E2E - homework12

Тестирование E2E (End-to-End) включает в себя создание тестовых сценариев, которые имитируют реальные действия пользователей от начала и до конца работы функционала. В этом случае тесты будут охватывать взаимодействие между компонентами `TokenAdapter`, `Controller` и `Repo` для обеспечения авторизации пользователя и управления токенами.

Пример протокола E2E тестирования:

1. **Наименование теста**: Тест авторизации пользователя и управления токеном
2. **Цель теста**: Проверить процесс создания, проверки и удаления токена для авторизации пользователя и предоставления доступа к роботу.
3. **Описание планируемого взаимодействия**:
   - Пользователь запрашивает токен через `TokenAdapter`.
   - `TokenAdapter` получает токен от `Repo`.
   - Пользователь использует токен для аутентификации через `Controller`.
   - `Controller` взаимодействует с `TokenAdapter` для проверки токена.
   - После успешной аутентификации токен отправляется роботу.
   - `Controller` использует `TokenAdapter` для удаления токена после его использования.
4. **Тестовые шаги**:
   a. Создание токена.
   b. Аутентификация с использованием токена.
   c. Добавление токена к роботу.
   d. Проверка валидности токена.
   e. Удаление токена.
5. **Ожидаемые результаты**:
   - Токен успешно создается и сохраняется в `Repo`.
   - `TokenAdapter` возвращает токен при запросе.
   - `Controller` подтверждает валидность токена и предоставляет доступ к роботу.

- Токен успешно отправляется роботу.
- Токен успешно удаляется после использования.

6. **Критерии успеха**:
   - Все компоненты системы работают согласованно.
   - Нет ошибок во время взаимодействия между компонентами.
   - Токен успешно проходит все стадии своего жизненного цикла.

Unit тесты для компонентов системы:

Для `TokenAdapter`:

1. **Тест получения токена**:
   - Вызов `getToken(User)` должен возвращать токен.
   - Проверка, что `Repo.createToken` вызывается с корректными параметрами.
2. **Тест проверки токена**:
   - Вызов `checkToken(Token)` с валидным токеном возвращает `true`.
   - Вызов `checkToken(Token)` с невалидным токеном возвращает `false`.
3. **Тест отправки токена роботу**:
   - Вызов `sendTokenToRobot(Token, Robot)` должен возвращать `true` при успешной отправке.
   - Вызов `sendTokenToRobot(Token, Robot)` должен возвращать `false` при неудаче.
4. **Тест удаления токена**:
   - Проверить, что `deleteToken(Token)` корректно вызывает метод `Repo.deleteToken`.

Для `Controller`:

1. **Тест проверки аутентификации**:
   - `checkAuth(User, Robot)` должен возвращать `true` при успешной аутентификации.
   - `checkAuth(User, Robot)` должен возвращать `false` при неудачной попытке аутентификации.

2. **Тест добавления токена к роботу**:
    - `addTokenToRobot(Token, Robot)` должен вызывать `TokenAdapter.sendTokenToRobot`.
3. **Тест удаления токена**:
    - `deleteToken(Token)` должен

вызывать `TokenAdapter.deleteToken`.

Для `Repo`:

1. **Тест создания токена**:
    - `createToken(string)` создает токен с заданными параметрами.
2. **Тест чтения токена**:
    - `readToken(int)` возвращает корректный токен для заданного ID.
3. **Тест обновления токена**:
    - `updateToken(params)` обновляет токен с заданными параметрами.
4. **Тест удаления токена**:
    - `deleteToken(int)` удаляет токен с заданным ID.

При написании unit тестов стоит использовать mock-объекты для имитации зависимостей и проверки взаимодействия между классами без необходимости в реальном выполнении операций (например, обращений к базе данных).