

Лекция 12

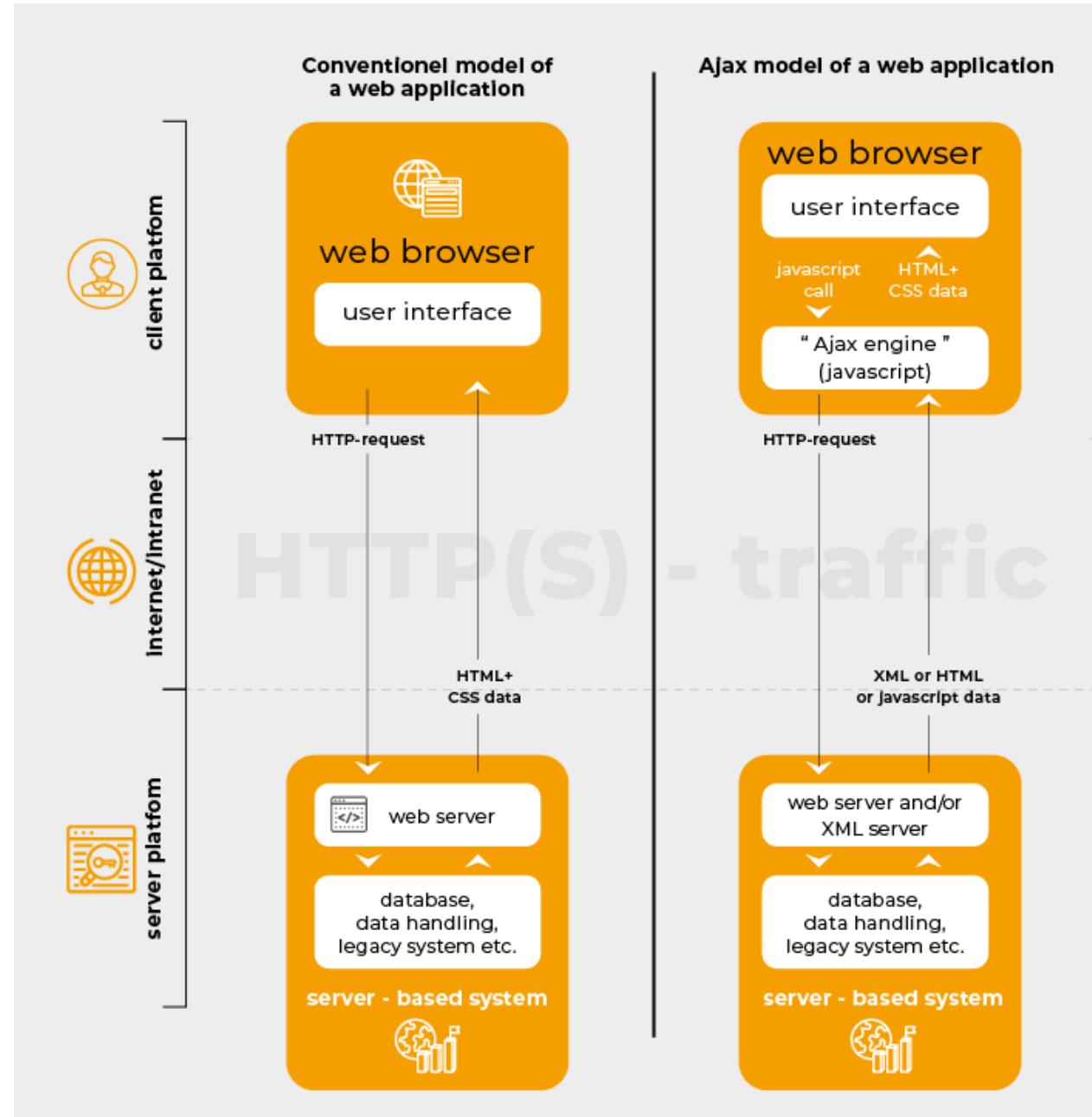
Axios. Бизнес-процесс

Проектирование систем и продуктовая веб-разработка

Канев Антон Игоревич

AJAX (повтор)

- **AJAX**, Ajax (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.
- В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.
- **JSON-RPC** (JavaScript Object Notation Remote Procedure Call — JSON-вызов удалённых процедур) — протокол удалённого вызова процедур, использующий JSON для кодирования сообщений.



Axios vs fetch

- Fetch — нативный низкоуровневый JavaScript интерфейс для выполнения HTTP-запросов с использованием обещаний через глобальный метод `fetch()`.
- Axios — JavaScript-библиотека, основанная на обещаниях, для выполнения HTTP-запросов.

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => response.json())
  .then(json => console.log(json))
```

```
axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => console.log(response));
```

The screenshot shows a code editor with two tabs: 'App.js' and 'Browser'. The 'App.js' tab contains code demonstrating both Axios and Fetch. The 'Browser' tab shows the resulting application interface, which includes a sidebar with 'Create Post', 'Update Post', and 'Delete Post' buttons, and a main area displaying Latin text: 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit'.

```
JS App.js x Browser Tests
< > ⌂ https://ociw9.csb.app/
1 import axios from "axios";
2 import React from "react";
3
4 const baseURL = "https://jsonplaceholder.typicode.com";
5
6 export default function App() {
7   const [post, setPost] = React.useState(null);
8
9   React.useEffect(() => {
10     axios.get(`${baseURL}/1`).then((response) => {
11       setPost(response.data);
12     });
13   }, []);
14
15   function createPost() {
16     axios
17       .post(baseURL, {
18         title: "Hello World!",
19         body: "This is a new post."
20       })
21       .then((response) => {
22         setPost(response.data);
23       });
24   }
25
26   return (
27     <div>
28       <h1>Hello World!</h1>
29       <p>This is a post:</p>
30       <pre>{JSON.stringify(post)}</pre>
31       <button onClick={createPost}>Create Post</button>
32       <button onClick={() => setPost(null)}>Delete Post</button>
33     </div>
34   );
35 }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
9a7572e27
```

Обработка ошибок

- Axios обрабатывает ошибки логично.
- Если сервер вернул ответ с HTTP статусом ошибки (например 404 или 500), то обещание будет отвергнуто.

```
fetch(url)
  .then(response => {
    return response.json().then(data => {
      if (response.ok) {
        return data;
      } else {
        return Promise.reject({status: response.status, data});
      }
    });
  })
  .then(result => console.log('success:', result))
  .catch(error => console.log('error:', error));
```

```
axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => console.log(response));
```

POST

- С axios всё просто, а с fetch уже не так:
- JSON обязан быть преобразован в строку, а заголовок Content-Type должен указывать, что отправляются JSON данные,
- иначе сервер будет рассматривать их как строку.

```
axios.post('/user', {  
  firstName: 'Fred',  
  lastName: 'Flintstone'  
});
```

```
fetch('/user', {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json'  
  },  
  body: JSON.stringify({  
    firstName: 'Fred',  
    lastName: 'Flintstone'  
  })  
});
```

Базовые значения для запросов

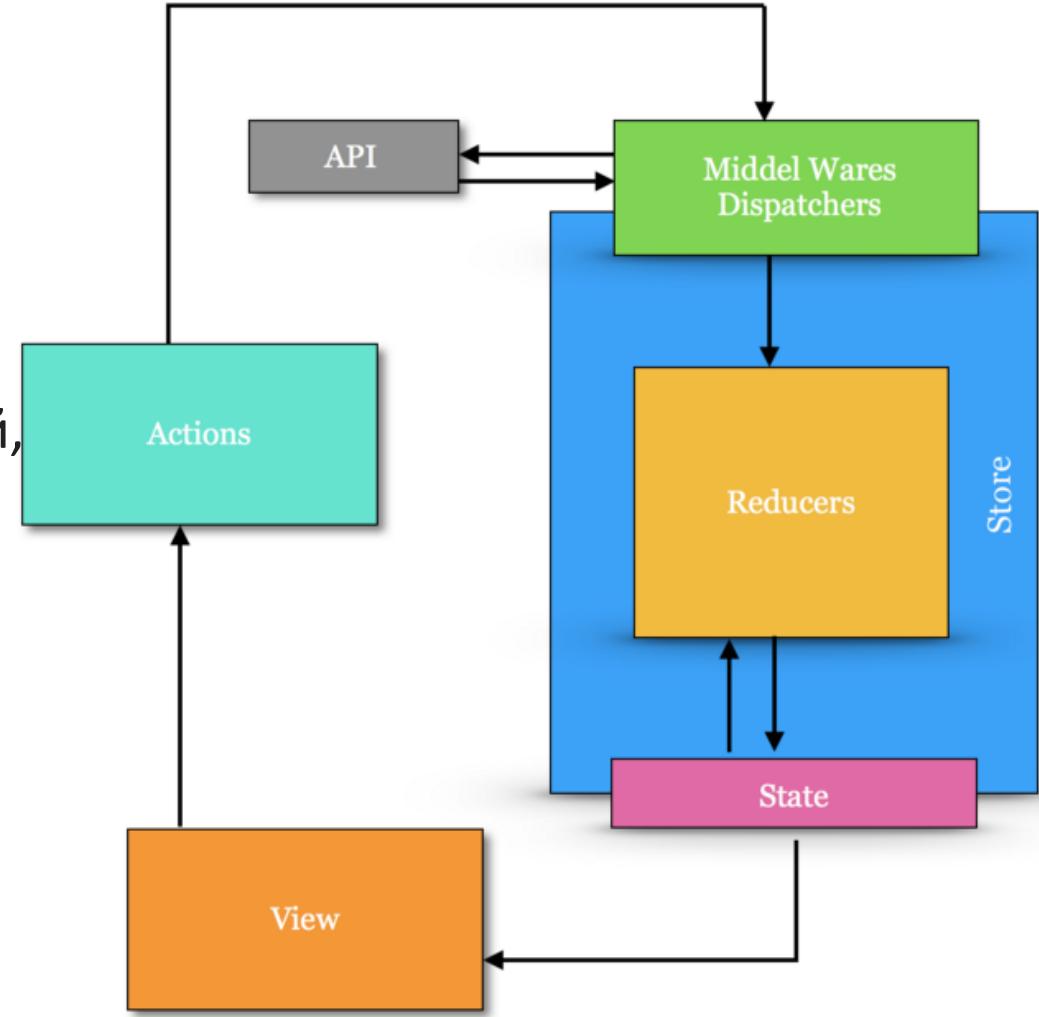
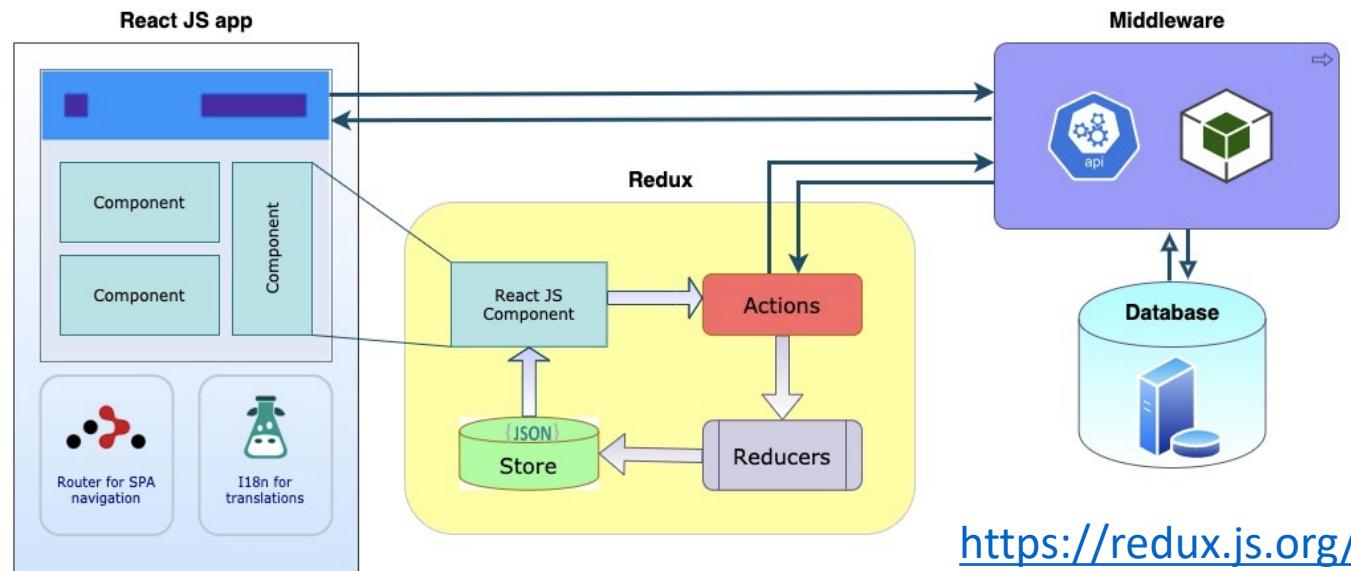
- `fetch` это явный API, вы ничего не получаете, если об этом не просите.
- Если используется аутентификация, основанная на сохранении сессии пользователя, то надо явно указывать куку.
- Если сервер расположен на поддомене, то надо явно прописывать CORS.
- Эти опции надо прописывать для всех вызовов сервера и у `fetch` нет механизма для установки значений по-умолчанию, а у `axios` есть.

```
axios.defaults.baseURL = 'https://api.example.com';
axios.defaults.headers.common['Accept'] = 'application/json';
axios.defaults.headers.post['Content-Type'] = 'application/json';
```

Redux Toolkit middleware

В наиболее общем случае, термин *middleware* часто используют для обозначения инфраструктуры: веб-серверов, серверов приложений, мониторов транзакций, программного обеспечения сервисных шин.

middleware - это код между двумя фреймворками создавшим и получившими запрос: логирование, изменение CORS заголовков и тд



- В 7ой лабораторной мы будем выполнять запрос не в самом компоненте как раньше
- Теперь мы модифицируем **наш action** который будет **сам выполнять** запрос к API

Redux-middleware

- **Мидлвары** (middlewares) — это функции, которые последовательно вызываются в процессе обновления данных в хранилище.

Мидлвары используются в задачах:

- Логирование
- Оповещение об ошибках
- Работа с асинхронным API
- Маршрутизация

```
const logger = store => next => action => {
  let result;
  console.groupCollapsed("dispatching", action.type);
  console.log("prev state", store.getState());
  console.log("action", action);
  result = next(action);
  console.log("next state", store.getState());
  console.groupEnd();
  return result;
};
```

```
const middleware = applyMiddleware(logger);
const store = createStore(reducers, middleware);
```

Redux Toolkit fetch

- Ранее мы уже рассмотрели простой вариант создания action с данными, полученными из API
- Таким образом Redux и обращение к API непосредственно друг с другом не связаны. Их объединяет обработчик события

```
initialState: {
  loading: 'idle',
  users: [],
},
reducers: {
  usersLoading(state, action) {
    // Используем подход "state machine"
    if (state.loading === 'idle') {
      state.loading = 'pending';
    }
  },
  usersReceived(state, action) {
    if (state.loading === 'pending') {
      state.loading = 'idle';
      state.users = action.payload;
    }
  },
},
```

```
// Деструктурируем и экспортируем обычных создателей
export const {
  usersLoading,
  usersReceived,
} = usersSlice.actions;

// Определяем `thunk`, отправляющего создателей
const fetchUsers = () => async (dispatch) => {
  dispatch(usersLoading());
  const response = await usersAPI.fetchAll();
  dispatch(usersReceived(response.data));
};
```

Redux Thunk middleware

- Рассмотрим пример обращения к API через thunk
- В таком примере само **обращение к API** у нас **скрыто в action**
- В коде обработчика мы просто создаем действие с нужными параметрами, а средствами redux toolkit выполняется запрос и заполняется payload

<https://github.com/reduxjs/redux-thunk>

- Обратите внимание на код справа: для **выполнения запроса** используется **сгенерированный** на основе swagger **код**, который мы импортируем из userAPI

```
import {
  createAsyncThunk,
  createSlice,
} from '@redux/toolkit';
import { userAPI } from './userAPI';

// Создаем преобразователя
const fetchUserById = createAsyncThunk(
  'user/fetchByIdStatus',
  async (userId, thunkAPI) => {
    const response = await userAPI.fetchById(userId);
    return response.data;
  }
);

// Обрабатываем операции в редукторах
const usersSlice = createSlice({
  name: 'users',
  initialState: { entries: [], loading: 'idle' },
  reducers: {
    // стандартная логика редуктора с авто-генерируемыми
  },
  extraReducers: {
    [fetchUserById.fulfilled]: (state, action) => {
      // Добавляем пользователя в массив состояния
      state.entries.push(action.payload);
    },
  },
});

// Позже, отправляем `thunk`
dispatch(fetchUserById(123));
```

Swagger (повтор)

- Swagger – это фреймворк для спецификации RESTful API.
- Swagger UI позволяет интерактивно просматривать спецификацию и отправлять запросы
- Полученное на бэке описание можно использовать для генерации кода фронтенда

The screenshot shows a web browser displaying the Swagger UI for a 'JWT Auth API'. The URL in the address bar is `https://localhost:5001/index.html`. The main title is 'JWT Auth API' with version 'v1' and 'OAS3' badges. Below the title, there are links to '/swagger/v1/swagger.json', 'A simple example JWT Auth API', and 'GitHub Repository - Website'. A green 'Authorize' button with a lock icon is visible on the right. The interface is divided into sections: 'Account' (containing a POST method for '/Account/login' with 'JWT login' notes) and 'Values' (containing three GET methods: '/api/Values' (notes: 'API allows anonymous'), '/api/Values/jwt' (notes: 'API requires JWT auth'), and '/api/Values/basic' (notes: 'API requires Basic auth')). Each method row has a lock icon on the right.

Кодогенерация для фронта из swagger

- У нас есть файл от Django по ссылке <http://127.0.0.1:8000/swagger/?format=openapi>
- Для кодогенерации скачаем библиотеку swagger-typescript-api
 - npm i --save-dev swagger-typescript-api
- Добавим в package.json команду

```
"scripts": {  
    "generate-backend-types": "swagger-typescript-api -p http://127.0.0.1:8000/swagger/?format=openapi --no-client -o ./types/autogenerated -n backend.ts" }  
• Выполним
  - npm run generate-backend-types
```
- Получим в папке types/autogenerated файл backend.ts внутри которого будут сгенерированы интерфейсы повторяющие сущности из бэкенда
- Можно сгенерировать сразу методы обращения к API. Нужно их использовать в коде TS

Пример кодогенерации

- В примере FSD при генерации мы получаем методы для выполнения наших запросов к API
- Далее в коде наших обработчиков событий или thunk мы используем эти методы

```
import { createAsyncThunk } from '@reduxjs/toolkit';

import { api } from '@api';

import type { LoginData, LoginBody } from './fetchLogin.types';

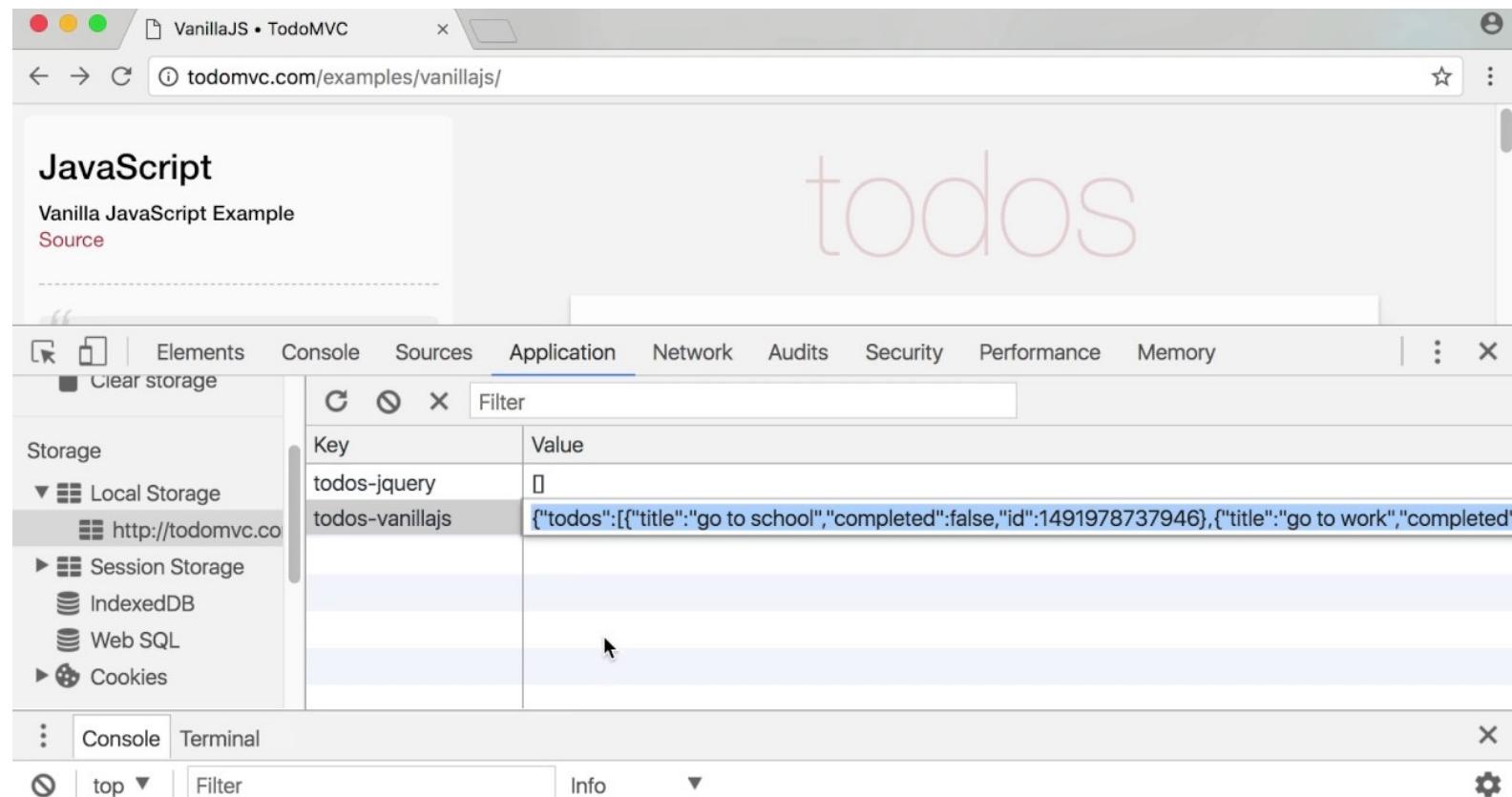
export const fetchLogin = createAsyncThunk<LoginData, LoginBody>('login/fetchLogin', async (body) => {
  const { data } = await api.authenticate.authenticateCreate(body);

  return data.role;
});
```

```
export class Api<SecurityDataType extends unknown> extends HttpClient<SecurityDataType> {
  authenticate = {
    /**
     * @description Login
     *
     * @tags authenticate
     * @name AuthenticateCreate
     * @request POST:/authenticate/
     * @secure
     */
    authenticateCreate: (data: Login, params: RequestParams = {}) =>
      this.request<UserSwagger, any>({
        path: `/authenticate/`,
        method: 'POST',
        body: data,
        secure: true,
        type: ContentType.Json,
        format: 'json'.
      })
  }
}
```

Вкладка Application. Local Storage

- Что делать с данными приложения если нажали F5?
- В варианте с JWT мы используем Local Storage (локальное хранилище) для хранения JWT
- **Не используем Local Storage** в нашем курсе кроме как для хранения JWT, поэтому по F5 либо пользователь пропадает, либо нужен запрос



The screenshot shows the Chrome DevTools interface with the Application tab selected. In the left sidebar, under 'Storage', 'Local Storage' is expanded, showing items for 'http://todomvc.com'. One item, 'todos-vanillajs', is selected and its value is displayed in the main pane: a JSON array of todos. Below the storage list, there's a code editor with three snippets of JavaScript related to local storage operations.

Key	Value
todos-jquery	[]
todos-vanillajs	{"todos": [{"title": "go to school", "completed": false, "id": 1491978737946}, {"title": "go to work", "completed": false, "id": 1491978737947}]} [{"title": "go to school", "completed": false, "id": 1491978737946}, {"title": "go to work", "completed": false, "id": 1491978737947}]

```
const storedSession = localStorage.getItem("session");
localStorage.setItem("session", JSON.stringify(state));
localStorage.removeItem("session");
```

Заявка-черновик

- Страница списка услуг включает сам список услуг и кнопку перехода в «корзину» - на страницу с заявкой-черновиком

Новый год



	Традиционное	Тонкое	
23 см	30	35	40

Мясное барбекю (1) i
Мясная пицца с пикантной
пепперони, альпийскими
колбасками, Моцареллой, луком и
соусом Барбекю

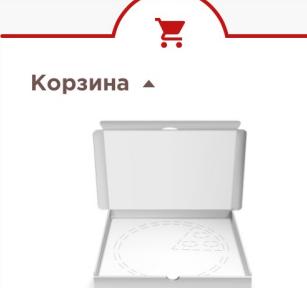
Пепперони (1) i
Американская классика с
пикантной пепперони, Моцареллой
и фирменным томатным соусом

Цыпленок Барбекю (1) i
Сочное куриное филе и ароматный
бекон в сочетании с фирменным
томатным соусом, Моцареллой,
луком и соусом Барбекю

+ чесночно-сырная корочка 99 ₽
В корзину 549 ₽

+ чесночно-сырная корочка 99 ₽
В корзину 469 ₽ 469 ₽

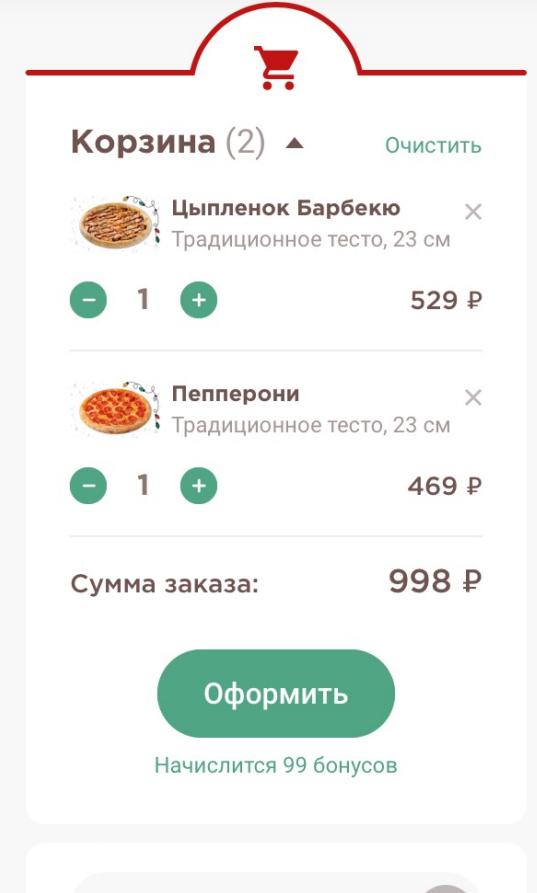
+ чесночно-сырная корочка 99 ₽
В корзину 529 ₽



Корзина ▲

Корзина пуста. Выберите пиццу из меню или повторите предыдущий заказ

Ввести промокод >



Корзина (2) ▲ Очистить

 Цыпленок Барбекю Традиционное тесто, 23 см	- 1 +	529 ₽
 Пепперони Традиционное тесто, 23 см	- 1 +	469 ₽

Сумма заказа: 998 ₽

Оформить

Начислятся 99 бонусов

Ввести промокод >

- При нажатии на кнопку «Корзины» мы открываем страницу конструктора заявки
- Там **5 кнопок**: Сохранить, Сформировать, Удалить, удал./ред. м-м

Диаграмма деятельности

- Также в 7 лабораторной нужно описать бизнес процесс для «ура-сценария» в вашей системе
- Его описываете на диаграмме деятельности или в BPMN 2.0
- У вас будет **3 дорожки**: создатель, модератор, выделенный сервис (например оплата). Дорожки называйте по вашей теме

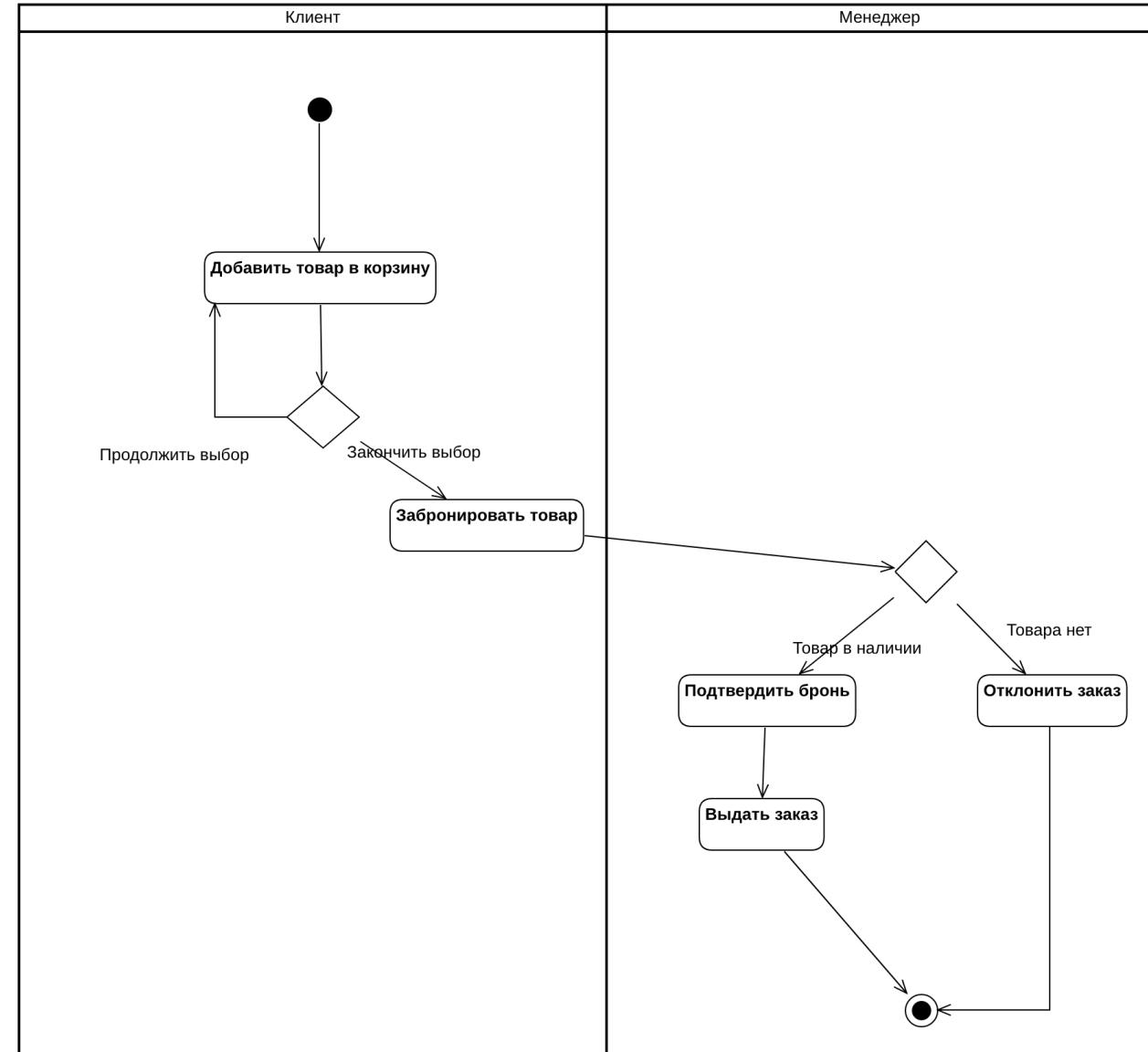


Диаграмма состояний

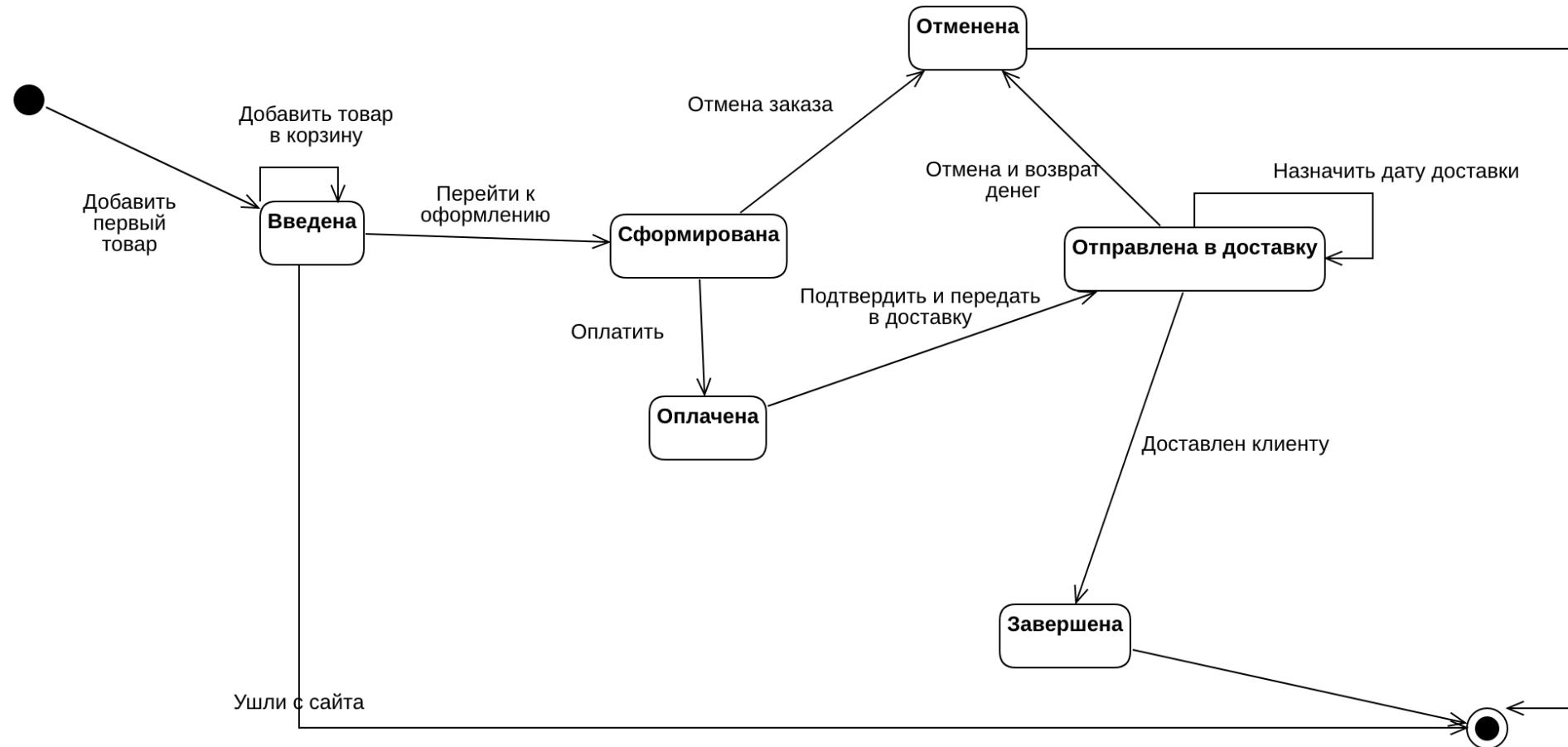
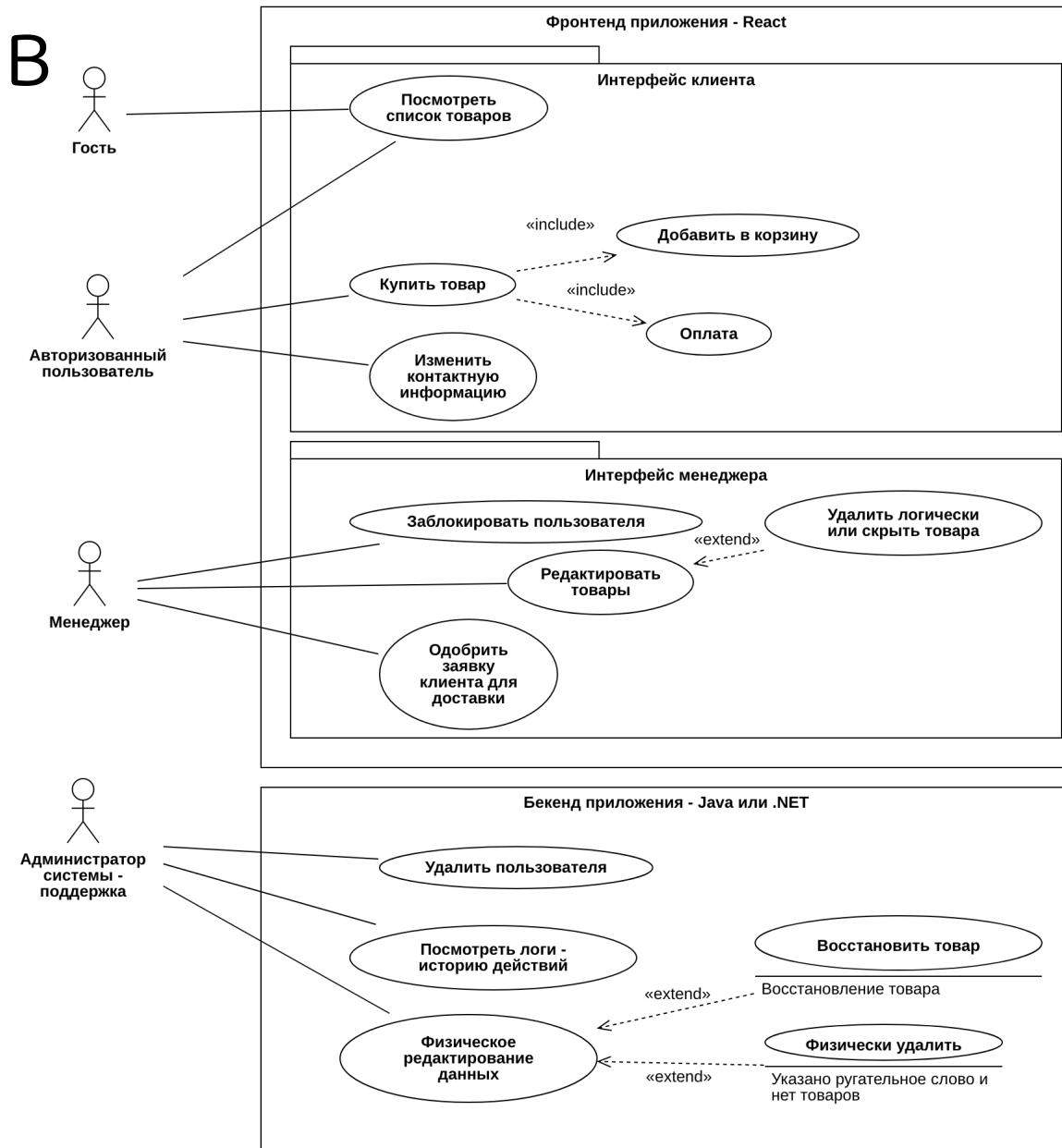


Диаграмма прецедентов

- В отчет вы также добавляете диаграмму прецедентов и диаграмму состояний
- На диаграмме прецедентов указываете роли пользователей вашей системы и действия, которые они могут в ней выполнить, как в функциональных требованиях
- Также требуется обновить и исправить все старые диаграммы и включить их в РПЗ
- **Менеджер для доп. задания**



Два списка услуг

- Ваша страница списка услуг для гостя и создателя заявки представлена карточками
- Тут вы добавляете виджет корзины и кнопку добавления услуги в заявку

The screenshot shows the Google Docs interface. At the top, there's a navigation bar with 'Docs' and a search bar. Below it, a button says 'Start a new document'. On the right, there's a 'Template gallery' section with several resume templates like 'Blank', 'Essay Playful', 'Resume Spearmint', 'Resume Serif', and 'Resume Swiss'. At the bottom, there's a 'STORAGE' section showing '0 GB used'.

The screenshot shows the Google Drive interface. It displays a list of files in 'My Drive'. A red circle with the number '1' highlights the 'New' button in the sidebar. Another red circle with the number '2' highlights the edit icon (pencil) next to a file named 'Travel the World.docx'.

Name	Last modified	Owner
Trade Show Docs	Nov 15, 2019 m	me
Tracking	Nov 15, 2019 m	me
Pictures	Sep 5, 2019 me	me
Meeting Documents	Sep 5, 2019 me	me
Travel the World.docx	Nov 22, 2019 m	me
Trade Show Materials.zip	Sep 5, 2019 me	me
Technology Proposal.docx	Nov 7, 2019 me	me
Survey Results.xlsx	Nov 22, 2019 m	me
nest_logo.jpg	Sep 5, 2019 me	me
Event Letter	Nov 22, 2019	me

- Но список услуг в **доп. задании** для редактирования модератором должен быть в виде таблицы
- Тут будут кнопки добавления, редактирования и удаления