Почти любое современное приложение получает данные из сторонних источников (базы данных, сторонние API, S3-хранилища...) и умеют в авторизацию/регистрацию. Разберем подробнее, как добавить эти вещи к себе.

Авторизация

Общее слово для процесса, состоящего из трех частей:
идентификация → аутентификация → авторизация.

Идентификация проверяет, предоставлены ли данные и валидны ли они.

Аутентификация проверяет, правильны ли эти данные, например: есть ли такая пара логин/пароль в базе данных.

Авторизация проверяет, если ли у пользователя с этими данными доступ к запрашиваему ресурсу.

Cookie Auth

Чаще всего для авторизации пользователей используется метод, построенный на куки-файлах.

Разберем упрощенный алгоритм, который будет использоваться в задании:

1. Пользователь отправляет пару логин/пароль на /user/login
2. Сервер ищет совпадение предоставленной пары в базе данных или в конфигурационных файлах и находит ее
3. Сервер собирает нужные данные о пользователе в объект и превращает его в json
4. Сервер шифрует json-данные симметричным шифрованием
5. Сервер добавляет заголовок Set-Cookie: <название куки>=<зашифрованные данные> и отправляет ответ
6. Браузер пользователя сохраняет запись из заголовка Set-Cookie в свое хранилище
7. Браузер пользователя отправляет заголовок Cookie с этими данными при каждом следующем обращении
8. Сервер достает данные из заголовка Cookie и дешифрует их своим ключом
9. При успешной расшифровке сервер убеждается, что этот пользователь авторизован, ведь ключ для шифрования есть только у сервера

Для работы по данному алгоритму в aiohttp существует библиотека для работы с сессиями aiohttp_session. Библиотека умеет хранить данные о сессии в разных типах хранилищ:

- SimpleCookieStorage хранит информацию об авторизованом пользователе прямо в теле cookie value. Предназначено только для удобства тестирования и крайне небезопасно во всех остальных случаях
- EncryptedCookieStorage хранит информацию так же, как SimpleCookie, но в зашифрованном виде. Используется симметричное шифрование 32-байтным ключом. Этот вариант необходимо использовать в домашней работе
- RedisStorage, MemcachedStorage, MongoStorage — хранит информацию в соответствующих хранилищах, а в cookie value хранит только уникальный ключ для ее получения. Требует настройку окружения

1. Авторизация пользователя. При запросе метода авторизации пользователя с правильными логином/паролем необходимо проставлять cookie в ответе. Это легко сделать с помощью метода aiohttp-session new_session, который сам зашифрует и проставит необходимую cookie. Объект сессии можно использовать как обычный словарь, в который можно положить данные, чтобы затем они оказались в cookie value.

Пример кода:

```
session = await new_session(request=self.request)
session['manager'] = manager_data
```

2. Вынесение логики для получения авторизации клиента в middleware. Необходимо, чтобы не дублировать логику в каждом View. Получить данные, с которыми авторизован запрос, можно с помощью метода get_session(request). Он проверит существование cookie с нужным именем, расшифрует ее содержимое и вернет его:

```
session = await get_session(request)
```

3. Запрещать доступ к некоторым View неавторизованным пользователям. Эту логику можно реализовать несколькими способами:

1. добавить обертку на уровне route
2. добавить middleware, которая работает после авторизационной
3. добавить mixin, который расширяет логику базового View. *AuthRequiredMixin уже реализован в следующем домашнем задании*
4. добавить декоратор на базовый View
5. вручную проверять авторизацию в каждом нужном методе View
6. и так далее

Выбирайте, что вам больше нравится. По моему опыту, лучше управлять логикой авторизованной зоны на уровне View, так что я бы предложил 3-й или 4-й вариант.

Примеры

1. Авторизация с помощью aiohttp-session