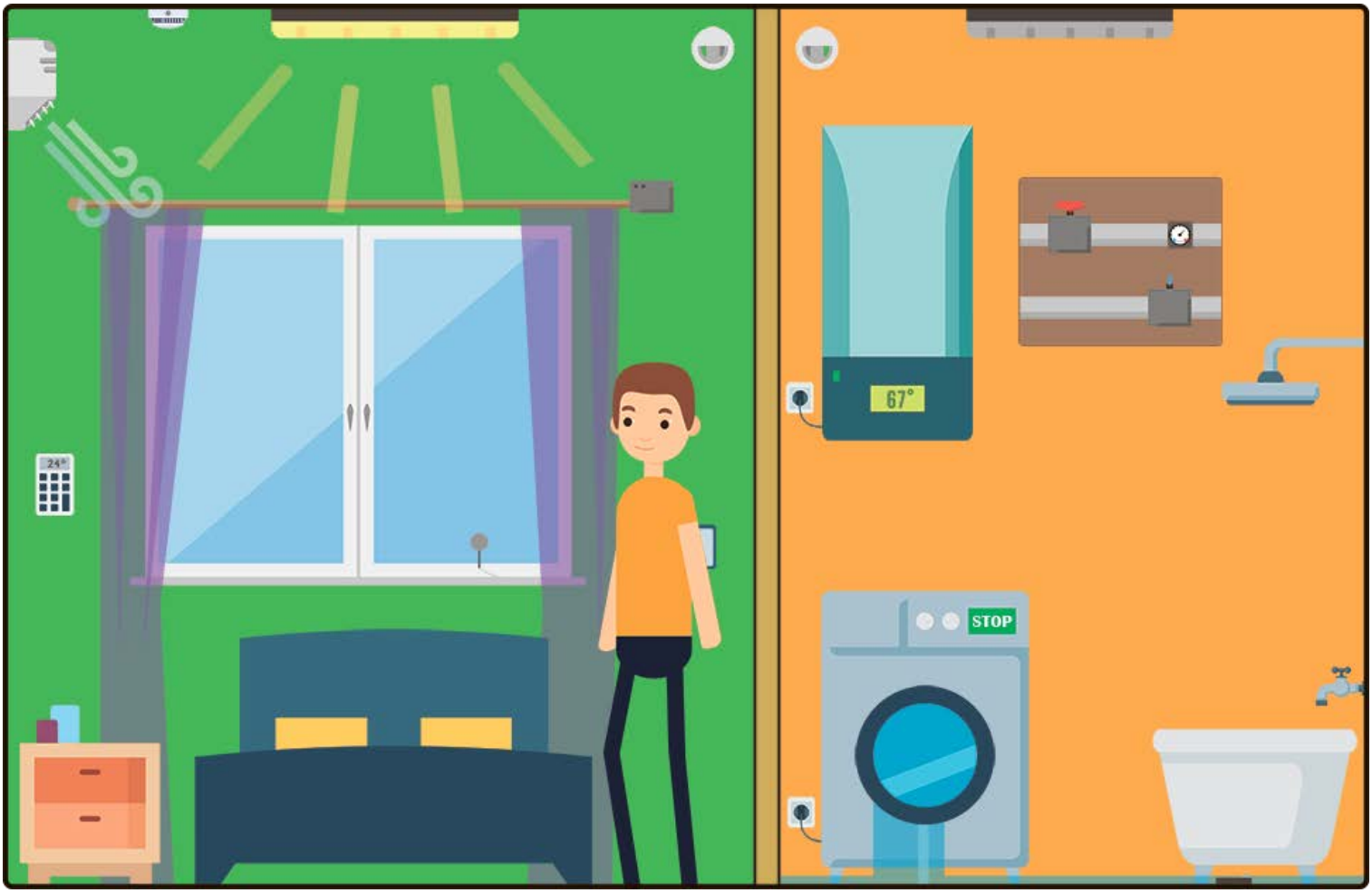
Задание по программированию: Курсовой проект «Web-приложение для управления умным домом»

Вашей задачей будет реализовать на Django сервер управления умным домом, имеющий web-интерфейс для настройки и ручного управления, который будет производить периодический опрос датчиков и осуществлять автоматическую реакцию в случае определенных ситуаций, используя API контроллера умного дома

Авторы курса сделали виртуальные контроллер, датчики и устройства, которыми он управляет. Зайдя на сайт и зарегистрировавшись, вы получите уникальный ключ (KEY), который нужно будет использовать при работе с API контроллера. Используя этот же ключ можно получить изображение с “виртуальной камеры” умного дома, на которой будет наглядно видно, какие устройства работают или почему сработал тот или иной датчик, и вручную управлять устройствами. Документацию по API можно посмотреть на этом же сайте.



Устройства, подключенные к контроллеру, доступны на запись (обычно true - включить/открыть, false - выключить/закрыть, но бывают варианты). И датчики и устройства доступны на чтение. Устройства, при чтении с них, работают как датчики и возвращают свое состояние, которое может отличаться от записанного.

Устройства (запись):

- **air_conditioner** – Кондиционер (true – вкл, false – выкл). При включении постепенно понижает температуру в спальне, пока она не достигнет 16 градусов и сильнее охладить уже не может.
- **bedroom_light** – Лампа в спальне (true – вкл, false – выкл).
- **bathroom_light** – Лампа в ванной (true – вкл, false – выкл).
- **curtains** – Занавески string (“open” – открыть, “close” – закрыть).
- **boiler** – Бойлер (true – вкл, false – выкл). При включении постепенно повышает температуру воды, пока она не достигнет 90 градусов. Для работы должен быть открыт входной кран холодной воды.
- **cold_water** – Входной кран холодной воды (true – открыть, false – закрыть). Позволяет открыть/перекрыть подачу

холодной воды в квартиру

- **hot_water** – Входной кран горячей воды (true – открыть, false – закрыть).
- **washing_machine** – Стиральная машина string (“on” – вкл, “off” – выкл). При включении начинает стирать, потом самостоятельно отключается. Может сломаться и протечь.

Датчики (чтение):

- **air_conditioner** – Кондиционер. (true – вкл, false – выкл).
- **bedroom_temperature** – Температура в спальне. Int (0 – 80).
- **bedroom_light** – Лампа в спальне. (true – вкл, false – выкл).
- **smoke_detector** – Датчик задымления на потолке. (true – задымление, false – нет).
- **bedroom_presence** – Датчик присутствия в спальне. (true – есть человек, false – нет).
- **bedroom_motion** – Датчик движения в спальне. (true – есть движение, false – нет).
- **curtains** – Занавески. string (“open” – открыты, “close” – закрыты, “slightly_open” – приоткрыты вручную).
- **outdoor_light** – Датчик освещенности за окном (0 – 100).
- **boiler** – Бойлер. (true – вкл, false – выкл).
- **boiler_temperature** – Температура горячей воды бойлере. Int (0 – 100 / null). Если перекрыта холодная вода, то воды в бойлере нет, и датчик возвращает null.
- **cold_water** – Входной кран холодной воды. (true – открыт, false – закрыт).
- **hot_water** – Входной кран горячей воды. (true – открыт, false – закрыт).
- **bathroom_light** – Лампа в ванной. (true – вкл, false – выкл).
- **bathroom_presence** – Датчик присутствия в ванной. (true – есть человек, false – нет).
- **bathroom_motion** – Датчик движения в ванной. (true – есть движение, false – нет)
- **washing_machine** – Стиральная машина. string (“on” – вкл, “off” – выкл, “broken” – сломана).
- **leak_detector** – Датчик протечки воды (true – протечка, false – сухо).

Приложение студента

Используя приложенный к этому заданию “скелет” приложения на Django (файл student.zip), вам нужно реализовать свой интерфейс управления умным домом.

studentФайл ZIP
Скачать файл

По адресу / должна открываться веб-форма, со следующими контролами:

- **bedroom_target_temperature** – input type=number, желаемая температура в спальне, запоминать настройку в базе, текущую настройку из базы выводить на форму. Допустимое значение от 16 до 50, default: 21
- **hot_water_target_temperature** – input type=number, желаемая температура горячей воды в доме, запоминать настройку в базе, текущую настройку из базы выводить на форму. Допустимое значение от 24 до 90, default: 80
- **bedroom_light** – checkbox, включает/выключает свет в спальне, синхронизировать значение с контроллером

- **bathroom_light** – checkbox, включает/выключает свет в ванной, синхронизировать значение с контроллером

Там же отображать текущие значения всех датчиков, прочитанные из контроллера. Для рендеринга шаблона прочитанные из контроллера значения должны быть в словаре в context.data.

Реализовать автоматически опрос контроллера в фоне каждую секунду (django celery) и реакцию на некоторые события.

Реакция на события:

1. Если есть протечка воды (leak_detector=true), закрыть холодную (cold_water=false) и горячую (hot_water=false) воду и отослать письмо в момент обнаружения.
2. Если холодная вода (cold_water) закрыта, немедленно выключить бойлер (boiler) и стиральную машину (washing_machine) и ни при каких условиях не включать их, пока холодная вода не будет снова открыта.
3. Если горячая вода имеет температуру (boiler_temperature) меньше чем hot_water_target_temperature - 10%, нужно включить бойлер (boiler), и ждать пока она не достигнет температуры hot_water_target_temperature + 10%, после чего в целях экономии энергии бойлер нужно отключить
4. Если шторы частично открыты (curtains == “slightly_open”), то они находятся на ручном управлении - это значит их состояние нельзя изменять автоматически ни при каких условиях.
5. Если на улице (outdoor_light) темнее 50, открыть шторы (curtains), но только если не горит лампа в спальне (bedroom_light). Если на улице (outdoor_light) светлее 50, или горит свет в спальне (bedroom_light), закрыть шторы. Кроме случаев когда они на ручном управлении
6. Если обнаружен дым (smoke_detector), немедленно выключить следующие приборы [air_conditioner, bedroom_light, bathroom_light, boiler, washing_machine], и ни при каких условиях не включать их, пока дым не исчезнет.
7. Если температура в спальне (bedroom_temperature) поднялась выше bedroom_target_temperature + 10% - включить кондиционер (air_conditioner), и ждать пока температура не опустится ниже bedroom_target_temperature - 10%, после чего кондиционер отключить.

Опрос контроллера и отправка ему ответа должны происходить внутри функции core.tasks.smart_home_manager. Эта функция должна вызываться периодически с интервалом в 5 секунд, например, с помощью celery. В начала своей работы функция запрашивает данные из контроллера, используя requests.get в API, затем анализирует настройки пользователя по желаемой температуре из БД, и текущую ситуацию, и в конце, если требуется коррекция, делает requests.post в API с командами для контроллера, если необходимо отправить письмо, то отправляет его.

Для отсылки писем нужно использовать send_mail из django.core.mail, а в settings нужно задать настройки EMAIL_HOST, EMAIL_PORT и другие EMAIL_*, так чтобы во время разработки вы отправляли письма через какую-нибудь почтовую систему и могли проверить их работу. Во время проверки задания на сервере эти настройки будут переопределены.

Для сохранения настроек в базе, нужно использовать модель Settings (name, value), заготовка для которой уже есть приложенных исходниках.

Веб-форма для настройки и управления умным домом должна открываться в корне сайта, содержать 4 input’а с именами (name=...): bedroom_target_temperature, hot_water_target_temperature, bedroom_light, bathroom_light.

В settings нужно добавить пременные SMART_HOME_API_URL и SMART_HOME_ACCESS_TOKEN и задать их значения, затем использовать их для взаимодействия с умным домом. Еще можно добавить EMAIL_RECEPIENT, в котором задать получателя писем от системы.

В приложенных исходниках есть несколько тестов django tests, которые тестируют некоторые базовые вещи. После того, как реализуете функционал, обазательно запустите manage.py test. На бою мы проверяем задание аналогичным образом через тесты контроллера по урлу /, и ручной вызов core.tasks.smart_home_manager.

Что будем проверять

Во время разработки вы можете в реальном времени наблюдать на сайте за реакцией умного дома на управляющие воздействия вашего приложения, а также руками выставлять показания датчиков для моделирования различных ситуаций.

Во время проверки на сервере интернет будет недоступен, requests.get будет заменен на mock, который будет возвращать приложению разные состояния контроллера (показания датчиков), а тесты будут проверять результат работы в requests.post и факт отправки письма, если оно требуется. Также будет проверяться работа веб-формы. Например если будет передана низкая температура горячей воды, в ответ будет ожидаться команда на включение бойлера, если он выключен. Но если к предыдущему примеру добавить еще и обнаруженную протечку, или пожар, то команда на включение бойлера подаваться не должна, а наоборот бойлеру должна подаваться команда на выключение, если он включен. Все возможные пограничные ситуации описаны в разделе “Реакция на события”. В случае если внешний сервер вернул ошибку или не отвечает нужно вернуть страницу с ошибкой со статус кодом 502.

Вам необходимо реализовать весь проект основываясь на скелете приложенном к этому заданию. Весь код должен находиться в приложении **core**. После реализации **заархивируйте содержимое папки core в zip архив и отправьте на проверку**.

Реализовывать валидацию данных можно с использованием библиотек

- marshmallow
- jsonschema

или с помощью класса Forms. Также в проекте установлен фреймворк для тестирования py.test
https://docs.pytest.org/en/latest/