Как мы автоматизировали тесты с несколькими пользователями

One way to rule them all

Обо мне

- Ванчук Василий @vvscode
- Разрабатываю на JS
- Люблю автоматизацию

Кому доклад будет полезен

- Вы автоматизатор?
- Проект, объединяющий пользователей?
- Любите истории про чужой опыт?

Для понимания

- Код и примеры будут в терминах JS
- Для понимания кода достаточно базовых знаний любого ООП языка
- Я постараюсь сделать упор на принцип, а не на конкретные реализации

Top-Down Test Design

01 Собрать тестовые сценарии

02 Записать их на псевдокоде

03 Подобрать интерфейсы, подходящие под псевдокод

04 Реализовать интерфейсы

Прелюдия

Контекст, в котором началась история

Про проект

- Сервис по доставке еды
- Приложение для курьеров доставки
- 3 (даже 4) участника процесса
- 2 из них должны оставаться онлайн большую часть теста
- Мобильные приложений на React Native
- Для полного тестирования нужно участие нескольких пользователей

Что делать?

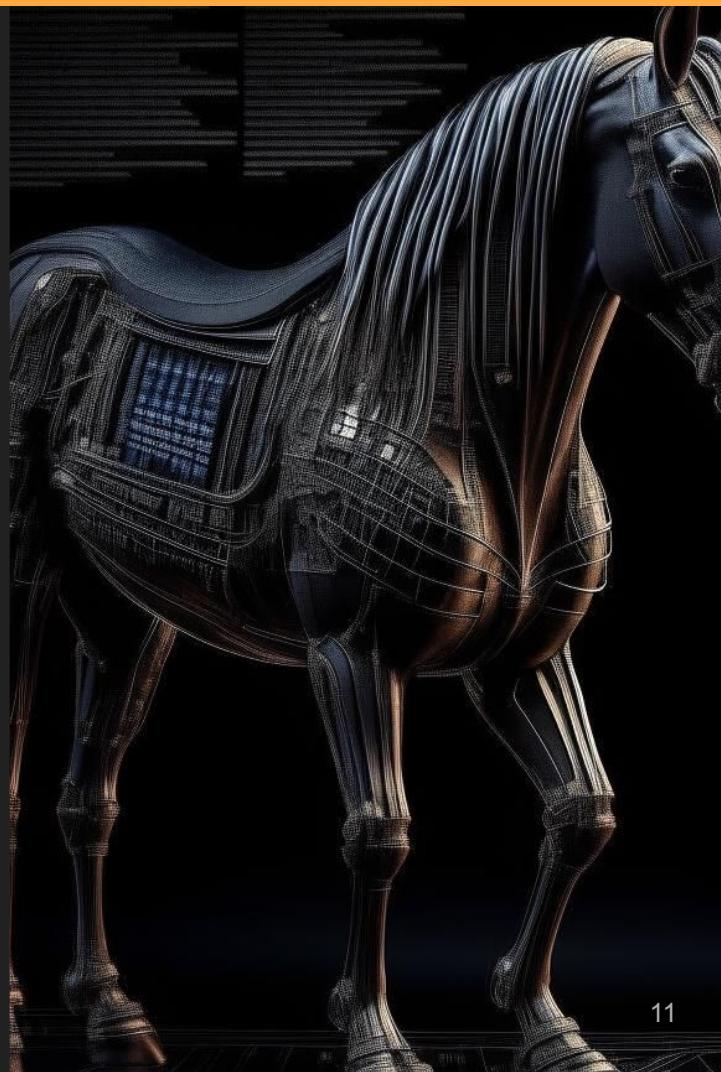
Какие есть варианты

Поднимать несколько клиентов

- Медленно
- Дорого
- Ненадежно (flaky[3])

Написать маковый сервер

- Сервер для тестирования, с упрощенной логикой
- Полный контроль над его состоянием
- Дополнительная работа по написанию и поддержке
- Решает вопрос с тестированием приложений, но не e2e тестами



Record’n’Play

- Записывает тесты на живом API (все запросы и ответы)
- В CI используем вместо API записанные данные (а не реальное API)
- Нужны модификаторы для time-sensitive данных
- Тестируется только приложение (сервер только в момент записи теста)
- Проблемы с обновлением API (все тесты нужно заново прогнать на живом сервере)

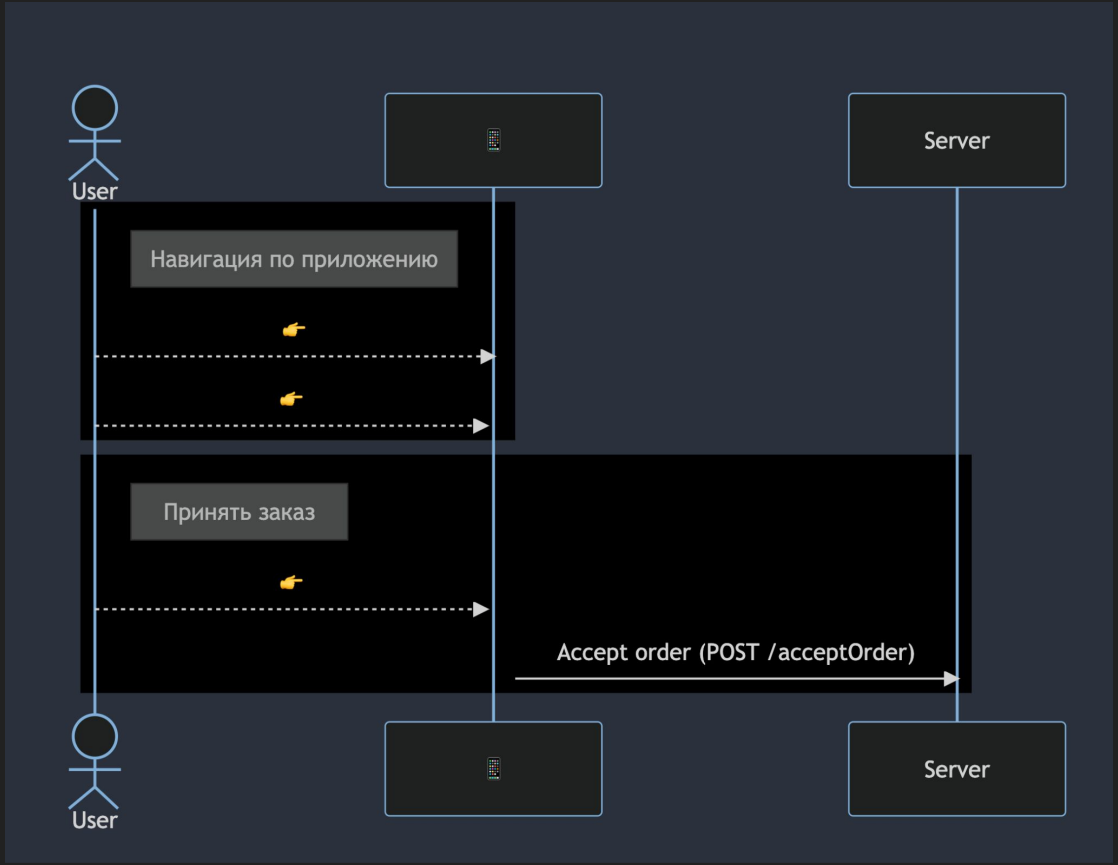
Тестовые ручки

- Тестовые ручки на живой сервер (доступные только в тестовом окружении)
- Код, который нужно писать и поддерживать
- В тестах мы тестируем тестовую (а не реальную) логику
- Код усложняется (в т.ч. для модульного тестирования)

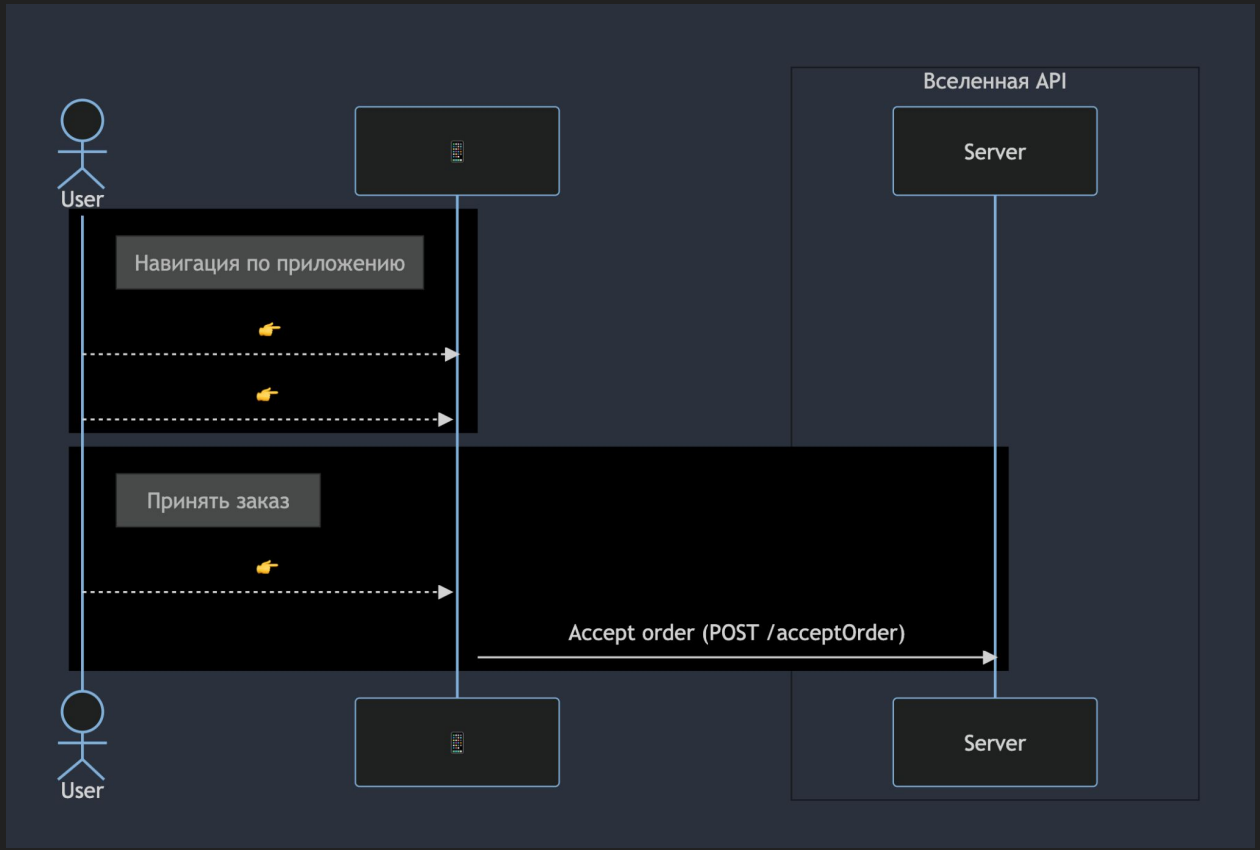
Симуляция

Утиный тест: оценка по кряку, не углубляясь в технические детали

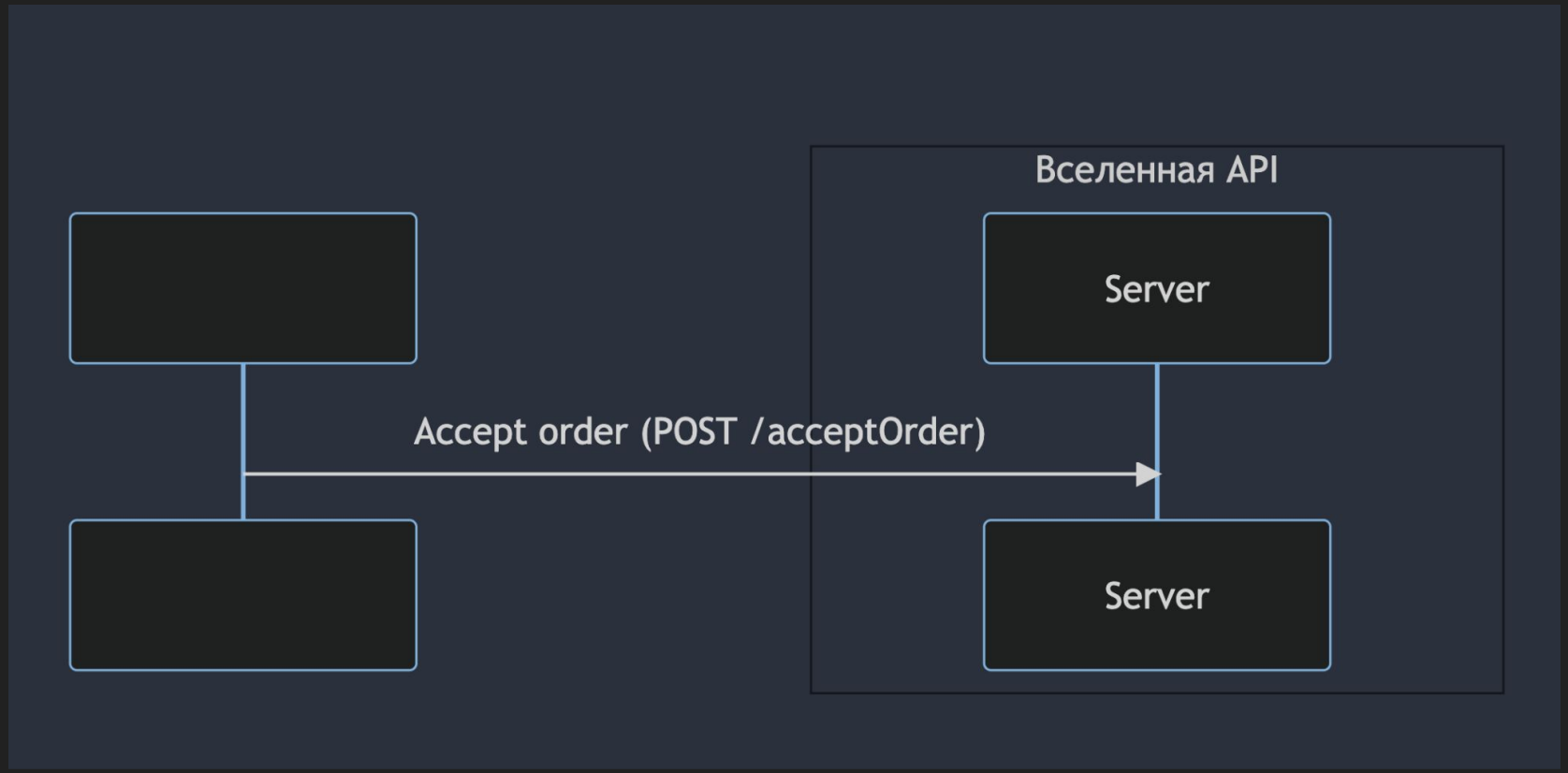
Как выглядит работа клиента



Что знает сервер?

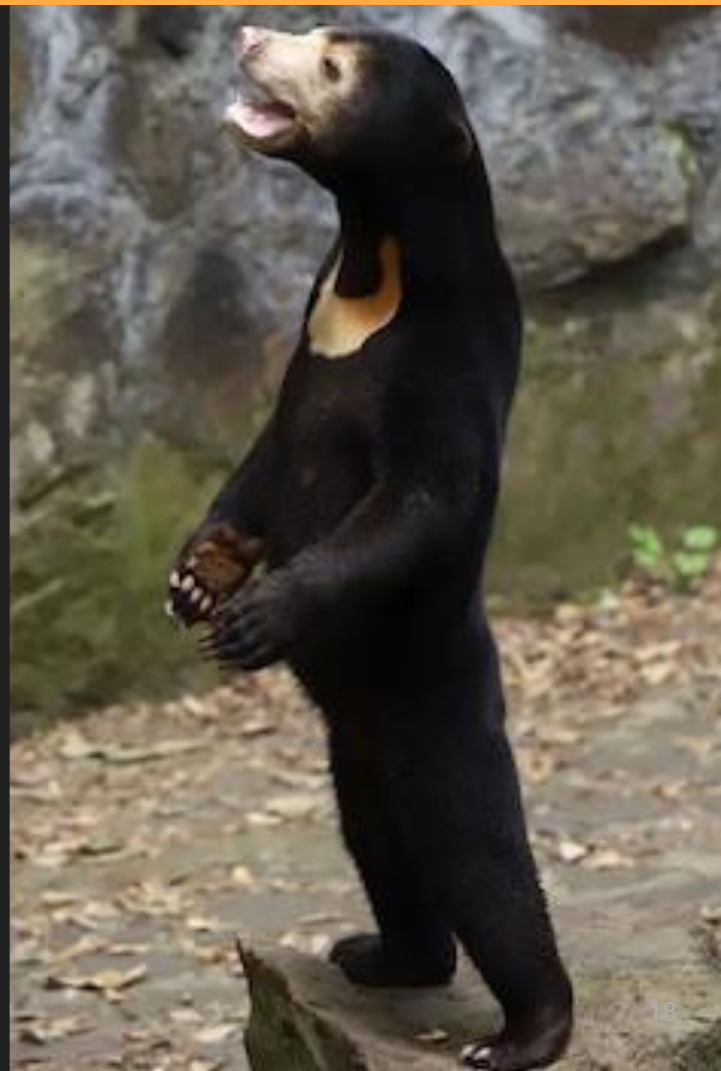


Сервер знает об окружающем мире - только то, что приходит в запросах



Fake it

- Технически нам просто нужно отправлять те же последовательности запросов с теми же данными
- Сервер даже не узнает о подмене



Помни про аутентификацию

Нередко делается на базе access/refresh токенов (с коротким, даже по меркам тестов, сроком жизни), так что периодически их нужно обновлять

Нужно держать в голове

Часто нас интересует не только отправка запросов (и получение ответов), но и получение событий от него (или от других пользователей посредством сервера):

- Long polling
- Short polling
- Server side events
- Websockets

Абстракт

Сделать легко — осталось упростить

Page Object

- Изолирует страницы
- Абстрагирует детали интерфейса
- Повышает читаемость
- Снижает зависимость от изменений в UI

- Контролирует конкретные способы достижения целей (проверки и действия)

User Object

- Изолирует действующих лиц
- Абстрагирует способы действий
- Создает единый язык
- Снижает зависимость от изменений в приложении

- Контролирует шаги в бизнес процессах

User Object

Искусство быть уткой

Реализация User Object

- Stateful object (Class)
- В тестах будем инициировать нужные сущности и менять их состояние по ходу теста

```
it.withActors('can decline incoming order', async ({ actors }) => {
  await Promise.all([
    actors.client.goOnline(), // <-- клиент
    actors.resto.goOnline(), // <-- ресторан
  ])

  await App.login()
  await App.goOnline()

  await actors.client.putAnOrder() // <-- клиент
  await actors.resto.acceptOrder() // <-- ресторан

  await HomeScreen.isIncomingOrderVisible()
  await HomeScreen.isDeclineButtonVisible()

  await HomeScreen.declineOrder()

  await HomeScreen.not.isIncomingOrderVisible()
  await HomeScreen.not.isDeclineButtonVisible()
})
```

Недостатки подхода

- Это код, который нужно писать*
- Надежность

Достоинства подхода

- Скорость тестов
- Надежность
- Тестирование API
- Автоматизации

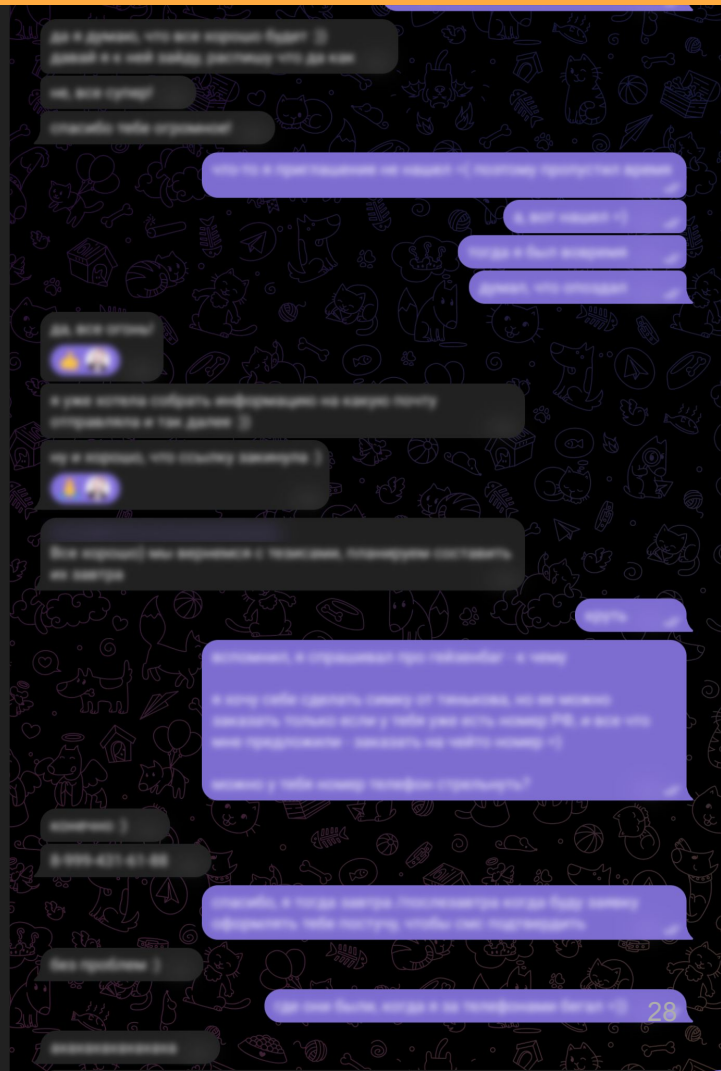
Упрощенный пример

Чатлане: от простого к сложному

Чат?

- Почти любое взаимодействие можно свести к чату
- Действующие лица одного типа
- Простой интерфейс (базовые операции)
- Минимум кода, полный концепт

- С одним отличием от реальной ситуации



Тестовые сценарии

Для рассмотрения возьмем минимальный пример, достаточный для рассмотрения

- Есть два пользователя
- Они могут обмениваться сообщениями

```
test(
  'Bob sends a message, and Alice receives it',
  () => {
    test.step('Bob logins', () => {})
    test.step('Alice logins', () => {})

    test.step('Bob sends message', () => {})
    test.step('Alice receives message', () =>
{}}
)
```

```
test(
  'Alice sends a message, and Bob receives it',
  () => {
    test.step('Bob logins', () => {})
    test.step('Alice logins', () => {})

    test.step('Alice sends message', () => {})
    test.step('Bob receives message', () => {})
  }
)
```

User Object

- Если нас интересует отправка через DM - нам нужен способ узнать Id экземпляра
- .login() для входа в систему (аутентификации)
- Методы сделаны асинхронными
  - Особенности JS
  - В реальности действующие лица могут взаимодействовать с системой параллельно

```
interface ChatUserObjectModel {
    userId: any;

    login(): Promise<unknown>;

    sendMessage(
      toId: ChatUserObjectModel['userId'],
      message: string
    ): Promise<unknown>;

    checkMessages(
      mask: string
    ): Promise<string[]>;

    getUserId(
    ): Promise<ChatUserObjectModel['userId']>;
}
```

User Object

- .kill() для очистки состояния
- И заодно добавим тип, описывающий конструктор класса*

```
interface ChatUserObjectModel {
  userId: any;

  login(): Promise<unknown>;

  sendMessage(
    toId: ChatUserObjectModel['userId'],
    message: string
  ): Promise<unknown>;

  checkMessages(
    mask: string
  ): Promise<string[]>;

  getUserId(
  ): Promise<ChatUserObjectModel['userId']>;

  // technical method
  kill(): Promise<unknown>;
}


type ChatUserObjectModelConstructor =
  new <T>(name: string) => ChatUserObjectModel;
```

Как такой UO будет выглядеть в тестах?

```
let bob, alice

test.beforeEach(() => {
  bob = new ChatUserObjectModel(bobName)
  alice = new ChatUserObjectModel(aliceName)
})

test('Bob sends a message, and Alice receives it', async () => {
  await test.step('Bob logins', async () => {
    await bob.login()
  })
  await test.step('Alice logins', async () => {
    await alice.login()
  })

  await test.step('Bob sends message', async () => {
    await bob.sendMessage(await alice.getUserId(), 'Hello from Bob')
  })
  await test.step('Alice checks message', async () => {
    const messages = await alice.checkMessages('Hello from Bob')
    test.expect(messages).toContain('Hello from Bob')
  })
})

```

Выбор Реализации

- После соглашения об интерфейсе UO мы можем обсудить реализации
- Мы можем взаимодействовать с UI
- Мы можем отправлять запросы на API напрямую
- Мы можем смешивать оба подхода

```
class UIChatUO
  implements ChatUserObjectModel {
  // работа со страницей и DOM
}

class APIChatUO
  implements ChatUserObjectModel {
  // работа с API запросами
}

class MixChatUO
  implements ChatUserObjectModel {
  // вариации смешаной реализации
}
```

UI Реализация

- User Object не отменяет Page Object
- В большинстве случаев мы все равно будем писать PO (для проверки конкретных способов достижения цели)
- В таком случае UO может быть реализован композицией Page Object’ов

```
class CometWithAuthUIObject implements ChatUserObjectModel {
  private browser?: Browser;
  private page?: Page;
  private chatPageObject?: ChatPageObject;

  private name: string;
  constructor(name: string) {
    this.name = name;
    this.userId = name;
  }

  login = async () => {
    const page = await this.getPage();
    const authPage = new AuthFormPageObject(page);

    await authPage.open();
    await authPage.fillLogin(this.name);
    await authPage.fillPassword();
    await authPage.submit();
    await authPage.goToChat();

    this.chatPageObject = new ChatPageObject(page);
    await this.chatPageObject.open();
  };
  // ...
}
```

API реализация

Стоит помнить (помимо stateful):

- Cookies
- Обработка редиректов
- Заголовки запросов
- Может использовать смесь подходов и протоколов

- Все это можно разрешать вручную, но проще взять инструменты, которые поддерживают автоматическое разрешение

```
class UserObject implements ChatUserObjectModel {
  constructor(apiUrl: string) {
    this.apiUrl = apiUrl;
  }

  async login(): Promise<unknown> {
    // Реализация метода login
  }

  async sendMessage(toId: string, message: string): Promise<unkn
    return this.sendRequest('send-message', { toId, message });
  }

  async checkMessages(mask: string): Promise<string[]> {
    // Реализация метода checkMessages
  }

  async getUserId(): Promise<string> {
    // Реализация метода getUserId
  }

  async kill(): Promise<unknown> {
    // Реализация метода kill
  }

  private async sendRequest(endpoint: string, data: any): Promis
    // Использование библиотеки для выполнения запросов
  }
}
```

К чему это все

Что с этим делать?

Смешивать UO и PO

```
// Мы можем смешивать User Object и Page Object

// UO используется для действий со стороны других пользователей
bob.sendMessage("Hello from Bob");

// PO используется для проверок в нашем тестируемом приложении
expect(alice.chatPage.isNotificationSoundPlayed()).toBe(true);
```

Не забывай свои корни - назад в контекст

```
// Мы можем смешивать User Object и Page Object

// UO используется для действий со стороны других пользователей
await client.placeOrder(orderDetails)

// PO используется для проверок в нашем тестируемом приложении
expect(await courierApp.homeScreen.isIncomingOrderBannerVisible()).toBe(true);
expect(await courierApp.system.isSoundPlaying()).toBe(true);
```

Мы можем проверять работу системы с точки зрения пользователей

```
test('Bob sends a message, and Alice receives it', async () => {
  await test.step('Bob logins', async () => await bob.login())
  await test.step('Alice logins', async () => await alice.login())

  await test.step(
    'Bob sends message',
    async () => await bob.sendMessage(await alice.getUserId(), 'Hello from Bob')
  )
  await test.step('Alice checks message', async () => {
    const messages = await alice.checkMessages('Hello from Bob')
    test.expect(messages).toContain('Hello from Bob')
  })
})
```

А еще - следите за руками

```
1 // Вспомним, что наш UO - это обобщенный интерфейс
2 // И что у нас есть тип описывающий конструктор объектов такого интерфейса
3 // Давайте немного обобщим наш тест с учетом этого типа
4 function runChatTests(
5   ActorModel1: ChatUserObjectModelConstructor,
6   ActorModel2: ChatUserObjectModelConstructor) {
7
8   let bobName: string;
9   let aliceName: string;
10
11  let bob: ChatUserObjectModel;
12  let alice: ChatUserObjectModel;
13
14  test.beforeEach(async ({ page }) => {
15    test.step(`Generate user names (${bobName}, ${aliceName})\n`, () => {});
16
17    await test.step('Create actors', () => {
18      bob = new ActorModel1(bobName);
19      alice = new ActorModel2(aliceName);
20    });
21  });
22
23  // ниже наши тесты
24 }
25
```

У нас целая ферма уток

```
(
  [
    [UIChatUO, UIChatUO],
    [APIChatUO, UIChatUO],
    [UIChatUO, APIChatUO],
    [APIChatUO, APIChatUO],
  ] as [ChatUserObjectModelConstructor, ChatUserObjectModelConstructor][]
).forEach((pair) => runChatTests(...pair));
```

Использовать UO для автоматизаций

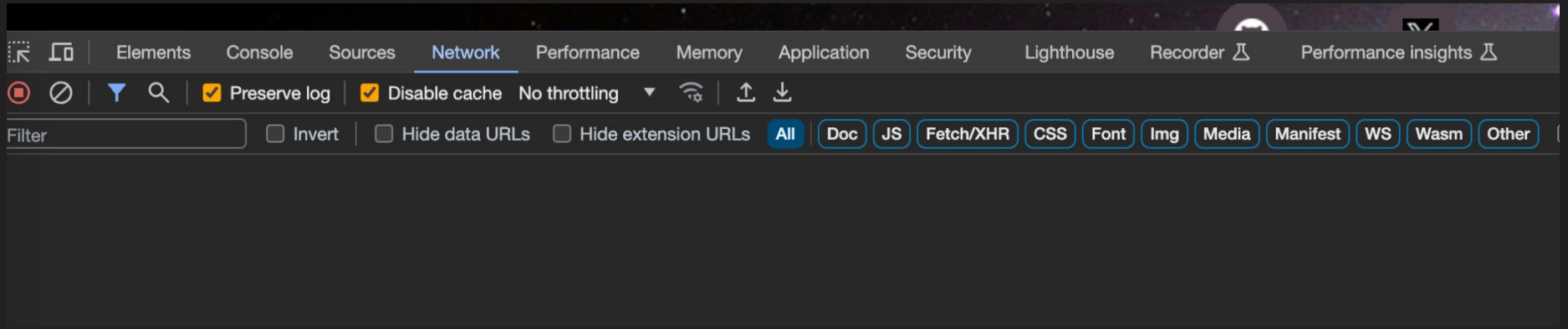
- Подготовка системы для ручного тестирования
- Прогон тестов для проверки статуса API
- Автоматизация работы операционистов

```
await resto.login();
await client.login();

await resto.goOnline();
await client.makeOrder(orderDetails);

// manual test for courier app here
```

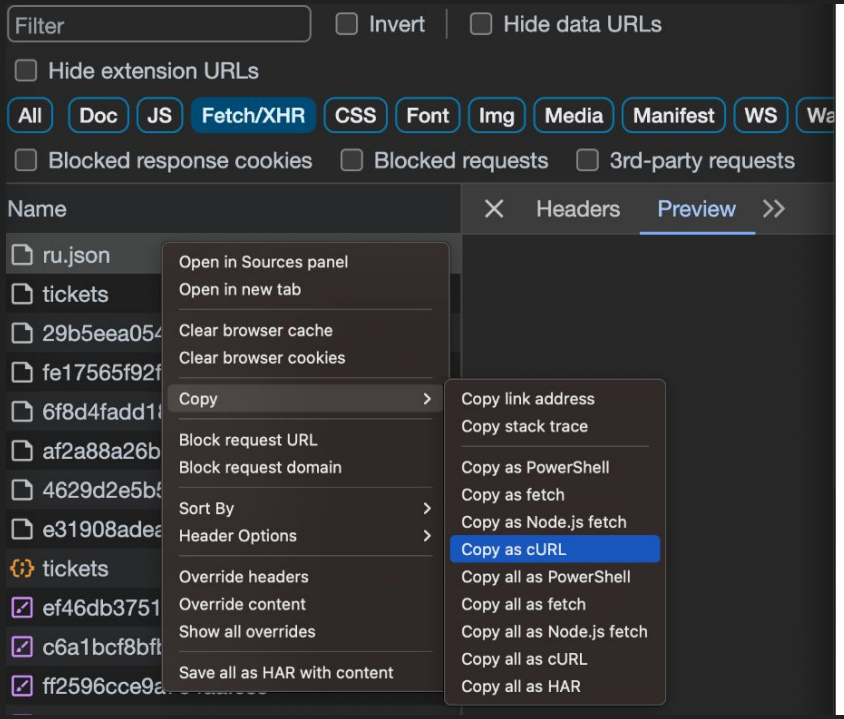
Практика

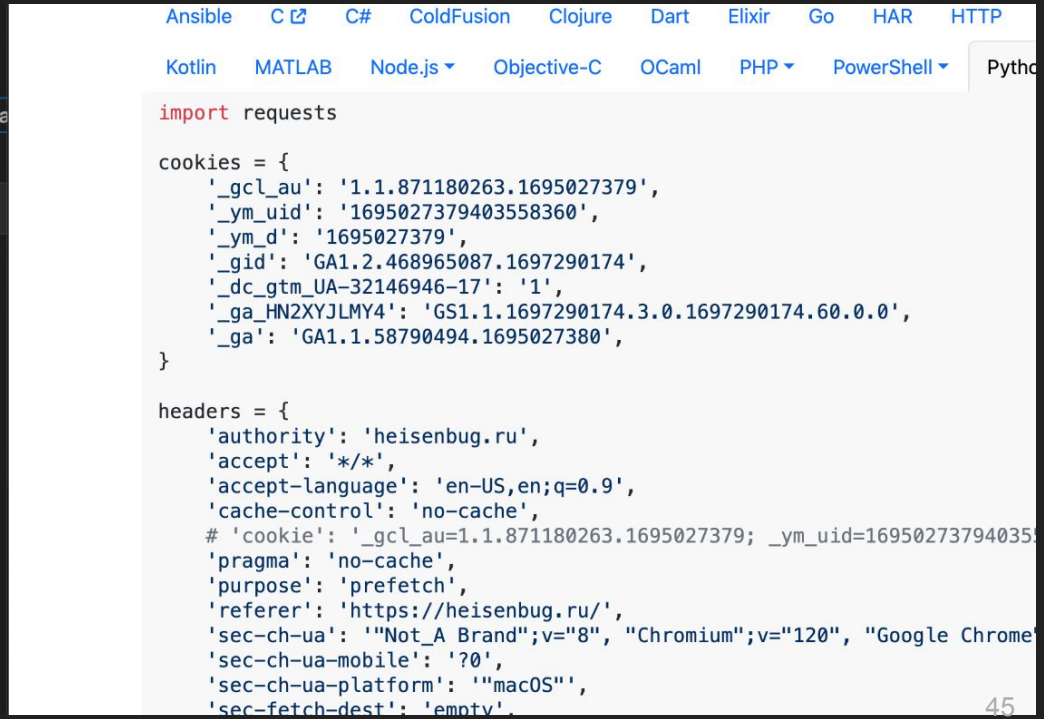
Здесь Родос, здесь прыгай

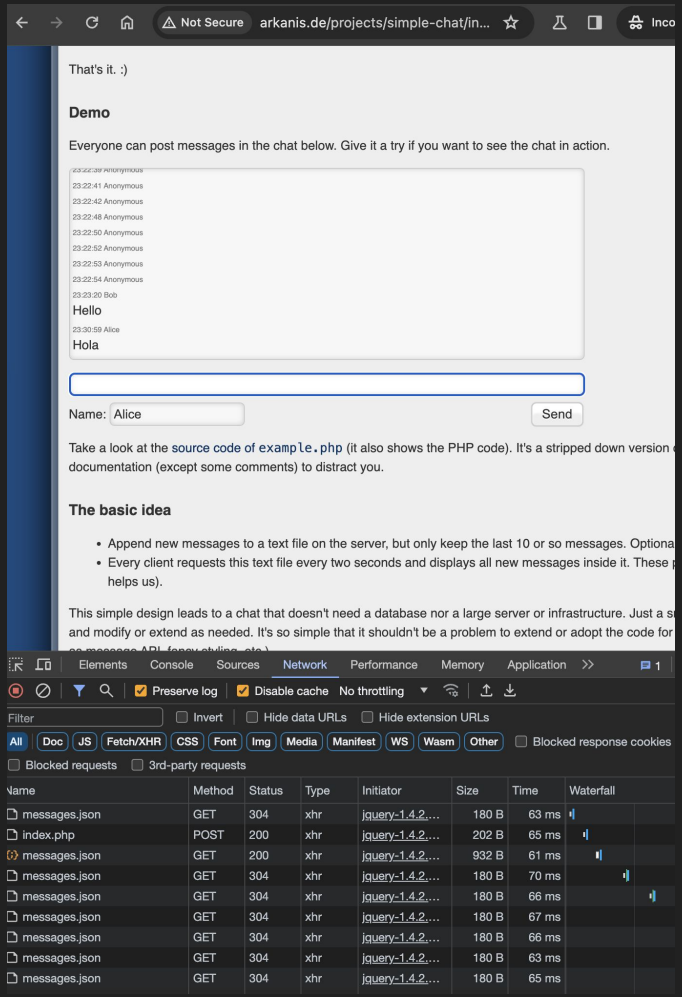
Инструменты разработчика



curlconverter






Not Secure
arkanis.de/projects/simple-chat/in...
That's it. :)
Demo
Everyone can post messages in the chat below. Give it a try if you want to see the chat in action.
23:22:41 Anonymous
23:22:42 Anonymous
23:22:48 Anonymous
23:22:50 Anonymous
23:22:52 Anonymous
23:22:53 Anonymous
23:22:54 Anonymous
23:23:20 Bob
Hello
23:30:59 Alice
Hola
Name: Alice
Send
Take a look at the source code of example.php (it also shows the PHP code). It's a stripped down version
documentation (except some comments) to distract you.
The basic idea
Append new messages to a text file on the server, but only keep the last 10 or so messages. Optiona
Every client requests this text file every two seconds and displays all new messages inside it. These
helps us).
This simple design leads to a chat that doesn't need a database nor a large server or infrastructure. Just a s
and modify or extend as needed. It's so simple that it shouldn't be a problem to extend or adopt the code for
Elements
Console
Sources
Network
Performance
Memory
Application
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
Hide extension URLs
All
Doc
JS
Fetch/XHR
CSS
Font
Img
Media
Manifest
WS
Wasm
Other
Blocked response cookies
Blocked requests
3rd-party requests
Name
Method
Status
Type
Initiator
Size
Time
Waterfall
messages.json GET 304 xhr jquery-1.4.2... 180 B 63 ms
index.php POST 200 xhr jquery-1.4.2... 202 B 65 ms
messages.json GET 200 xhr jquery-1.4.2... 932 B 61 ms
messages.json GET 304 xhr jquery-1.4.2... 180 B 70 ms
messages.json GET 304 xhr jquery-1.4.2... 180 B 66 ms
messages.json GET 304 xhr jquery-1.4.2... 180 B 67 ms
messages.json GET 304 xhr jquery-1.4.2... 180 B 66 ms
messages.json GET 304 xhr jquery-1.4.2... 180 B 63 ms
messages.json GET 304 xhr jquery-1.4.2... 180 B 65 ms

Simple chat

ChatUserObject via UI

```
export class SimpleChatUIObject implements ChatUserObjectModel {
  userId: string;

  private browser!: Browser;
  private page!: Page;
  private name: string;

  constructor(name: string) {
    this.name = name;
    this.userId = name;
  }

  login = async () => {
    await this.initPage();
    await this.page.goto('http://arkanis.de/projects/simple-chat/#demo');
  };

  sendMessage = async (to: string, message: string) => {
    await this.page.fill('input[name="content"]', message);
    await this.page.fill('input[name="name"]', this.name);
    await this.page.click('button[type="submit"]');
  };

  checkMessages = async (mask: string) => {
    const messages = await this.page.$$eval('ul[id=messages] li', (list) =>
      [...list].map((item) => `${item.childNodes[1].textContent}`),
    );
    return messages.filter((el) => el.includes(mask));
  };

  getUserId = async () => this.name;

  kill = async () => {
    await this.page.close();
    await this.browser.close();
  };

  private async initPage() {
    this.browser = await chromium.launch({
      headless: false,
    });
    this.page = await this.browser.newPage();
  }
}
```

ChatUserObject via UI

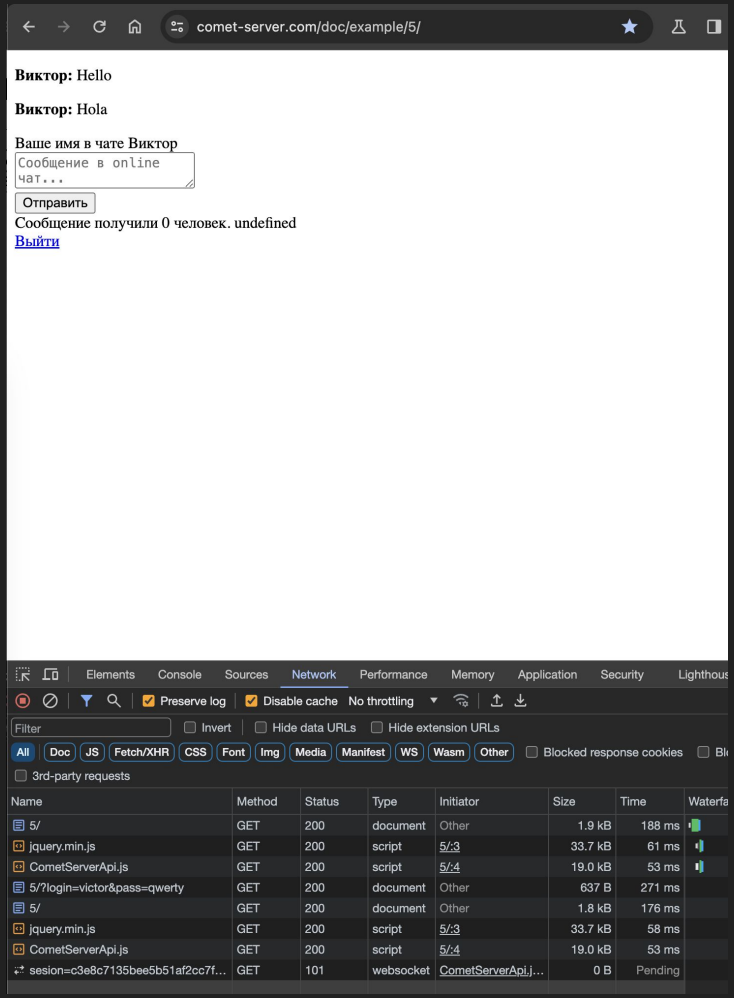
```
13      this.name = name;
14      this.userId = name;
15    }
16
17    login = async () => {
18      await this.initPage();
19      await this.page.goto('http://arkanis.de/projects/simple-chat/#demo');
20    };
21
22    sendMessage = async (to: string, message: string) => {
23      await this.page.fill('input[name="content"]', message);
24      await this.page.fill('input[name="name"]', this.name);
25      await this.page.click('button[type="submit"]');
26    };
27
28    checkMessages = async (mask: string) => {
29      const messages = await this.page.$$eval('ul[id=messages] li', (list) =>
30        [...list].map((item) => `${item.childNodes[1].textContent}`),
31      );
32      return messages.filter((el) => el.includes(mask));
33    };
34
```

ChatUserObject via API

```
export class SimpleChatAPIObject implements ChatUserObjectModel {
  userId: string = '';

  private name: string;
  private timer: NodeJS.Timeout;

  private messages: {
    from: string;
    message: string;
  }[] = [];

  constructor(name: string) {
    this.name = name;
    this.timer = setInterval(this.fetchMessages, FETCH_PERIOD_MS);
  }

  login = async () => {};

  sendMessage = async (to: string, message: string) => {
    return await axios.post(
      'http://arkanis.de/projects/simple-chat/index.php',
      new URLSearchParams({
        name: this.name,
        content: message,
      }),
    );
  };

  fetchMessages = async () => {
    const response = await axios.get('http://arkanis.de/projects/simple-
chat/messages.json');
    this.messages = response.data.map((el: any) => ({
      from: el.name,
      message: el.content,
    }));
  };

  checkMessages = async (mask: string) => {
    await sleep(FETCH_PERIOD_MS);
    return this.messages.map((el) => el.message).filter((el) =>
el.includes(mask));
  };

  getUserId = async () => this.name;

  kill = async () => {
    clearTimeout(this.timer);
  };
}
```

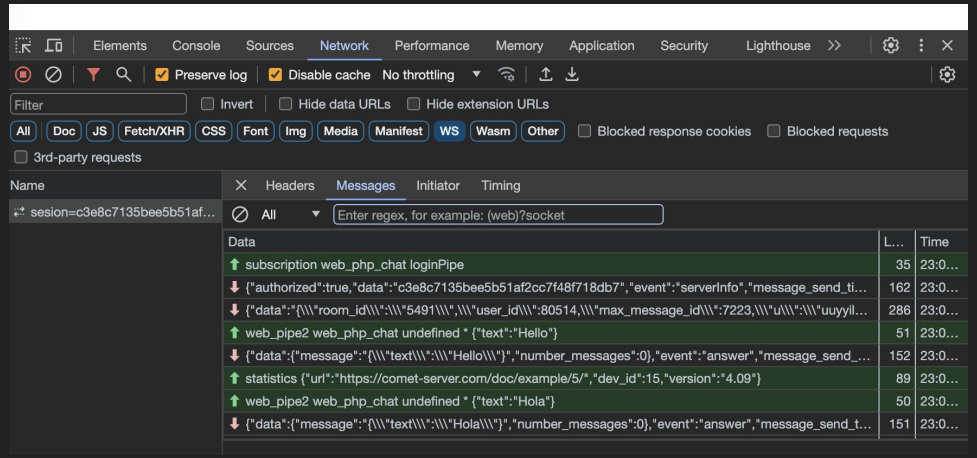
ChatUserObject via API

```
17
18  constructor(name: string) {
19    this.name = name;
20    this.timer = setInterval(this.fetchMessages, FETCH_PERIOD_MS);
21  }
22
23  login = async () => {};
24
25  sendMessage = async (to: string, message: string) => {
26    return await axios.post(
27      'http://arkanis.de/projects/simple-chat/index.php',
28      new URLSearchParams({
29        name: this.name,
30        content: message,
31      }),
32    );
33  };
34
35  fetchMessages = async () => {
36    const response = await axios.get('http://arkanis.de/projects/simple-
    chat/messages.json');
37    this.messages = response.data.map((el: any) => ({
38      from: el.name,
39      message: el.content,
40    }));
41  };
42
43  checkMessages = async (mask: string) => {
44    await sleep(FETCH_PERIOD_MS);
45    return this.messages.map((el) => el.message).filter((el) =>
    el.includes(mask));
46  };
47
```



[Comet chat with Auth]

Comet chat with Auth



PO как часть UO

```ts
import type { Page } from 'playwright';

export class AuthFormPageObject {
  constructor(private page: Page) {}

  async open() {
    await this.page.goto('https://comet-server.com/doc/example/5/');
  }

  async fillLogin(login: string) {
    await this.page.fill('input[name=login]', login);
  }

  async fillPassword(password: string = 'qwerty') {
    await this.page.fill('input[name=pass]', password);
  }

  async submit() {
    await this.page.click('input[type=submit]');
  }

  async goToChat() {
    await this.page.click('a[href*="https://comet-server.com/doc/"]');
    await this.page.waitForURL((url) => !url.search.includes('login'));
  }
}
```

PO как часть UO

```ts
import type { Page } from 'playwright';

export class ChatPageObject {
  constructor(private page: Page) {}

  async open() {
    await this.page.goto('https://comet-server.com/doc/example/5/');
    await this.page.reload();
  }

  async fillMessage(message: string) {
    await this.page.fill('textarea[id="WebChatTextID"]', message);
  }

  async sendMessage() {
    await this.page.click('input[type="button"]');
  }

  async getMessages(): Promise<{ from: string; message: string }[]> {
    return await this.page.$$eval('div[id="WebChatFormForm"] p', (pList) =>
      [...pList].map((p) => ({
        from: p.childNodes[0].textContent!.replace(': ', ''),
        message: `${p.childNodes[1].textContent}`,
      })),
    );
  }
}
```

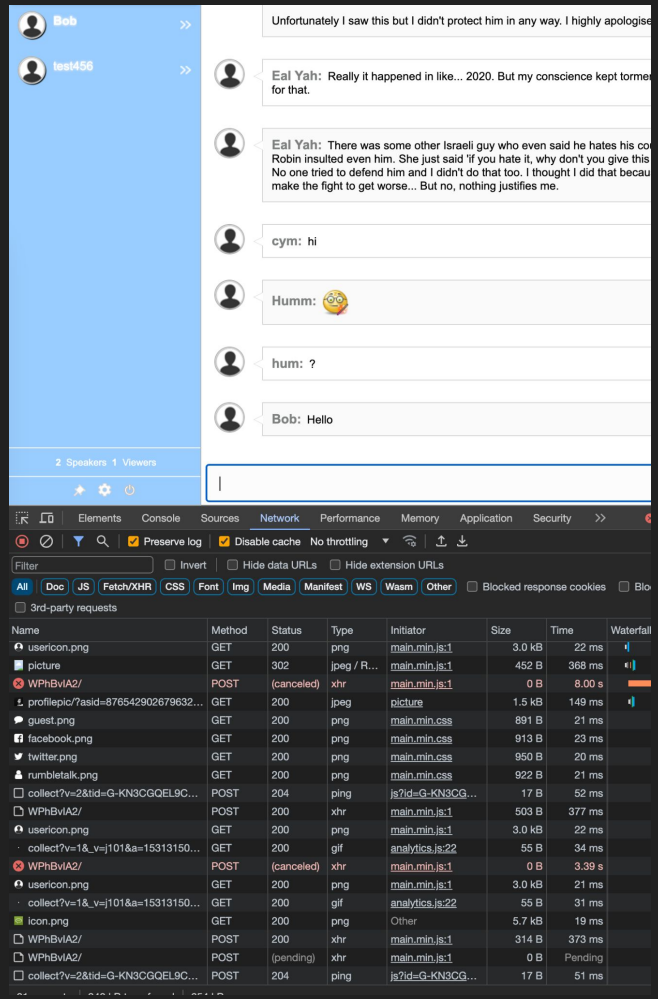
UO как композиция PO

```
export class CometWithAuthUIObject implements ChatUserObjectModel {
  userId: string;

  private browser?: Browser;
  private page?: Page;
  private chatPageObject?: ChatPageObject;

  private name: string;
  constructor(name: string) {
    this.name = name;
    this.userId = name;
  }

  login = async () => {
    const page = await this.getPage();
    const authPage = new AuthFormPageObject(page);
    await authPage.open();
    await authPage.fillLogin(this.name);
    await authPage.fillPassword();
    await authPage.submit();
    await authPage.goToChat();
    this.chatPageObject = new ChatPageObject(page);
    await this.chatPageObject.open();
  };

  sendMessage = async (to: string, message: string) => {
    if (!this.chatPageObject) {
      throw new Error('Login first');
    }
    await this.chatPageObject.fillMessage(message);
    await this.chatPageObject.sendMessage();
  };

  checkMessages = async (mask: string) => {
    if (!this.chatPageObject) {
      throw new Error('Login first');
    }
    await sleep();
    const chatHistory = await this.chatPageObject.getMessages();
    return chatHistory.map((el) => el.message);
  };

  getUserId = async () => this.name;

  kill = async () => {
    this.page?.close();
    this.browser?.close();
  };

  private async getPage() {
    if (this.page) {
      return this.page;
    }
    const browser = await chromium.launch({
      headless: false,
      executablePath: '/Applications/Google Chrome.app/Contents/MacOS/Google
Chrome',
    });
    this.page = await browser.newPage();
    return this.page;
  }
}
```

UO как композиция PO

```
19
20  login = async () => {
21    const page = await this.getPage();
22    const authPage = new AuthFormPageObject(page);
23    await authPage.open();
24    await authPage.fillLogin(this.name);
25    await authPage.fillPassword();
26    await authPage.submit();
27    await authPage.goToChat();
28    this.chatPageObject = new ChatPageObject(page);
29    await this.chatPageObject.open();
30  };
31
32  sendMessage = async (to: string, message: string) => {
33    if (!this.chatPageObject) {
34      throw new Error('Login first');
35    }
36    await this.chatPageObject.fillMessage(message);
37    await this.chatPageObject.sendMessage();
38  };
39
40  checkMessages = async (mask: string) => {
41    if (!this.chatPageObject) {
42      throw new Error('Login first');
43    }
44    await sleep();
45    const chatHistory = await this.chatPageObject.getMessages();
46    return chatHistory.map((el) => el.message);
47  };
48
49  getUserId = async () => this.name;
50
51  kill = async () => {
52    this.page?.close();
53    this.browser?.close();
54  };
```

https://github.com/vvscode/heisenbug-demo/blob/master/userObjectModels/CometWithAuthAPIObject.ts


Bob
test456
Unfortunately I saw this but I didn't protect him in any way. I highly apologise
Eal Yah: Really it happened in like... 2020. But my conscience kept tormer
for that.
Eal Yah: There was some other Israeli guy who even said he hates his co
Robin insulted even him. She just said 'if you hate it, why don't you give this
No one tried to defend him and I didn't do that too. I thought I did that becau
make the fight to get worse... But no, nothing justifies me.
cym: hi
Humm:
hum: ?
Bob: Hello
2 Speakers 1 Viewers
Elements Console Sources Network Performance Memory Application Security
Preserve log Disable cache No throttling
Filter Invert Hide data URLs Hide extension URLs
All Doc JS Fetch/XHR CSS Font Img Media Manifest WS Wasm Other Blocked response cookies
3rd-party requests
Name Method Status Type Initiator Size Time Waterfall
usericon.png GET 200 png main.min.js:1 3.0 kB 22 ms
picture GET 302 jpeg / R... main.min.js:1 452 B 368 ms
WPhBvlA2/ POST (canceled) xhr main.min.js:1 0 B 8.00 s
profilepic/?asid=876542902679632... GET 200 jpeg picture 1.5 kB 149 ms
guest.png GET 200 png main.min.css 891 B 21 ms
facebook.png GET 200 png main.min.css 913 B 23 ms
twitter.png GET 200 png main.min.css 950 B 20 ms
rumbletalk.png GET 200 png main.min.css 922 B 21 ms
collect?v=2&tid=G-KN3CGQEL9C... POST 204 ping js?id=G-KN3CG... 17 B 52 ms
WPhBvlA2/ POST 200 xhr main.min.js:1 503 B 377 ms
usericon.png GET 200 png main.min.js:1 3.0 kB 22 ms
collect?v=1&_v=j101&a=15313150... GET 200 gif analytics.js:22 55 B 34 ms
WPhBvlA2/ POST (canceled) xhr main.min.js:1 0 B 3.39 s
usericon.png GET 200 png main.min.js:1 3.0 kB 21 ms
collect?v=1&_v=j101&a=15313150... GET 200 gif analytics.js:22 55 B 31 ms
icon.png GET 200 png Other 5.7 kB 19 ms
WPhBvlA2/ POST 200 xhr main.min.js:1 314 B 373 ms
WPhBvlA2/ POST (pending) xhr main.min.js:1 0 B Pending
collect?v=2&tid=G-KN3CGQEL9C... POST 204 ping js?id=G-KN3CG... 17 B 51 ms

Rumbletalk

Простой интерфейс может прятать сложную логику

Избегайте сложности

Жизнь и так непроста

```
// const bob = new ChatUserObjectModel('Bob')

const bob = await getChatUserObjectModel('Bob');

const [sam, alice] = await getChatUserObjectModels(
  ['Sam', 'Alice']
);

const [rob, dave] = await getChatUserObjectModels(
  ['Rob', 'Dave'],
  { state: MUTUAL_FRIENDSHIP }
);

it.withActors("Test name", async ({actors}) => {
  // ...
});
```

Используйте фабрики

В нашем базовом примере мы использовали конструкторы, для создания User Object. Но скорее стоит подумать про фабрики

- Подготовка пользователей может быть сложным и долгим процессом
- Вам может понадобиться связка пользователей
- Фабрика может не только создать пользователей, но и привести их в нужное состояние

Генерируйте код

Писать API реализацию может быть проще

- Если совсем этого не делать
- Если создавать их путем композиции API клиентов, которые могут быть [сгенерированы из документации]

Переиспользуйте User Object

Убедить разработчиков продукта использовать User Object может быть удачной идеей. Им все равно нужно делать запросы и работать с состоянием

- UO будет протестирован
- Поддержкой будет заниматься продуктовая команда
- Возможности UO будут меняться под запросы продукта

Выводы

- Мы рассмотрели один из вариантов решения задачи, когда нам нужно контролировать внешние (по отношению к нашему приложению) сущности
- Паттерн оказался удачным (пока мы не наткнулись на очевидные минусы)
- Погружение в теорию оправдало наш подход
- Помните о силе абстракции, которая является одним из важных фундаментальных принципов, помогающих управлять сложностью программных систем и облегчать их поддержку

Спасибо за ваше время и внимание 🙂