	It is a floating-point number represented in Python by a float instance.
ImageField	class ImageField(upload_to=None, height_field=None, width_field=None, max_length=100, **options)	It inherits all attributes and methods from FileField, but also validates that the uploaded object is a valid image.
IntegerField	class IntegerField(**options)	It is an integer field. Values from -2147483648 to 2147483647 are safe in all databases supported by Django.
NullBooleanField	class NullBooleanField(**options)	Like a BooleanField, but allows NULL as one of the options.
PositiveIntegerField	class PositiveIntegerField(**options)	Like an IntegerField, but must be either positive or zero (0). Values from 0 to 2147483647 are safe in all databases supported by Django.
SmallIntegerField	class SmallIntegerField(**options)	It is like an IntegerField, but only allows values under a certain (database-dependent) point.
TextField	class TextField(**options)	A large text field. The default form widget for this field is a Textarea.
TimeField	class TimeField(auto_now=False, auto_now_add=False, **options)	A time, represented in Python by a datetime.time instance.

تصویر ۳-۱ انواع فیلد ها در مدل جنگو

³¹ Field

³² Attributes

هر فیلد به آرگومان^{۳۳} هایی نیاز دارد که برای تنظیم ویژگی های ستون استفاده می شود. برای مثال ، CharField ، برای تعیین پایگاه داده varchar به max_length نیاز دارد.

جدول زیر نمایانگر آپشن های موجود برای فیلد ها است:

Field Options	Particulars
Null	Django will store empty values as NULL in the database.
Blank	It is used to allowed field to be blank.
Choices	An iterable (e.g., a list or tuple) of 2-tuples to use as choices for this field.
Default	The default value for the field. This can be a value or a callable object.
help_text	Extra "help" text to be displayed with the form widget. It's useful for documentation even if your field isn't used on a form.
primary_key	This field is the primary key for the model.
Unique	This field must be unique throughout the table.

تصویر ۳-۲ / آپشن های موجود برای فیلد های مدل

۳-۲ رابطه ها در جداول

در جداول بین موجودیت های مختلف - یا به طور دقیق تر در جنگو - بین مدل های مختلف ممکن است رابطه ای وجود داشته باشد. روابط به سه نوع وجود دارند:

۱. کلید خارجی^{۳۴}

۲. یک به یک^{۳۵}

۳. چند به چند^{۳۶}

۳-۳ مدل Meta

Model Meta اساساً برای تغییر رفتار زمینه های مدل مانند تغییر یا ایجاد گزینه های سفارشی استفاده می شود. لازم به ذکر است که افزودن کلاس Meta به مدل کاملاً اختیاری است و فقط به تعدادی از مدل های این پروژه، کلاس متا اضافه شده است.

³³ Argument

³⁴ Foreign Key

³⁵ One to One

³⁶ Many to Many

۳-۴ توسعه مدل ها از طریق فایل models.py

برای به پایان رساندن موفق این پروژه، در وهله اول لازم است که فایل models.py هر هفت اپلیکیشن موجود در پروژه، توسعه داده شوند.

لازم به ذکر است در پایان توسعه هر فایل models.py باید دو دستور زیر اجرا شوند تا فیلدهای جدید یا تغییرات ایجاد شده در آن ها در پایگاه داده ذخیره شوند:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

۳-۴-۱ توسعه مدل users:

این مدل شامل اطلاعات لازم برای موجودیت کاربران است. برای ساخت این مدل از مدل AbstractUser که مربوط به خود جنگو می باشد استفاده شده است. دلیل این کار، بهره بردن از احراز کننده نام کاربری، رمز عبور و ایمیل می باشد.

لازم به ذکر است که خود جنگو پس از ساخت پروژه، مدل یوزر را اضافه می کند که شامل صفات محدودی از کاربران است. اما برای این پروژه نیاز به توسعه بیشتر این مدل می باشد.

خصوصیات ای که به این مدل اضافه می شود به شرح زیر است:

- جنسیت
- زبان
- لاگین از طریق ایمیل یا اکانت گیت هاب
- نام
- عکس پروفایل
- بیوگرافی
- تاریخ تولد
- میزبان نمونه بودن

که نتیجه نهایی بصورت زیر خواهد بود:

```

models.py X
users > models.py > ...
1 from django.utils.translation import gettext_lazy as _
2 from django.db import models
3 from django.contrib.auth.models import AbstractUser
4 from django.urls import reverse
5
6
7 class User(AbstractUser):
8
9     """
10     Custom User Model
11     """
12
13     GENDER_MALE = "Male"
14     GENDER_FEMALE = "Female"
15     GENDER_OTHER = "Other"
16
17     GENDER_CHOICES = (
18         (GENDER_MALE, _("Male")),
19         (GENDER_FEMALE, _("Female")),
20         (GENDER_OTHER, _("Other")),
21     )
22
23     LANGUAGE_ENGLISH = "en"
24     LANGUAGE_Farsi = "fa"
25
26     LANGUAGE_CHOICES = (
27         (LANGUAGE_ENGLISH, _("English")),
28         (LANGUAGE_Farsi, _("Farsi")),
29     )
30
31     LOGIN_EMAIL = "email"
32     LOGIN_GITHUB = "github"
33
34     LOGIN_CHOICES = (
35         (LOGIN_EMAIL, _("Email")),
36         (LOGIN_GITHUB, _("Github")),
37     )
38
39     first_name = models.CharField(_("first name"), max_length=30, blank=True)
40
41     avatar = models.ImageField(_("avatar"), upload_to="avatars", blank=True)
42     gender = models.CharField(
43         _("gender"), choices=GENDER_CHOICES, max_length=10, blank=True
44     )
45     bio = models.TextField(_("bio"), blank=True)
46     birthdate = models.DateField(_("birthdate"), blank=True, null=True)
47     language = models.CharField(
48         _("language"),
49         choices=LANGUAGE_CHOICES,
50         max_length=2,
51         blank=True,
52         default=LANGUAGE_Farsi,
53     )
54
55     balance = models.IntegerField(_("balance"), default=100000)
56     superhost = models.BooleanField(_("superhost"), default=False)
57     email_verified = models.BooleanField(default=False)
58     email_secret = models.CharField(max_length=120, default="", blank=True)
59     login_method = models.CharField(
60         max_length=50, choices=LOGIN_CHOICES, default=LOGIN_EMAIL
61     )

```

تصویر ۳-۳ مدل users

۳-۴-۲ توسعه مدل rooms و core:

قبل از اینکه مدل اتاق ها توسعه داده شود، ابتدا باید مدل ای در اپلیکیشن core (هسته) ساخته شود که توانایی نگهداری تاریخ و ساعت object های ساخته شده را داشته باشد. سپس در فایل ای جداگانه به نام managers.py در اپلیکیشن هسته، مدل ای ساخته می شود که توانایی بازیابی مشخصات object های ساخته شده را داشته باشد. بنابراین ابتدا فایل managers.py در این اپلیکیشن ایجاد شده و صورت زیر مدل مربوطه ایجاد می شود:

```
managers.py X
core > managers.py > ...
1  from django.db import models
2  from django.contrib.auth.models import UserManager
3
4
5  class CustomModelManager(models.Manager):
6      def get_or_none(self, **kwargs):
7          try:
8              return self.get(**kwargs)
9          except self.model.DoesNotExist:
10             return None
11
12
13  class CustomUserManager(CustomModelManager, UserManager):
14      pass
```

تصویر ۳-۴ فایل managers.py

سپس فایل models.py اپلیکیشن core توسعه داده می شود. این مدل به صورت زیر خواهد بود:

```
models.py X
core > models.py > ...
1  from django.db import models
2  from . import managers
3
4
5  class TimeStampedModel(models.Model):
6
7      created = models.DateTimeField(auto_now_add=True)
8      updated = models.DateTimeField(auto_now=True)
9      objects = managers.CustomModelManager()
10
11      class Meta:
12          abstract = True
```

تصویر ۳-۵ مدل core

حال مدل اتاق ها را توسعه خواهیم داد. این مدل شامل اطلاعات لازم برای موجودیت اتاق ها است.

ابتدا مدل ای به نام `AbstractItem` می‌سازیم که از مدل `TimeStampedModel` اپلیکیشن `core` ارث^{۳۷} می‌برد و فقط صفت نام را دارد و با استفاده از کلاس متا، صفت انتزاعی^{۳۸} بودن آن را معادل `True` قرار می‌دهیم.

سپس چهار کلاس برای نوع اتاق، امکانات رفاهی^{۳۹}، تسهیلات^{۴۰} و قوانین اتاق^{۴۱} می‌سازیم.

لازم به ذکر است که هر چهار کلاس از مدل `AbstractItem` ارث بری می‌کنند و در کلاس متای آنها، نحوه نمایش نامشان در حالت های مفرد یا جمع زبان انگلیسی، سفارشی سازی شده است.

پس از پیاده سازی این چهار کلاس، آن ها را در مدل `Room` بصورت صفت، فراخوانی می‌کنیم.

سپس کلاس `Photo` ساخته می‌شود که شامل صفات فایل، کپشن^{۴۲} و اتاق می‌باشد و در صورت فراخوانی بصورت متنی، متن کپشن برگشت داده خواهد شد.

لازم با ذکر است که نوع صفت اتاق از نوع رابطه کلید خارجی است.

حال می‌توان مدل اتاق را براساس کلاس هایی که تا اینجا ساخته شدند، بسازیم. این مدل شامل صفات اتاق و تعدادی تابع برای انجام پردازش ها است.

صفات مدل اتاق به شرح زیر است:

- نام
- توضیحات
- شهر
- قیمت
- آدرس
- تعداد مهمان های مجاز
- تعداد اتاق خواب
- تعداد تخت خواب
- تعداد سرویس بهداشتی

³⁷ Inheritance

³⁸ Abstract

³⁹ Amenities

⁴⁰ Facilities

⁴¹ House Rules

⁴² Caption

- ساعت ورود
- ساعت خروج
- قابلیت رزرو آنی
- میزبان (کلید خارجی به کاربر)
- نوع اتاق
- امکانات رفاهی
- تسهیلات
- قوانین خانه

که نتیجه نهایی به صورت زیر است:

```

69 class Room(core_models.TimeStampedModel):
70
71     """Room Model Definition"""
72
73     name = models.CharField(_("name"), max_length=140)
74     description = models.TextField(
75         _("description"),
76     )
77     country = CountryField(
78         _("country"),
79     )
80     city = models.CharField(_("city"), max_length=80)
81     price = models.IntegerField(
82         _("price"),
83     )
84     address = models.CharField(_("address"), max_length=140)
85     guests = models.IntegerField(
86         _("guests"), help_text=_("How many people will be staying?")
87     )
88     beds = models.IntegerField(
89         _("beds"),
90     )
91     bedrooms = models.IntegerField(
92         _("bedrooms"),
93     )
94     baths = models.IntegerField(
95         _("baths"),
96     )
97     check_in = models.TimeField(
98         _("check_in"),
99     )
100     check_out = models.TimeField(
101         _("check_out"),
102     )
103     instant_book = models.BooleanField(_("instant_book"), default=False)
104     host = models.ForeignKey(
105         "users.User", related_name="rooms", on_delete=models.CASCADE
106     )
107     room_type = models.ForeignKey(
108         "RoomType", related_name="rooms", on_delete=models.SET_NULL, null=True
109     )
110     amenities = models.ManyToManyField("Amenity", related_name="rooms", blank=True)
111     facilities = models.ManyToManyField("Facility", related_name="rooms", blank=True)
112     house_rules = models.ManyToManyField("HouseRule", related_name="rooms", blank=True)
113

```

تصویر ۳-۶ مدل rooms

ابتدا تابع ای برای برگرداندن نام اتاق در صورت فراخوانی اتاق بصورت متنی، می‌نویسیم. سپس برای رعایت قوانین نگارشی زبان انگلیسی، تابع ای برای بزرگ کردن حرف اول^{۴۳} نام شهر مربوط به اتاق نوشته می‌شود که پس از انجام این دستور، آن را در پایگاه داده ذخیره می‌کند.

آنگاه تابع ای برای محاسبه امتیاز اتاق نوشته می‌شود که عمل ریاضی میانگین‌گیری^{۴۴} را انجام می‌دهد. سپس دو تابع برای نمایش عکس‌های مربوط به اتاق نوشته می‌شود که تابع اول، عکس اول اتاق را در صفحه اصلی وب‌سایت نمایش می‌دهد و تابع دوم، چهار عکس از اتاق در صفحه مربوط به اتاق را نمایش می‌دهد.

و در پایان تابع ای برای نمایش تقویم، جهت رزرو اتاق نوشته می‌شود.

```
114 def __str__(self):
115     return self.name
116
117 def save(self, *args, **kwargs):
118     self.city = str.capitalize(self.city)
119     super().save(*args, **kwargs)
120
121 def get_absolute_url(self):
122     return reverse("rooms:detail", kwargs={"pk": self.pk})
123
124 def total_rating(self):
125     all_reviews = self.reviews.all() # type: ignore
126     all_ratings = 0
127     if len(all_reviews) > 0:
128         for review in all_reviews:
129             all_ratings += review.rating_average()
130     return round(all_ratings / len(all_reviews), 2)
131     return 0
132
133 def first_photo(self):
134     try:
135         (photo,) = self.photos.all()[1:] # type: ignore
136         return photo.file.url
137     except ValueError:
138         return None
139
140 def get_next_four_photos(self):
141     photos = self.photos.all()[1:5] # type: ignore
142     return photos
143
144 def get_calendars(self):
145     now = timezone.now()
146     this_year = now.year
147     this_month = now.month
148     next_month = this_month + 1
149     if this_month == 12:
150         next_month = 1
151     this_month_cal = Calendar(this_year, this_month)
152     next_month_cal = Calendar(this_year, next_month)
153     return [this_month_cal, next_month_cal]
```

تصویر ۳-۷ توابع مدل *rooms*

⁴³ Capitalize

⁴⁴ Average

۳-۴-۳ توسعه مدل reviews

این مدل شامل اطلاعات لازم برای موجودیت نقد و بررسی است. برای ساخت این مدل از مدل TimeStampedModel ارث بری شده است. ابتدا صفات تعیین شده اند و سپس توابع ای برای پردازش های مورد نظر نوشته شده اند.

صفات این مدل شامل موارد زیر است:

- متن نقد
- امتیاز شبیه بودن اتاق نسبت به اطلاعات داده شده در وبسایت
- امتیاز ارتباطات
- امتیاز تمیزی
- امتیاز موقعیت مکانی
- امتیاز رزرو کردن راحت
- امتیاز ارزشمند بودن اتاق
- کاربر (کلید خارجی)
- اتاق (کلید خارجی)

ابتدا تابع ای برای فراخوانی نقد به صورت متن نوشته می شود به این صورت که ابتدا اسم اتاق نوشته شود و بعد از آن نقد نوشته شده نمایش داده شود.

سپس تابع ای برای محاسبه و میانگین گیری امتیاز نقد و بررسی نوشته می شود و جهت جلوگیری از نمایش بیش از حد ارقام بعد از ممیز، تعداد ارقام بعد از ممیز به ۲ رقم محدود می شود.

در پایان در کلاس متا مربوط به این مدل، تعیین می شود که به ترتیب تاریخ ایجاد نقد و بررسی، این نقد ها در وبسایت نمایش داده شوند.

نتیجه نهایی به صورت زیر است:

```

models.py X
reviews > models.py > Review > __str__
1  from django.db import models
2  from django.core.validators import MinValueValidator, MaxValueValidator
3  from django.utils.translation import gettext_lazy as _
4  from core import models as core_models
5
6
7  class Review(core_models.TimeStampedModel):
8
9      review = models.TextField(
10         _("review"),
11     )
12     accuracy = models.IntegerField(
13         _("accuracy"), validators=[MinValueValidator(1), MaxValueValidator(5)]
14     )
15     communication = models.IntegerField(
16         _("communication"), validators=[MinValueValidator(1), MaxValueValidator(5)]
17     )
18     cleanliness = models.IntegerField(
19         _("cleanliness"), validators=[MinValueValidator(1), MaxValueValidator(5)]
20     )
21     location = models.IntegerField(
22         _("location"), validators=[MinValueValidator(1), MaxValueValidator(5)]
23     )
24     check_in = models.IntegerField(
25         _("check_in"), validators=[MinValueValidator(1), MaxValueValidator(5)]
26     )
27     value = models.IntegerField(
28         _("value"), validators=[MinValueValidator(1), MaxValueValidator(5)]
29     )
30     user = models.ForeignKey(
31         "users.User", related_name="reviews", on_delete=models.CASCADE
32     )
33     room = models.ForeignKey(
34         "rooms.Room", related_name="reviews", on_delete=models.CASCADE
35     )

```

تصویر ۳-۸ مدل reviews

```

37  def __str__(self):
38      return f"{self.room}: ' {self.review} '"
39
40  def rating_average(self):
41
42      avg = (
43          self.accuracy
44          + self.communication
45          + self.cleanliness
46          + self.location
47          + self.check_in
48          + self.value
49      ) / 6
50      return round(avg, 2)
51
52      rating_average.short_description = _("Avg.")
53
54  class Meta:
55      ordering = ("-created",)
56

```

تصویر ۳-۹ توابع مدل reviews

۳-۴-۴ توسعه مدل reservation:

این مدل شامل اطلاعات لازم برای موجودیت رزرواسیون است. برای ساخت این مدل از مدل TimeStampedModel ارث بری شده است.

کلاس BookedDay ماهیت روز رزرو شده را پیاده سازی می کند که شامل صفات روز و رزرواسیون (کلید خارجی) است. در کلاس متای آن نیز نوع نوشتار حالت مفرد و جمع BookedDay تعیین می شود و در صورت فراخوانی بصورت متن، روز رزرواسیون برگردانده می شود.

در مدل اصلی رزرواسیون، صفات زیر پیاده سازی شده اند:

- وضعیت رزرواسیون : تایید شده، کنسل شده یا در دست بررسی است؟
- تاریخ ورود
- تاریخ خروج
- مهمان (کلید خارجی به کاربر)
- اتاق مورد نظر (کلید خارجی)

نتیجه بصورت زیر خواهد بود:

```
21 class Reservation(core_models.TimeStampedModel):
22
23     STATUS_PENDING = "pending"
24     STATUS_CONFIRMED = "confirmed"
25     STATUS_CANCELED = "canceled"
26
27     STATUS_CHOICES = (
28         (STATUS_PENDING, _("pending")),
29         (STATUS_CONFIRMED, _("confirmed")),
30         (STATUS_CANCELED, _("canceled")),
31     )
32
33     status = models.CharField(
34         _("status"), max_length=12, choices=STATUS_CHOICES, default=STATUS_PENDING
35     )
36     check_in = models.DateField(_("check_in"))
37     check_out = models.DateField(_("check_out"))
38     guest = models.ForeignKey(
39         "users.User", related_name="reservations", on_delete=models.CASCADE
40     )
41     room = models.ForeignKey(
42         "rooms.Room", related_name="reservations", on_delete=models.CASCADE
43     )
```

تصویر ۳-۱۰ مدل reservations

در ابتدا تابع بازگردانی متن را بصورت نمایش ابتدا نام اتاق و سپس تاریخ ورود تعیین می‌کنیم.

سپس دو تابع برای زمانیکه رزرواسیون تایید شد می‌نویسیم. تابع اول مربوط به حالت ای است که مهمان در حال اقامت در اتاق است و تابع دوم مربوط به حالت ای است که دوران رزرواسیون به اتمام رسیده است.

و در پایان، روز های رزرو شده را در پایگاه داده ذخیره می‌کنیم.

نتیجه به صورت زیر خواهد بود:

```
45 def __str__(self):
46     return f"{self.room} - {self.check_in} "
47
48 def in_progress(self):
49     now = timezone.now().date()
50     return now >= self.check_in and now <= self.check_out
51
52 in_progress.boolean = True
53
54 def is_finished(self):
55     now = timezone.now().date()
56     is_finished = now > self.check_out
57     if is_finished:
58         BookedDay.objects.filter(reservation=self).delete()
59     return is_finished
60
61 is_finished.boolean = True
62
63 def save(self, *args, **kwargs):
64     if self.pk is None:
65         start = self.check_in
66         end = self.check_out
67         difference = end - start
68         existing_booked_day = BookedDay.objects.filter(
69             day__range=(start, end)
70         ).exists()
71         if not existing_booked_day:
72             super().save(*args, **kwargs)
73             for i in range(difference.days + 1): # type: ignore
74                 day = start + datetime.timedelta(days=i)
75                 BookedDay.objects.create(day=day, reservation=self)
76             return
77     return super().save(*args, **kwargs)
78
```

تصویر ۳-۱۱) توابع مدل *reservations*

۳-۴-۵ توسعه مدل lists:

این مدل شامل اطلاعات اتاق ها و کاربران مربوط به آن است که در ادامه برای ساخت لیست اتاق های مورد علاقه از آن استفاده خواهد شد. برای ساخت این مدل از مدل TimeStampedModel ارث بری شده است.

در ابتدا مدل به همراه صفات نام، کاربر (رابطه یک به یک) و اتاق ها (رابطه چند به چند) ایجاد شده سپس دو تابع بازگردانی نام لیست و تابع شمارنده تعداد اتاق ها نوشته می شود.

نتیجه نهایی به این صورت خواهد بود:

```
models.py
lists > models.py > ...
1 from django.db import models
2 from django.utils.translation import gettext_lazy as _
3 from core import models as core_model
4
5
6 class List(core_model.TimeStampedModel):
7
8     name = models.CharField(max_length=80)
9     user = models.OneToOneField(
10         "users.User", related_name="list", on_delete=models.CASCADE)
11
12     rooms = models.ManyToManyField("rooms.Room", related_name="lists", blank=True)
13
14     def __str__(self):
15         return self.name
16
17     def count_rooms(self):
18         return self.rooms.count()
19
20     count_rooms.short_description = _("Number of Rooms")
21
```

تصویر ۳-۱۲ مدل lists

۳-۴-۶ توسعه مدل conversation:

این مدل شامل پیام های رد و بدل شده بین اعضای سایت خواهد بود. برای ساخت این مدل از مدل TimeStampedModel ارث بری شده است.

ابتدا کلاس conversation نوشته می شود که شامل صفت مشارکت کنندگان است. دو تابع برای شمارش تعداد پیام و تعداد مشارکت کنندگان نوشته می شود و یک تابع هم برای بازگردانی متنی این مدل که شامل نام مشارکت کنندگان خواهد بود.

سپس کلاس Message نوشته می‌شود که شامل صفات متن پیام، کاربر (کلید خارجی) و گفتگو (کلید خارجی) است و یک تابع هم برای بازگردانی کلاس بصورت متنی دارد.

نتیجه نهایی بدین صورت خواهد بود:

```
6 class Conversation(core_models.TimeStampedModel):
7
8     participants = models.ManyToManyField(
9         "users.User", related_name="conversation", blank=True
10    )
11
12    def __str__(self):
13        usernames = []
14        for user in self.participants.all():
15            usernames.append(user.username)
16        return ", ".join(usernames)
17
18    def count_messages(self):
19        return self.messages.count() # type: ignore
20
21    count_messages.short_description = _("Number of messages")
22
23    def count_participants(self):
24        return self.participants.count()
25
26    count_participants.short_description = _("Number of participants")
27
28
29 class Message(core_models.TimeStampedModel):
30     message = models.TextField()
31     user = models.ForeignKey(
32         "users.User", related_name="messages", on_delete=models.CASCADE
33     )
34     conversation = models.ForeignKey(
35         "Conversation", related_name="messages", on_delete=models.CASCADE
36     )
37
38    def __str__(self):
39        return f"{self.user} says: {self.message}"
40
```

تصویر ۳-۱۳ مدل conversations

۳-۵ اضافه کردن اطلاعات مدل ها به ادمین

پس از ساخت مدل ها باید اطلاعاتی که قصد داریم در پنل ادمین به نمایش درآید را تعیین کنیم.

fieldsets شامل فیلدهایی است که به مقدار آنها در پنل کاربری دسترسی داریم و می توانیم آن ها را مقداردهی کنیم یا تغییر بدهیم.

list_filter همانطور که از نامش مشخص است، پارامترهای نمایش داده شده هنگام مشاهده لیست داده ها را مشخص می کند.

جهت پرهیز از زیاده گوئی، یک نمونه از فایل admin.py مربوط به اپلیکیشن users در زیر آورده شده است:

```
admin.py X
users > admin.py > ...
6 @admin.register(models.User)
7 class CustomUserAdmin(UserAdmin):
8
9     """Custom User Admin"""
10
11     fieldsets = UserAdmin.fieldsets + (
12         (
13             "Custom Profile",
14             {
15                 "fields": (
16                     "avatar",
17                     "gender",
18                     "bio",
19                     "birthdate",
20                     "language",
21                     "superhost",
22                     "balance",
23                     "login_method",
24                 )
25             },
26         ),
27     ) # type: ignore
28
29     list_filter = UserAdmin.list_filter + ("superhost",)
30
31     list_display = (
32         "username",
33         "first_name",
34         "last_name",
35         "email",
36         "is_active",
37         "language",
38         "superhost",
39         "is_staff",
40         "is_superuser",
41         "balance",
42     )
```

تصویر ۳-۱۴ فایل ادمین users

فصل چهارم

توسعه View ها و URL ها

در این فصل توسعه View و URL ها بررسی شده است. View، URL و Template ها مسئولیت نمایش محتویات وبسایت و صفحات مختلف آن را بر عهده دارند.

۴-۱ توسعه ویو از طریق فایل `views.py`

این بخش مسئول پردازش درخواست‌ها است. ویو رابطی است که بخش‌های `model` و `template` را به هم وصل میکند. درخواست‌های کاربران در این بخش پردازش شده و پاسخ مناسب به آن‌ها نشان داده خواهد شد.

ارتباط این سه بخش به این صورت است که در `views.py` عملکرد لازم پیاده‌سازی می‌شود سپس یک آدرس مخصوص این عملکرد در فایل `urls.py` ثبت می‌شود و در پایان از طریق فایل `template` مربوطه که آدرسش در `views.py` ثبت شده، خروجی مد نظر را به کمک تکنولوژی‌های فرانت‌اند، به کاربر نمایش می‌دهیم.

لازم به ذکر است که بعلت طولانی بودن تعداد خطوط کد‌های مربوط به فایل‌های `views.py`، تصاویر کد‌های پیاده‌سازی شده‌ای که در ادامه خواهید دید، فقط بخشی از کد اصلی را شامل می‌شوند. تمام کد‌ها و جزئیات پیاده‌سازی در ریپازیتوری گیت‌هاب پروژه در دسترس است.

۴-۱-۱ توسعه ویو `users`:

عملکرد‌های پیاده‌سازی شده در این بخش شامل موارد زیر می‌باشد:

- ورود به سیستم (لاگین)
- خروج از سیستم (لاگ‌اوت)
- ثبت نام در سایت
- ورود با اکانت گیت‌هاب
- مشاهده پروفایل
- تغییر مشخصات پروفایل
- تغییر کلمه عبور
- تغییر حالت میزبانی (میزبان بودن یا نبودن)
- تغییر زبان وب‌سایت (فارسی یا انگلیسی)

بعنوان مثال، اولین کلاس پیاده‌سازی شده، مربوط به عملکرد ورود به سیستم است.

در ابتدا آدرس تمپلیت مربوط به این کلاس را مشخص می‌کنیم. سپس مشخص می‌کنیم که عملکرد مورد نظر باید بصورت فرم باشد.

سپس تابع ای تعریف می‌کنیم که تایید کند فرم مربوطه معتبر^{۴۵} است یا به عبارتی دیگر، ورودی های کاربر صحیح است و احراز هویت به درستی صورت گرفته است.

سپس تابع ای تعریف می‌کنیم که اگر کاربر قرار بود از طریق صفحه لاگین^{۴۶} به صفحه خاص ای منتقل^{۴۷} شود، این انتقال صورت بگیرد، در غیر اینصورت به صفحه اصلی وبسایت هدایت شود.

نتیجه نهایی به صورت زیر خواهد بود:

```
17 class LoginView(mixins.LoggedOutOnlyView, FormView):
18
19     template_name = "users/login.html"
20     form_class = forms.LoginForm
21
22     def form_valid(self, form):
23         email = form.cleaned_data.get("email")
24         password = form.cleaned_data.get("password")
25         user = authenticate(self.request, username=email, password=password)
26         if user is not None:
27             login(self.request, user)
28             return super().form_valid(form)
29
30     def get_success_url(self):
31         next_arg = self.request.GET.get("next")
32         if next_arg is not None:
33             return next_arg
34         else:
35             return reverse("core:home")
```

تصویر ۴-۱ ویو *users*

۴-۱-۲ توسعه ویو *rooms*:

عملکرد های پیاده سازی شده در این بخش شامل موارد زیر می‌باشد:

- صفحه اصلی وبسایت
- مشخصات اتاق ها
- جستجو بین اتاق ها
- تغییر مشخصات اتاق ها
- مشخصات عکس های مربوط به اتاق

⁴⁵ Validator

⁴⁶ Login

⁴⁷ Redirect

- تابع حذف عکس های اتاق ها
- تغییر عکس اتاق ها
- اضافه کردن عکس به اتاق ها
- ساخت اتاق جدید

بعنوان مثال، آخرین کلاس پیاده سازی شده، مربوط به ساخت اتاق جدید است.

در ابتدا مشخص می کنیم که محتویات این کلاس مربوط به کدام فرم است (فرم در فصل ۶ بررسی خواهد شد) سپس آدرس تمپلیت مربوط به این کلاس را مشخص می کنیم.

آنگاه تابع ای تعریف می کنیم که تایید کند فرم مربوطه معتبر است به این صورت که اتاق ساخته شده در پایگاه داده ذخیره می شود و میزبان اتاق معادل کاربر فعلی در نظر گرفته می شود.

در پایان نیز به کاربر پیام "اتاق ساخته شد" نمایش داده می شود و کاربر به صفحه اتاق جدید هدایت می شود.

نتیجه نهایی به صورت زیر خواهد بود:

```

152 class CreateRoomView(user_mixins.LoggedInView, FormView):
153
154     form_class = forms.CreateRoomForm
155     template_name = "rooms/room_create.html"
156
157     def form_valid(self, form):
158         room = form.save() # type: ignore
159         room.host = self.request.user # type: ignore
160         room.save()
161         form.save_m2m() # type: ignore
162         messages.success(self.request, _("Room Uploaded"))
163         return redirect(reverse("rooms:detail", kwargs={"pk": room.pk}))
164

```

تصویر ۴-۲ ویو rooms

۴-۱-۳ توسعه ویو reviews:

تابع ساخت یک نقد و بررسی در اینجا پیاده سازی خواهد شد.

به این صورت که نوع متد از نوع POST در نظر گرفته می شود. سپس فرم و اتاق مربوطه فراخوانی شده و اگر اتاق وجود نداشت کاربر به صفحه اصلی هدایت خواهد شد. در پایان، اگر فرم تایید بشود، نقد کاربر ذخیره شده کاربر به صفحه اتاق مربوطه هدایت می شود و پیام "اتاق بررسی شد" به کاربر نمایش داده می شود.

پیاده سازی به این صورت خواهد بود:

```
1 from django.contrib import messages
2 from django.utils.translation import gettext_lazy as _
3 from django.urls import reverse
4 from django.shortcuts import redirect
5 from rooms import models as room_models
6 from . import forms
7
8
9 def create_review(request, room):
10     if request.method == "POST":
11         form = forms.CreateReviewForm(request.POST)
12         room = room_models.Room.objects.get_or_none(pk=room)
13         if not room:
14             return redirect(reverse("core:home"))
15         if form.is_valid():
16             review = form.save()
17             review.room = room
18             review.user = request.user
19             review.save()
20             messages.success(request, _("Room reviewed"))
21             return redirect(reverse("rooms:detail", kwargs={"pk": room.pk}))
```

تصویر ۴-۳ ویو *reviews*

۴-۱-۴ توسعه ویو *reservations*:

سه عملکرد در این ویو پیاده سازی خواهد شد:

- ساخت رزرواسیون
- مشاهده جزئیات رزرواسیون
- تغییر رزرواسیون

بعنوان مثال، دومین کلاس پیاده سازی شده، مربوط به مشاهده جزئیات رزرواسیون است.

در ابتدا کلید اصلی^{۴۸} درخواست مربوطه را مشخص می‌کنیم و در پایگاه داده آن را جستجو می‌کنیم.

اگر چنین رزرواسیون ای وجود نداشت، درخواست غیر معتبر است و خطای ۴۰۴^{۴۹} نمایش داده خواهد شد. در غیر اینصورت، فرم مربوطه فراخوانی خواهد شد و اطلاعات مربوطه نمایش داده خواهد شد.

نتیجه نهایی به صورت زیر خواهد بود:

⁴⁸ Primary Key (PK)

⁴⁹ 404 Error

```

38 class ReservationDetailView(View):
39     def get(self, *args, **kwargs):
40         pk = kwargs.get("pk")
41         reservation = models.Reservation.objects.get_or_none(pk=pk)
42         if not reservation or (
43             reservation.guest != self.request.user
44             and reservation.room.host != self.request.user
45         ):
46             raise Http404()
47         form = review_forms.CreateReviewForm()
48         return render(
49             self.request,
50             "reservations/detail.html",
51             {"reservation": reservation, "form": form},
52         )

```

تصویر ۴-۴ ویو reservations

۴-۱-۵ توسعه ویو lists:

یک تابع اضافه کردن یا حذف کردن اتاق مورد علاقه و یک کلاس مشاهده لیست اتاق های مورد علاقه در این جا پیاده سازی خواهد شد. در ابتدا درخواست کاربر را در قالب متغیر action دریافت می کنیم و اتاق مربوطه را نیز فرا می خوانیم. اگر اتاق و درخواست معتبر بودند، لیست ای با عنوان اتاق های مورد علاقه من ساخته خواهد شد. سپس اگر درخواست کاربر مبنی بر اضافه کردن اتاق به لیست بود، اتاق مورد نظر اضافه خواهد شد و اگر کاربر درخواست حذف را داشت، اتاق از لیست حذف خواهد شد. سپس کاربر به صفحه مشخصات همان اتاق هدایت می شود.

یک کلاس مشاهده لیست مورد علاقه ها نیز پیاده سازی می شود که فقط شامل آدرس تمپلیت مربوطه است.

نتیجه نهایی به صورت زیر خواهد بود:

```

9 @login_required
10 def toggle_room(request, room_pk):
11     action = request.GET.get("action", None)
12     room = room_models.Room.objects.get_or_none(pk=room_pk)
13     if room is not None and action is not None:
14         the_list, _ = models.List.objects.get_or_create(
15             user=request.user, name="My Favourites Houses"
16         )
17         if action == "add":
18             the_list.rooms.add(room)
19         elif action == "remove":
20             the_list.rooms.remove(room)
21     return redirect(reverse("rooms:detail", kwargs={"pk": room_pk}))
22
23
24 class SeeFavsView(TemplateView):
25
26     template_name = "lists/list_detail.html"

```

تصویر ۴-۵ ویو lists

۴-۱-۶ توسعه ویو `conversations`:

این ویو شامل یک تابع ایجاد بستر مکالمه و یک کلاس مشاهده جزئیات مکالمه است.

بعنوان مثال، برای پیاده سازی تابع مربوطه، ابتدا کاربران حاضر در مکالمه در پایگاه داده جستجو می شوند و اگر وجود داشتند، بعنوان مشارکت کنندگان در نظر گرفته می شوند.

سپس اگر بستر مکالمه وجود نداشت یا عبارتی دیگر این دو کاربر تابحال گفتگو ای با هم نداشتند، یک بستر ایجاد می شود و این دو کاربر به آن بستر اضافه خواهند شد. و در پایان به صفحه جزئیات گفتگو منتقل خواهند شد.

نتیجه نهایی به صورت زیر خواهد بود:

```
10 def go_conversation(request, a_pk, b_pk):
11     user_one = user_models.User.objects.get(pk=a_pk)
12     user_two = user_models.User.objects.get(pk=b_pk)
13     if user_one is not None and user_two is not None:
14         try:
15             conversation = models.Conversation.objects.get(
16                 Q(participants=user_one) & Q(participants=user_two)
17             )
18         except models.Conversation.DoesNotExist:
19             conversation = models.Conversation.objects.create()
20             conversation.participants.add(user_one, user_two)
21     return redirect(reverse("conversations:detail", kwargs={"pk": conversation.pk}))
```

تصویر ۴-۶ ویو `conversations`

۴-۲ توسعه URL ها از طریق فایل `urls.py`

پس از توسعه مدل ها و ویو ها، لازم است که صفحات مورد نیاز وبسایت آدرس دهی شوند. در هر اپلیکیشن یک فایل `urls.py` وجود دارد که همین وظیفه را بر عهده دارد. پس از اضافه کردن این آدرس ها، باید آدرس ریشه هر اپلیکیشن در فایل `urls.py` کل پروژه اضافه شود. در ضمن در هر فایل `urls.py` باید نام اپلیکیشن مشخص شود.

نکته مهم دیگر این است که در بعضی از صفحات نیاز است که داده هایی از طریق URL به `view` مورد نظر انتقال داده شود و از همان داده ارسالی برای ایجاد `query` های مورد نیاز و بازیابی اطلاعات پایگاه داده استفاده شود. به این منظور باید در URL مربوطه، `id` مورد نظر مشخص شود.

مثلا در URL زیر برای ورود به صفحه مشخصات پروفایل، باید `id` کاربر به URL داده شود:

```
path("<int:pk>/", views.UserProfileView.as_view(), name="profile")
```

بنابراین در این مرحله قرار است تمام فایل های `urls.py` توسعه داده شوند. آدرس ها در این فایل ها به ۳ بخش تقسیم می شوند:

۱. آدرس (به همراه id مربوطه در صورت نیاز)

۲. ویو مربوطه

۳. نام آدرس

۴-۲-۱ توسعه آدرس های `users`:

```
4 app_name = "users"
5
6 urlpatterns = [
7     path("login/", views.LoginView.as_view(), name="login"),
8     path("login/github/", views.github_login, name="github-login"),
9     path("login/github/callback/", views.github_callback, name="github-callback"),
10    path("logout/", views.log_out, name="logout"),
11    path("sigup/", views.SignUpView.as_view(), name="signup"),
12    path(
13        "verify/<str:key>/", views.complete_verification, name="complete-verification"
14    ),
15    path("update-profile/", views.UpdateProfileView.as_view(), name="update"),
16    path("update-password/", views.UpdatePasswordView.as_view(), name="password"),
17    path("<int:pk>/", views.UserProfileView.as_view(), name="profile"),
18    path("switch-hosting/", views.switch_hosting, name="switch-hosting"),
19    path("switch-language/", views.switch_language, name="switch-language"),
20 ]
```

تصویر ۴-۷ URL های `users`

۴-۲-۲ توسعه آدرس های `rooms`:

```
4 app_name = "rooms"
5
6 urlpatterns = [
7     path("create/", views.CreateRoomView.as_view(), name="create"),
8     path("<int:pk>/", views.RoomDetail.as_view(), name="detail"),
9     path("<int:pk>/edit/", views.EditRoomView.as_view(), name="edit"),
10    path("<int:pk>/photos/", views.RoomPhotosView.as_view(), name="photos"),
11    path("<int:pk>/photos/add", views.AddPhotoView.as_view(), name="add-photo"),
12    path(
13        "<int:room_pk>/photos/<int:photo_pk>/delete/",
14        views.delete_photo,
15        name="delete-photo",
16    ),
17    path(
18        "<int:room_pk>/photos/<int:photo_pk>/edit/",
19        views.EditPhotoView.as_view(),
20        name="edit-photo",
21    ),
22    path("search/", views.SearchView.as_view(), name="search"),
23 ]
```

تصویر ۴-۸ URL های `rooms`

۳-۲-۴ توسعه آدرس های reviews:

```
5 app_name = "reviews"
6
7 urlpatterns = [
8     path("create/<int:room>", views.create_review, name="create"),
9 ]
```

تصویر ۹-۴ URL های reviews

۴-۲-۴ توسعه آدرس های reservations:

```
4 app_name = "reservations"
5
6 urlpatterns = [
7     path(
8         "create/<int:room>/<int:year>-<int:month>-<int:day>",
9         views.create,
10        name="create",
11    ),
12    path("<int:pk>/", views.ReservationDetailView.as_view(), name="detail"),
13    path("<int:pk>/<str:verb>", views.edit_reservation, name="edit"),
14 ]
```

تصویر ۱۰-۴ URL های reservations

۵-۲-۴ توسعه آدرس های lists:

```
4 app_name = "lists"
5
6 urlpatterns = [
7     path("toggle/<int:room_pk>", views.toggle_room, name="toggle-room"),
8     path("favs/", views.SeeFavsView.as_view(), name="see-favs"),
9 ]
```

تصویر ۱۱-۴ URL های lists

۶-۲-۴ توسعه آدرس های conversations:

```
4 app_name = "conversations"
5
6 urlpatterns = [
7     path("go/<int:a_pk>/<int:b_pk>", views.go_conversation, name="go"), # ty
8     path("<int:pk>/", views.ConversationDetailView.as_view(), name="detail"),
9 ]
```

تصویر ۱۲-۴ URL های conversations

۷-۲-۴ توسعه آدرس های core:

این آدرس در واقع همان صفحه اصلی وبسایت^{۵۰} است.

```
4 app_name = "core"
5
6 urlpatterns = [
7     path("", room_views.HomeView.as_view(), name="home"),
8 ]
```

تصویر ۴-۱۳ URL های core

۸-۲-۴ توسعه آدرس های config:

در این فایل علاوه بر آدرس ریشه تمام اپلیکیشن ها، آدرس فایل های رسانه ای^{۵۱} هم باید مشخص شود:

```
7 urlpatterns = [
8     path("", include("core.urls", namespace="core")),
9     path("rooms/", include("rooms.urls", namespace="rooms")),
10    path("users/", include("users.urls", namespace="users")),
11    path("reservations/", include("reservations.urls", namespace="reservations")),
12    path("reviews/", include("reviews.urls", namespace="reviews")),
13    path("lists/", include("lists.urls", namespace="lists")),
14    path("conversations/", include("conversations.urls", namespace="conversations")),
15    path("admin/", admin.site.urls),
16 ]
17
18
19 if settings.DEBUG:
20     urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

تصویر ۴-۱۴ URL های config

⁵⁰ Home Page

⁵¹ Media Files

فصل پنجم

توسعه Template ها

در این فصل، بخش سوم و پایانی مدل MVT یعنی تمپلیت ها توسعه داده خواهند شد.

۵-۱ تفاوت فایل های Media و فایل های Static

مدیا به فایل هایی اطلاق می شود که در حین استفاده از وبسایت و توسط دیگر کاربران آپلود می شوند اما فایل های استاتیک یا ایستا فایل هایی هستند که به هیچ وجه قرار نیست تغییر کنند؛ پس باید شخصا توسط توسعه دهنده در دایرکتوری پروژه قرار داده شوند.

۵-۲ نحوه تنظیم Static و Media در پروژه

برای استفاده از این فایل ها، باید یک URL و یک آدرس Root مشخص برای آنها در فایل settings.py اضافه شود.

کد مربوط به فایل های استاتیک:

```
STATIC_URL = "/static/"
STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")]
```

کد مربوط به فایل های مدیا:

```
MEDIA_URL = "/media/"
MEDIA_ROOT = os.path.join(BASE_DIR, "uploads")
```

همچنین کد زیر باید به فایل urls.py اصلی اضافه شود.

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

۵-۳ نحوه تنظیم Template ها در پروژه

پس از اجرای دستور startproject که در فصل دوم به آن اشاره شد، جنگو تنظیمات مربوط به تمپلیت ها را در فایل settings.py انجام می دهد اما دایرکتوری ای برای آن در نظر نمی گیرد.

بنابراین توسعه دهنده باید یک پوشه در محل پروژه ایجاد کند به نام templates سپس آدرس آن را در تنظیمات اضافه کند:

```

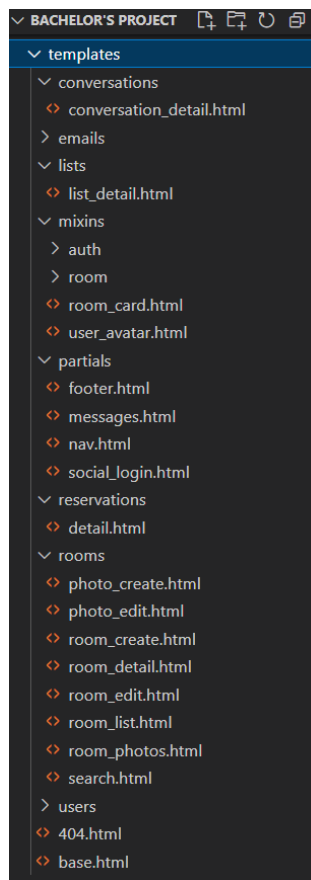
72 TEMPLATES = [
73     {
74         "BACKEND": "django.template.backends.django.DjangoTemplates",
75         "DIRS": [os.path.join(BASE_DIR, "templates")],
76         "APP_DIRS": True,
77         "OPTIONS": {
78             "context_processors": [
79                 "django.template.context_processors.debug",
80                 "django.template.context_processors.request",
81                 "django.contrib.auth.context_processors.auth",
82                 "django.contrib.messages.context_processors.messages",
83             ],
84         },
85     ],
86 ]

```

تصویر ۵-۱ تنظیم تمپلیت ها در پروژه

۵-۴ پیاده سازی تمپلیت ها

پس از ایجاد پوشه مربوط به تمپلیت ها، لازم است که تمپلیت های مربوط به هم اپلیکیشن نیز داخل پوشه های جداگانه سازماندهی شوند. که این سازماندهی برای این پروژه بصورت زیر انجام شده است:



تصویر ۵-۲ سازماندهی تمپلیت ها

تمامی فایل های تمپلیت شامل پیاده سازی کد های زیر خواهند بود:

- HTML
- CSS [7]
- Template Tags (برای پیاده سازی عملیات های منطقی مانند حلقه یا شرط)
- JavaScript (فقط در یک مورد) [8]

همانطور که در فصل قبل اشاره شد هدف اصلی تمپلیت ها، ایجاد رابط کاربری گرافیکی^{۵۲} و تعامل با کاربران است به این صورت که تمپلیت ها با کمک فرم ها از کاربر اطلاعات دریافت می کنند و به view ارسال می کنند تا پردازش مورد نظر انجام شود.

باید به این نکته توجه شود که فایل های تمپلیت بیشترین حجم و زمان برای پیاده سازی را شامل می شوند بنابراین در این فصل، جهت پرهیز از زیاده گویی، سعی شده فقط کلیات بیان شود.

لازم به ذکر است که بعلت طولانی بودن تعداد خطوط کد های مربوط به فایل های تمپلیت، تصاویر کد های پیاده سازی شده ای که در ادامه خواهید دید، فقط بخشی از کد اصلی را شامل می شوند. تمام کد ها و جزئیات پیاده سازی در ریپازیتوری گیت هاب پروژه در دسترس است.

۵-۴-۱ توسعه base.html:

این فایل پایه و اساس تمام تمپلیت ها است به این صورت که از این پس بقیه تمپلیت ها با استفاده از دستور include می توانند به محتویات این فایل دسترسی پیدا کنند.

در این فایل تنظیمات لوگو، باکس جستجوی اتاق و کد JavaScript مربوط به ترجمه محتویات وبسایت پیاده سازی شده است. از تمپلیت های import شده در این فایل می توان به messages، nav و footer اشاره کرد.

۵-۴-۲ توسعه 404.html

محتویات این فایل در هنگام ورود کاربر به URL ای که وجود ندارد، نمایش داده خواهد شد.

⁵² Graphical User Interface


```

404.html X
templates > 404.html > ...
1  {% extends "base.html" %}
2  {% load static i18n %}
3  {% block page_title %}
4  Not found
5  {% endblock page_title %}
6
7  {% block content %}
8
9  <h1>{% trans "Sorry Not found" %}!</h1>
10
11
12  {% endblock content %}

```

تصویر ۳-۵ فایل 404.html

۳-۴-۵ توسعه تمپلیت های گروه users

در اولین فایل از تمپلیت های این گروه، پیاده سازی تمپلیت مربوط به ورود کاربر انجام شده است. این تمپلیت و بسیاری دیگر از تمپلیت های دیگر از base.html ارث بری دارند.

جهت ورود کاربر با اکانت گیت هاب و همچنین اعتبارسنجی داده های ورودی، دو تمپلیت social_login و auth_form به این فایل import شده اند. همچنین اگر کاربر اکانت ای ندارد، می تواند با استفاده از لینک داده شده، به صفحه ثبت نام ارجاع داده شود.

```

login.html X
templates > users > login.html > ...
1  {% extends "base.html" %}
2  {% load static i18n %}
3  {% block page_title %}
4  Log In
5  {% endblock page_title %}
6
7  {% block search-bar %}
8  {% endblock search-bar %}
9
10 {% block content %}
11
12 <div class="container lg:w-5/12 md:w-1/4 xl:w-1/4 mx-auto my-10 flex flex-col items-center border p-6 border-gray-400">
13
14   {% include 'partials/social_login.html' %}
15
16   {% include 'mixins/auth/auth_form.html' with form=form cta="Login" %}
17
18   <div class="mt-5">
19     <span class="mr-2">{% trans "Don't have an account" %}?</span>
20     <a href="{% url 'users:signup' %}" class="text-teal-500 font-medium">{% trans "Sign up" %}</a>
21   </div>
22 </div>
23 {% endblock content %}

```

تصویر ۴-۵ تمپلیت لاگین

دیگر تمپلیت های این گروه عبارتند از: ثبت نام - تغییر رمز عبور - تغییر مشخصات پروفایل - مشخصات کاربر

۵-۴-۴ توسعه تمپلیت های گروه rooms

در فایل search از این گروه، به پیاده سازی تمپلیت صفحه جستجو بین اتاق ها پرداخته شده است. در ابتدا یک فرم با متد GET ایجاد شده که اطلاعا کاربر را دریافت می کند سپس برای نمایش نتایج از تمپلیت room_card استفاده شده است که وظیفه اش نمایش اتاق ها همراه با عکس و اطلاعات اولیه آن ها است.

```
{% block content %}

<div class="container lg:w-5/12 md:w-1/2 xl:w-1/4 mx-auto my-10 flex flex-col items-center border p-6 border-gray-400">
  <form method="get" action="{% url 'rooms:search' %}">
    {{form.as_p}}
    <button class="btn bg-red-500 text-white mt-5">{% trans "Search" %}</button>
  </form>
</div>

<div class="min-h-75vh">

  <h3 class="mb-12 text-2xl text-center">{% trans "Results" %}</h3>
  <div class="container mx-auto pb-10 ">
    <div class="flex flex-wrap -mx-24 mb-10">
      {% for room in rooms %}
      {% include 'mixins/room_card.html' with room=room %}
      {% endfor %}
    </div>
  </div>
</div>

{% endblock content %}
```

تصویر ۵-۵ تمپلیت سرچ

دیگر تمپلیت های این گروه عبارتند از: آپلود عکس - تغییر عکس - ساخت اتاق - جزئیات اتاق - تغییر مشخصات اتاق - لیست اتاق ها - عکس اتاق ها

۵-۴-۵ توسعه تمپلیت های گروه reservation

فقط یک فایل detail در این گروه وجود دارد که وظیفه اش نمایش اطلاعات رزرواسیون است همراه با قابلیت تایید یا کنسل کردن درخواست و یا مکالمه با میزبان.

```
<div class="py-10 px-5">
  {% if reservation.status != 'canceled' %}
  {% if reservation.status == 'confirmed' and reservation.is_finished %}
  <span class="font-medium text-2xl text-center w-full block mb-5">{% trans "Write your review" %}</span>
  <form action="{% url 'reviews:create' reservation.room.pk %}" method="POST" class="w-1/2 mx-auto">
    {% csrf_token %}
    {{form}}
    <button class="btn-link mt-5">{% trans "Submit Review" %}</button>
  </form>
  {% else %}
  {% if reservation.status == 'pending' %}
  <a href="{% url 'reservations:edit' reservation.pk 'cancel' %}" class="btn-link block px-5">{% trans "Cancel Reservation" %}</a>
  {% if reservation.room.host == user %}
  <a href="{% url 'reservations:edit' reservation.pk 'confirm' %}" class="btn-link block px-3 mb-5">{% trans "Confirm Reservation" %}</a>
  {% endif %}
  {% endif %}
  {% endif %}
  {% endif %}
</div>
```

تصویر ۵-۶ تمپلیت جزئیات رزرواسیون

۵-۴-۶ توسعه تمپلیت های گروه partials

در اولین فایل از این گروه، جزئیات footer پیاده سازی شده است که شامل اطلاعات مربوط به حق کپی رایت^{۵۳} و گزینه تغییر زبان وبسایت می باشد.

دیگر تمپلیت های این گروه عبارتند از: پیام های سیستمی – نوار پیمایش (nav) – ورود با اکانت گیت هاب

```

<> footer.html X
templates > partials > <> footer.html > ...
1  {% load i18n %}
2  <footer class="container mx-auto text-center py-10 border-t font-medium text-gray-600">
3      <div class="flex flex-col">
4          <span>
5              {% trans "This is Ilya Jafari's Bachelor's project supervised by professor Rahmanimanesh" %}
6          </span>
7          <span>&copy; {% trans "2022 Semnan University, All rights reserved" %}</span>
8      </div>
9      <div class="mt-10 flex">
10         {% get_current_language as LANGUAGE_CODE %}
11         <p>{% trans "Language:" %}&nbsp;</p>
12         <select class="w-1/6 h-8" id="js-lang">
13             <option value="en" {% if LANGUAGE_CODE == 'en' %} selected {% endif %}>English</option>
14             <option value="fa" {% if LANGUAGE_CODE == 'fa' %} selected {% endif %}>Farsi</option>
15         </select>
16     </div>
17
18 </footer>
```

تصویر ۷-۵ تمپلیت فوتر

۵-۴-۷ توسعه تمپلیت های گروه mixins

mixins شامل متد هایی است که دیگر تمپلیت ها می توانند بدون ارث بری، از آن ها استفاده کنند.

در فایل user_avatar نحوه نمایش عکس پروفایل کاربر پیاده سازی شده است به صورتی که اگر کاربر عکس پروفایل داشت، نمایش داده شود در غیر اینصورت یک دایره خاکستری نمایش داده شود.

دیگر تمپلیت های این گروه عبارتند از: اعتبارسنجی فرم – اعتبارسنجی اتاق – نمایش اتاق ها همراه با جزئیات

۵-۴-۷ توسعه تمپلیت های گروه lists

فقط فایل list_detail در این گروه وجود دارد که شامل پیاده سازی لیست اتاق های مورد علاقه کاربر است.

```
{% block content %}

<div class="min-h-75vh">

  <h3 class="mb-12 text-2xl text-center">{% trans "Your Favourites" %}</h3>

  <div class="container mx-auto pb-10 ">
    <div class="flex flex-wrap -mx-24 mb-10">
      {% for room in user.list.rooms.all %}
      {% include 'mixins/room_card.html' with room=room %}
      {% endfor %}
    </div>
  </div>

</div>
{% endblock content %}
```

تصویر ۵-۸ تمپلیت لیست ها

۵-۴-۸ توسعه تمپلیت های گروه conversations

فقط فایل conversation_detail در این گروه وجود دارد که شامل پیاده سازی جزئیات مکالمه بین دو کاربر است.

پیاده سازی به این صورت است که ابتدا مشارکت کنندگان همراه با نام و عکس پروفایل نمایش داده خواهند شد. سپس اگر پیام ای وجود داشته باشد نمایش داده خواهد شد در غیر این صورت نوشته می شود که پیام ای وجود ندارد.

در پایان، یک فرم برای ارسال پیام وجود دارد.

```

10 {% block content %}
11
12 <div class="container mx-auto my-10 mt-32 flex justify-between min-h-50vh">
13
14     <div class="border w-1/4 p-10">
15         <span class="text-center w-full block text-lg font-medium">{% trans "Conversation between" %}</span>
16         <div class="flex justify-between mt-10 items-center">
17             {% for user in conversation.participants.all %}
18                 <div class="flex flex-col items-center">
19                     {% include "mixins/user_avatar.html" with user=user %}
20                     <span class="mt-2 text-gray-500">{{user.first_name}}</span>
21                 </div>
22             {% if forloop.first %}
23                 <span class="font-medium text-2xl">&</span>
24             {% endif %}
25             {% endfor %}
26         </div>
27     </div>
28     <div class="flex-grow">
29         <div class="border ml-10 p-10 flex flex-col">
30             {% if conversation.messages.count == 0 %}
31                 {% trans "no messages" %}
32             {% else %}
33                 {% for message in conversation.messages.all %}
34                     <div class="mb-10 {% if message.user.pk == user.pk %}
35                         self-end
36                         text-right
37                     {% endif %}">
38                         <span class="text-sm font-medium text-gray-600">{{message.user.first_name}}</span>
39                         <div class="mt-px p-5 w-56 rounded
40                             {% if message.user.pk != user.pk %}
41                                 bg-teal-500
42                                 text-white
43                             {% else %}
44                                 bg-gray-300
45                             {% endif %}
46                         ">
47                             {{message.message}}
48                     </div>
49                 </div>
50             {% endfor %}
51             {% endif %}
52         </div>
53         <form class="mt-10 w-1/2 mx-auto" method="POST">
54             {% csrf_token %}
55             <input class="border-box mb-5" name="message" placeholder="Write a Message" required />
56             <button class="btn-link">{% trans "Send Comment" %}</button>
57         </form>
58     </div>
59 </div>
60
61 </div>
62 {% endblock content %}

```

تصویر ۵-۹ تمپلیت مکالمه

فصل ششم

توسعه فرم ها

برای دریافت اطلاعات یا تغییر آن ها توسط کاربرانی که از این وبسایت استفاده می کنند، لازم است از فرم های جنگو استفاده کنیم.

۶-۱ تعریف فرم ها

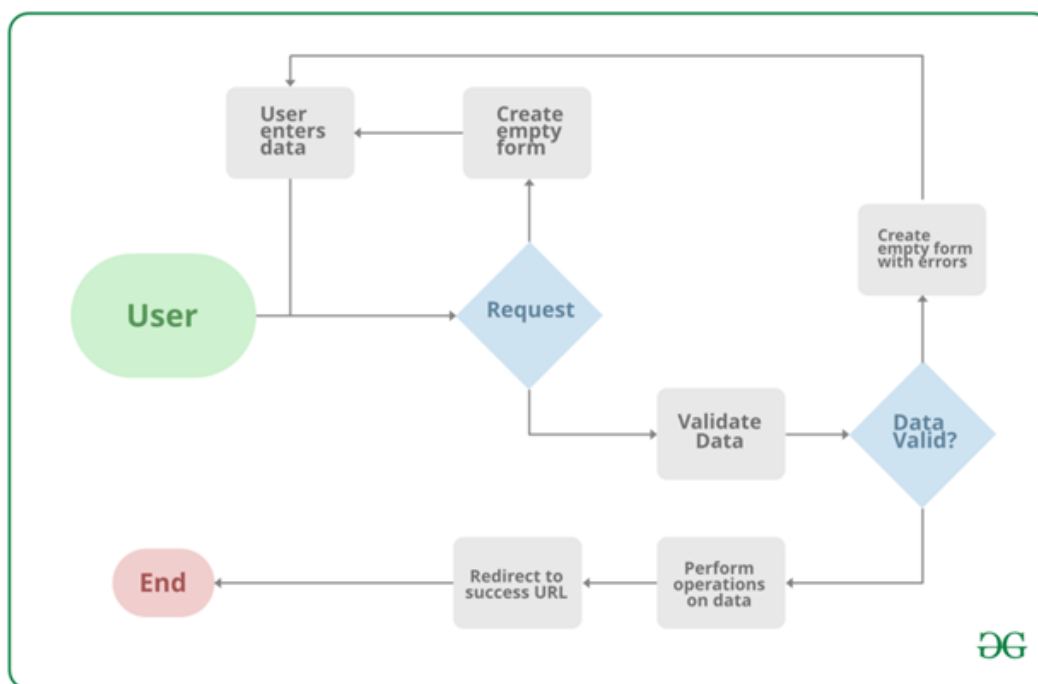
فرم ها نقش مهمی در شکل گیری ارتباطات بین کاربر و وبسایت ایفا می کنند چه این کاربران از مدیران وبسایت باشند، چه افراد بازدید کننده از وبسایت.

اساساً از فرم ها برای گرفتن ورودی از کاربر و استفاده آن در عملیات منطقی در پایگاه داده ها استفاده می شود. به عنوان مثال، ثبت نام یک کاربر با در نظر گرفتن نام، ایمیل، رمز عبور و غیره. تمامی فرم ها در فایل forms.py در پوشه اپلیکیشن ها پیاده سازی خواهند شد.

۶-۱-۱ فرم جنگو

Django ModelForm کلاسی است که برای تبدیل مستقیم مدل به فرم جنگو از آن استفاده می شود. اگر قصد برنامه نویسی، ایجاد یک برنامه مبتنی بر پایگاه داده است، در این صورت استفاده از این نوع از فرم الزامی می باشد و البته مسیر پیاده سازی را نیز آسان تر می کند.

توجه داشته باشید که همه انواع کارهایی که توسط فرم های جنگو انجام می شود را می توان با موارد پیشرفته تر HTML انجام داد، اما استفاده از جنگو، آسان تر و کارآمدتر است به ویژه در قسمت اعتبارسنجی.



تصویر ۶-۱ دیاگرام روش کار فرم ها

۶-۲ توسعه فرم ها از طریق فایل forms.py

قبل از پیاده سازی فرم ها باید به این مورد توجه کرد که همه اپلیکیشن ها نیازی به فرم ندارند.

۶-۲-۱ فرم users

کاربران به دو فرم ورود (لاگین) و ثبت نام نیاز دارند.

در فرم مربوط به لاگین، ایمیل و رمز عبور از کاربر گرفته می شوند. سپس با تابع `clean` داده های ورودی تمیز خواهند شد^{۵۴} تا آماده مقایسه^{۵۵} با داده های پایگاه داده شوند.

اگر این مقایسه جواب مثبت داشت، درخواست کاربر وارد مراحل بعدی خواهد شد و اگر جواب منفی بود، با توجه به ارور ایجاد شده، یکی از پیام های "رمز عبور اشتباه است" یا "این کاربر وجود ندارد" نمایش داده خواهند شد.

```
class LoginForm(forms.Form):

    email = forms.EmailField(widget=forms.EmailInput(attrs={"placeholder": _("Email")}))
    password = forms.CharField(
        widget=forms.PasswordInput(attrs={"placeholder": _("Password")})
    )

    def clean(self):
        email = self.cleaned_data.get("email")
        password = self.cleaned_data.get("password")
        try:
            user = models.User.objects.get(email=email)
            if user.check_password(password):
                return self.cleaned_data
            else:
                self.add_error(
                    "password", forms.ValidationError(_("Password is wrong"))
                )
        except models.User.DoesNotExist:
            self.add_error("email", forms.ValidationError(_("User does not exist")))
```

تصویر ۶-۲ فرم لاگین کاربر

در فرم مربوط به ثبت نام، اطلاعات مربوط به نام، نام خانوادگی، ایمیل و رمز عبور از کاربر گرفته خواهد شد. سپس با دو تابع مجزا، داده های ورودی ایمیل و رمز عبور تمیز و اعتبارسنجی خواهند شد.

⁵⁴ Clean Data

⁵⁵ Comparison

و در پایان، اطلاعات کاربر جدید در پایگاه داده ذخیره خواهد شد.

```
28 class SignUpForm(forms.ModelForm):
29     class Meta:
30         model = models.User
31         fields = ("first_name", "last_name", "email")
32         widgets = {
33             _("first_name"): forms.TextInput(attrs={"placeholder": _("First Name")}),
34             _("last_name"): forms.TextInput(attrs={"placeholder": _("Last Name")}),
35             _("email"): forms.EmailInput(attrs={"placeholder": _("Email")}),
36         }
37
38         password = forms.CharField(
39             widget=forms.PasswordInput(attrs={"placeholder": _("Password")})
40         )
41         password1 = forms.CharField(
42             widget=forms.PasswordInput(attrs={"placeholder": _("Confirm Password")})
43         )
44
45     def clean_email(self):
46         email = self.cleaned_data.get("email")
47         try:
48             models.User.objects.get(email=email)
49             raise forms.ValidationError(
50                 _("That email is already taken"), code="existing_user"
51             )
52         except models.User.DoesNotExist:
53             return email
54
55     def clean_password1(self):
56         password = self.cleaned_data.get("password")
57         password1 = self.cleaned_data.get("password1")
58
59         if password != password1:
60             raise forms.ValidationError(_("Password confirmation does not match!"))
61         else:
62             return password
```

تصویر ۳-۶ فرم ثبت نام کاربر

۶-۲-۲ فرم rooms

سه فرم جستجوی اتاق، آپلود عکس اتاق و ساخت اتاق در این فایل پیاده سازی خواهند شد.

برای جستجو بین اتاق ها، از دو متغیر نام شهر و نام کشور استفاده خواهد شد که کشور پیش فرض، ایران است.

برای آپلود عکس، فرم ما شامل دو فیلد کپشن و فایل خواهد بود که پس از ورود داده ها توسط کاربر، عکس ها در پایگاه داده ذخیره خواهند شد.

و در پایان در فرم ساخت اتاق، پس از وارد کردن داده ها، اطلاعات در پایگاه داده ذخیره خواهد شد. فیلد های ورودی عبارتند از:

نام، توضیحات، کشور، شهر، قیمت، آدرس، تعداد مهمان، تعداد تخت، تعداد اتاق خواب، تعداد سرویس بهداشتی، ساعت ورود، ساعت خروج، نوع اتاق، امکانات رفاهی، تسهیلات و قوانین

```
7  class SearchForm(forms.Form):
8      city = forms.CharField(initial="Anywhere")
9      country = CountryField(default="IR").formfield()
32 class CreatePhotoForm(forms.ModelForm):
33     class Meta:
34         model = models.Photo
35         fields = ("caption", "file")
36
37     def save(self, pk, *args, **kwargs):
38         photo = super().save(commit=False)
39         room = models.Room.objects.get(pk=pk)
40         photo.room = room
41         photo.save()
44 class CreateRoomForm(forms.ModelForm):
45     class Meta:
46         model = models.Room
47         fields = (
48             "name",
49             ("description"),
50             ("country"),
51             ("city"),
52             ("price"),
53             ("address"),
54             ("guests"),
55             ("beds"),
56             ("bedrooms"),
57             ("baths"),
58             ("check_in"),
59             ("check_out"),
60             ("instant_book"),
61             ("room_type"),
62             ("amenities"),
63             ("facilities"),
64             ("house_rules"),
65         )
66
67     def save(self, *args, **kwargs):
68         room = super().save(commit=False)
69         return room
```

تصویر ۴-۶ فرم جستجو بین اتاق ها

۶-۲-۳ فرم reviews

در این فرم نقد کاربر و تمام امتیازهای لازم جهت ایجاد یک نقد از کاربر دریافت می‌شود، سپس نقد و امتیاز در پایگاه داده ذخیره می‌شود.

```
5 class CreateReviewForm(forms.ModelForm):
6     accuracy = forms.IntegerField(max_value=5, min_value=1)
7     communication = forms.IntegerField(max_value=5, min_value=1)
8     cleanliness = forms.IntegerField(max_value=5, min_value=1)
9     location = forms.IntegerField(max_value=5, min_value=1)
10    check_in = forms.IntegerField(max_value=5, min_value=1)
11    value = forms.IntegerField(max_value=5, min_value=1)
12
13    class Meta:
14        model = models.Review
15        fields = (
16            ("review"),
17            ("accuracy"),
18            ("communication"),
19            ("cleanliness"),
20            ("location"),
21            ("check_in"),
22            ("value"),
23        )
24
25    def save(self):
26        review = super().save(commit=False)
27        return review
```

تصویر ۵-۶ فرم reviews

۶-۲-۴ فرم conversations

در این فرم، پیام مورد نظر از کاربر گرفته خواهد شد.

```
5 class AddCommentForm(forms.Form):
6
7     message = forms.CharField(
8         required=True, widget=forms.TextInput(attrs={"placeholder": _("Add a Comment")})
9     )
10
```

تصویر ۶-۶ فرم conversations

مراجع

- [1] Django Software Foundation, "Documentation," [Online]. Available: <https://docs.djangoproject.com/en/2.2/>
- [2] Microsoft, "Download Visual Studio Code," [Online]. Available: <https://code.visualstudio.com/download>
- [3] GitHub, "GitHub Desktop," [Online]. Available: <https://desktop.github.com/>
- [4] "Git Documentation," [Online]. Available: <https://git-scm.com/doc>
- [5] E. Jafari, "Bachelor-Project Repository," [Online]. Available: <https://github.com/ilyaJafari99/Bachelor-Project>
- [6] Python, "Python documentation," [Online]. Available: <https://docs.python.org/3/>
- [7] tailwindcss, "Getting started with Tailwind CSS," [Online]. Available: <https://v2.tailwindcss.com/docs>
- [8] Mozilla.org, "JavaScript," [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>



Semnan University

Faculty of Electrical & Computer Engineering

B.Sc. Thesis in Computer Engineering

Design & Implementation of a Website for Rental Properties

By:

Eilia Jafari Chamazkoti

Supervisor:

Dr. Mohammad RahmaniManesh

January 2023