**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ**

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент | | Тимшин Дмитрий Алексеевич | | | | | | | | | | | |
| ( Фамилия, И., О. ) | | | | | | | | | | | | | |
| Факультет | | | ИТИП | | | | | | | | | | |
| Кафедра | ИС | | | | | | | | | | Группа | | М3406 |
| Направление (специальность) | | | | | | | | 09.03.02 | | | | | |
| Руководитель | | | | Посевкин Р.В., доцент, кандидат технических наук | | | | | | | | | |
| ( Фамилия, И.О., должность, ученое звание, степень ) | | | | | | | | | | | | | |
| Дисциплина | | | | | | | Web-программирование | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Наименование темы | | | | | | | Разработка клиент-серверного веб-приложения | | | | | | |
|  | | | | | | |  | | | | | | |
| Задание | | | | | | Разработать клиент-серверное веб-приложение, позволяющее получать данные о погоде с помощью внешнего API OpenWeather. | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Краткие методические указания | | | | | | | | |  | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Содержание пояснительной записки | | | | | | | | | |  | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Рекомендуемая литература | | | | | | | | | | | |  | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Руководитель | | | | |  | | | | | | | | |
| Подпись, дата | | | | | | | | | | | | | |
| Студент | | | |  | | | | | | | | | |
| Подпись, дата | | | | | | | | | | | | | |

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**УНИВЕРСИТЕТ ИТМО**

**Факультет информационных технологий и программирования**

**КУРСОВАЯ РАБОТА**

Тема: Разработка клиент-серверного веб-приложения

Работу выполнил студент: Тимшин Дмитрий Алексеевич группы М3406

(фамилия, имя, отчество) (номер группы)

Руководитель Посевкин Руслан Владимирович

(фамилия, имя, отчество)

Работа защищена " " 201 г. с оценкой

Подписи членов комиссии:

САНКТ-ПЕТЕРБУРГ

2019

**Оглавление**

[**Результаты выполнения первого этапа** 4](#_heading=h.gjdgxs)

[**Задание** 4](#_heading=h.30j0zll)

[**Инструкция по работе с приложением** 4](#_heading=h.1fob9te)

[**Теоретическая информация** 4](#_heading=h.3znysh7)

[**Демонстрация работы приложения** 6](#_heading=h.2et92p0)

[**Результаты выполнения второго этапа** 9](#_heading=h.tyjcwt)

[**Задание** 9](#_heading=h.3dy6vkm)

[**Инструкция по работе с приложением** 9](#_heading=h.1t3h5sf)

[**Теоретическая информация** 10](#_heading=h.4d34og8)

[**Демонстрация работы приложения** 13](#_heading=h.2s8eyo1)

[**Результаты выполнения третьего этапа** 17](#_heading=h.17dp8vu)

[**Задание** 17](#_heading=h.3rdcrjn)

[**Теоретическая информация** 17](#_heading=h.26in1rg)

[**Демонстрация тестирования первой части** 18](#_heading=h.lnxbz9)

[**Демонстрация тестирования второй части** 19](#_heading=h.35nkun2)

[**Результаты выполнения четвертого этапа** 21](#_heading=h.44sinio)

[**Задание** 21](#_heading=h.2jxsxqh)

[**Теоретическая информация** 21](#_heading=h.z337ya)

[**Демонстрация работы приложения** 22](#_heading=h.3j2qqm3)

# **Результаты выполнения первого этапа**

## **Задание**

Сверстать страницу, содержащую поле ввода и кнопку для поиска информации о погоде в заданном городе. По нажатию на кнопку происходит вызов внешнего API погоды (например, https://openweathermap.org/api). Получаемые данные (не менее 5 элементов) отрисовываются на этой же странице.

Для отрисовки полученных данных должен использоваться один из клиентских шаблонизаторов (например, https://proglib.io/p/templating-languages-and-engines/). Для визуального оформления страницы используется CSS препроцессор (SASS, LESS, Stylus).

Работа делается в публичном github-репозитории. Все необходимые инструкции для локального запуска проекта должны быть описаны в README.md в корне проекта.

Ссылка на репозиторий:

## **Инструкция по работе с приложением**

1. Для установки зависимостей необходимо выполнить:

npm i

1. Для сборки проекта необходимо выполнить:

npm run build

1. Открыть файл index.html

Для актуализации стилей необходимо пересобрать проект выполнив:

npm run build

Или сгенерировать только стили выполнив:

sass --watch assets/styles.scss assets/styles.css

## **Теоретическая информация**

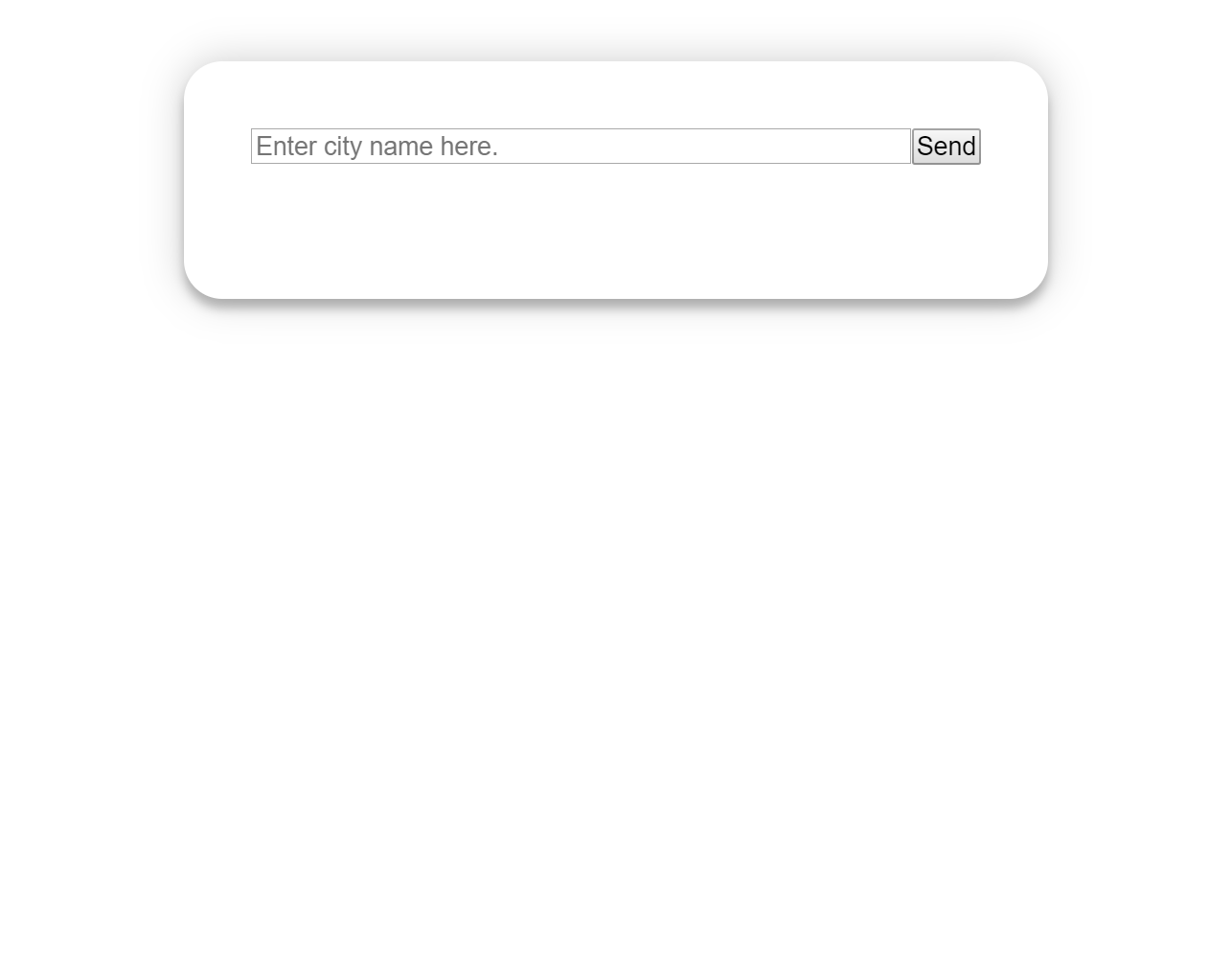
CSS препроцессор (CSS preprocessor) — это программа, которая имеет свой собственный синтаксис (syntax), но может сгенерировать из него CSS код. Существует множество препроцессоров. Большинство из них расширяет функционал чистого CSS, добавляя такие опции как: примеси, вложенные правила, селекторы наследования и др. Эти особенности облегчают работу с CSS: упрощают чтение кода и его дальнейшую поддержку.

В качестве препроцессора используется Sass. Sass это расширение CSS, которое придаёт мощи и элегантности этому простому языку. Sass даст вам возможность использовать переменные, вложенные правила, миксины, инлайновые импорты и многое другое, всё с полностью совместимым с CSS синтаксисом. Sass помогает сохранять огромные таблицы стилей хорошо организованными, а небольшим стилям работать быстро.

Шаблонизатор — программное обеспечение, позволяющее использовать html-шаблоны для генерации конечных html-страниц. Основная цель использования шаблонизаторов — это отделение представления данных от исполняемого кода. Часто это необходимо для обеспечения возможности параллельной работы программиста и дизайнера-верстальщика. Использование шаблонизаторов улучшает читаемость кода и внесение изменений во внешний вид. При работе над приложением используется Marko.

Marko – это легковесный и быстрый html-шаблонизатор от eBay. Он работает на Node.js и поддерживает потоковую передачу данных, асинхронный рендеринг и пользовательские теги.

## **Демонстрация работы приложения**

****

*Рисунок 1. Стандартный интерфейс приложения.*

****

*Рисунок 2. Погода в Токио.*

****

*Рисунок 3. Демонстрация ошибки при попытке получить погоду.*

# **Результаты выполнения второго этапа**

## **Задание**

Создать приложение с использованием библиотеки React.

При первой загрузке страницы происходит запрос пользователя на получение данных о геолокации с использованием HTML5 Geolocation API. Если пользователь соглашается предоставить данные о геолокации – получаем из внешнего API данные о погоде. Если нет – запрашиваем информацию для города по умолчанию (город по умолчанию можно выбрать самостоятельно). Информация о городе, данные о погоде (температура, ветер, давление, влажность), иконка погоды, координаты отрисовываются на странице в соответствии с макетом.

Иконка и все необходимые данные есть в API https://openweathermap.org.

В интерфейсе также есть кнопка с повторным запросом геолокации пользователя.

У пользователя есть возможность добавления и удаления городов в избранное. Информация о погоде отображается для всех городов из избранного в соответствии с макетом. Избранное сохраняется в LocalStorage.

Пока происходит загрузка данных по конкретному городу/локации – показываем loader и/или сообщение об ожидании загрузки данных.

Работа с глобальным состоянием приложения (например, список избранных городов) реализуется с помощью Redux.

Локальное состояние компонента (например, состояние ожидания загрузки данных) – через локальный state компонента.

Ссылка на репозиторий:

## **Инструкция по работе с приложением**

1. Для установки зависимостей необходимо выполнить:

npm i

1. Для запуска приложения необходимо выполнить:

npm run start

## **Теоретическая информация**

React (иногда React.js или ReactJS) — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как Redux.

React исходит из принципа, что логика рендеринга неразрывно связана с прочей логикой UI: с тем, как обрабатываются события, как состояние изменяется во времени и как данные готовятся к отображению. Вместо того, чтобы искусственно разделить технологии, помещая разметку и логику в разные файлы, React разделяет ответственность с помощью слабо связанных единиц, называемых «компоненты», которые содержат и разметку, и логику.

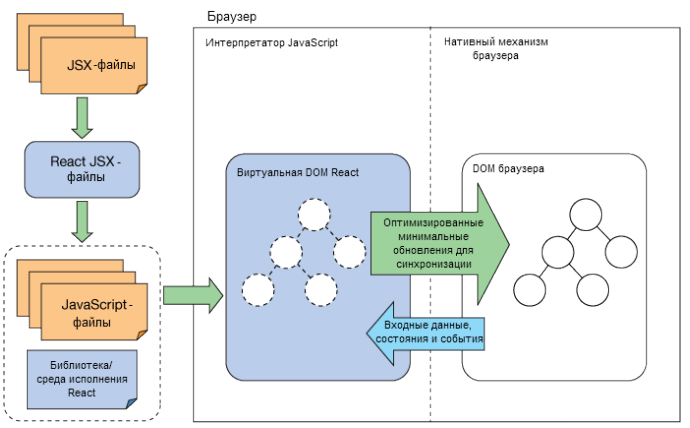
React можно использовать и без JSX, но большинство людей ценит его за наглядность при работе с UI, живущем в JavaScript-коде. Помимо этого, JSX помогает React делать сообщения об ошибках и предупреждениях понятнее.

Компоненты позволяют разделить UI на независимые, повторно используемые части и работать с каждой из них отдельно.

Концептуально, компоненты похожи на JavaScript-функции. Они принимают произвольные данные (называемые props) и возвращают React-элементы, которые описывают то, что должно появиться на экране.

Виртуальный DOM (VDOM) — это концепция программирования, в которой идеальное или «виртуальное» представление пользовательского интерфейса хранится в памяти и синхронизируется с «настоящим» DOM при помощи библиотеки, такой как ReactDOM. Этот процесс называется согласованием.

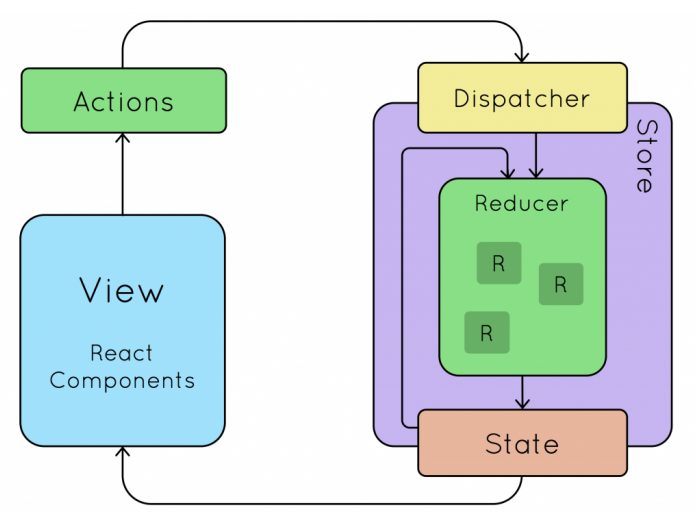
Такой подход и делает API React декларативным: вы указываете, в каком состоянии должен находиться пользовательский интерфейс, а React добивается, чтобы DOM соответствовал этому состоянию. Это абстрагирует манипуляции с атрибутами, обработку событий и ручное обновление DOM, которые в противном случае пришлось бы использовать при разработке приложения.

****

*Рисунок 4. Общая схема работы React.*

Redux — это инструмент управления как состоянием данных, так и состоянием интерфейса в JavaScript-приложениях. Он подходит для одностраничных приложений, в которых управление состоянием может со временем становиться сложным.

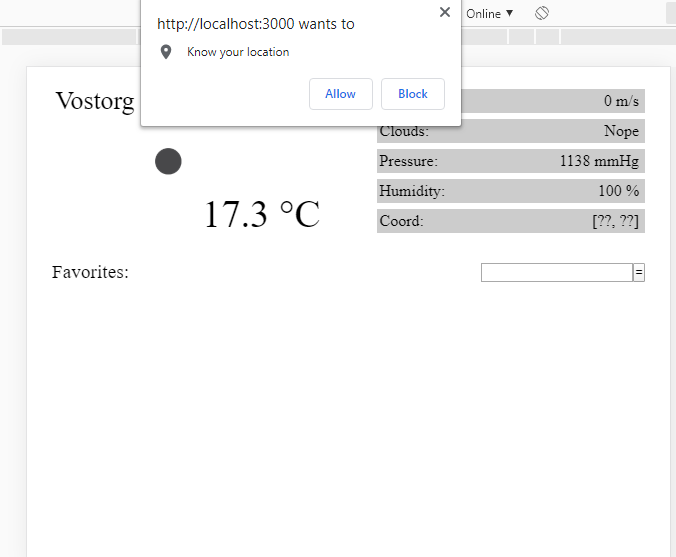
Redux предлагает хранить все состояние приложения в одном месте, называемом «store» («хранилище»). Компоненты «отправляют» изменение состояния в хранилище, а не напрямую другим компонентам. Компоненты, которые должны быть в курсе этих изменений, «подписываются» на хранилище. Хранилище может рассматриваться как «посредник» во всех изменениях состояния в приложении. С Redux компоненты не связываются друг с другом напрямую, все изменения должны пройти через единственный источник истины, через хранилище.

****

*Рисунок 5. Схема работы Redux.*

LocalStorage представляет собой постоянное хранилище данных, расположенное у пользователя на локальном диске. Таким образом, в нём можно сохранять данные, которые будут доступны веб-приложению даже если пользователь перейдёт на другую страницу сайта, либо даже вовсе покинет страницу и зайдёт через некоторое время.

## **Демонстрация работы приложения**

****

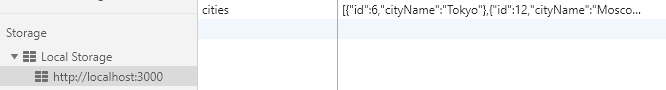
*Рисунок 6. Запрос прав на определение геолокации.*

****

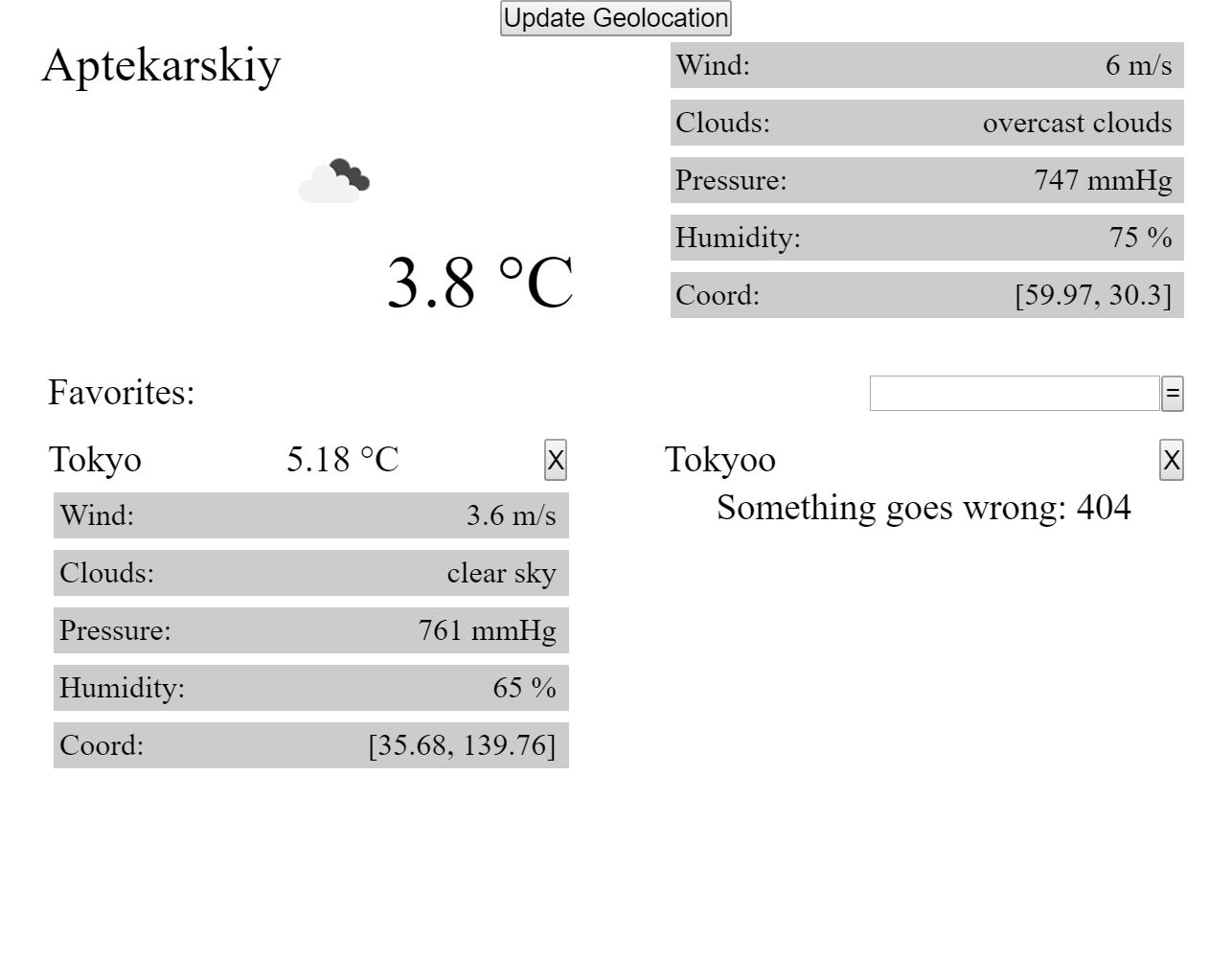
*Рисунок 7. Отображение погоды в районе геолокации пользователя.*

****

*Рисунок 8. Отображение погоды и загрузки в городах из избранного.*

****

*Рисунок 9. Данные об избранном хранятся в LocalStorage.*

****

*Рисунок 10. Отображение ошибки при попытке получить погоду.*

# **Результаты выполнения третьего этапа**

## **Задание**

Покрыть модульными тестами функциональность приложения, разработанного в рамках лабораторной работы №1. Пригодятся такие библиотеки как Mocha (https://mochajs.org), Chai (https://www.chaijs.com) и Sinon (https://sinonjs.org/). Запуск тестов осуществляется с помощью команды npm run test.

Покрыть снэпшот-тестами все состояния компонентов из лабораторной работы №2. Для тестирования использовать библиотеку Jest.

Ссылки на репозитории:

Первая часть -

Вторая часть -

## **Теоретическая информация**

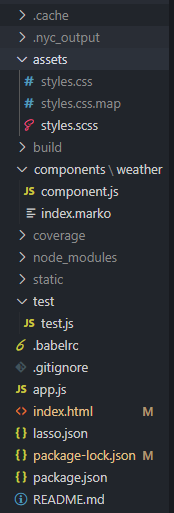
Mocha (Мока) — многофункциональная среда JavaScript-тестирования, работающая на Node.js и в браузере, что делает асинхронное тестирование простым и интересным. Тесты Mocha запускаются серийно, что позволяет гибко и точно создавать отчеты.

ChaiJS — это библиотека для node и, как Mocha, Chai может использоваться на стороне сервера или в браузере. Используется для сравнения, ожидаемого и полученного результатов. Chai может быть использован совместно с любой библиотекой для тестирования.

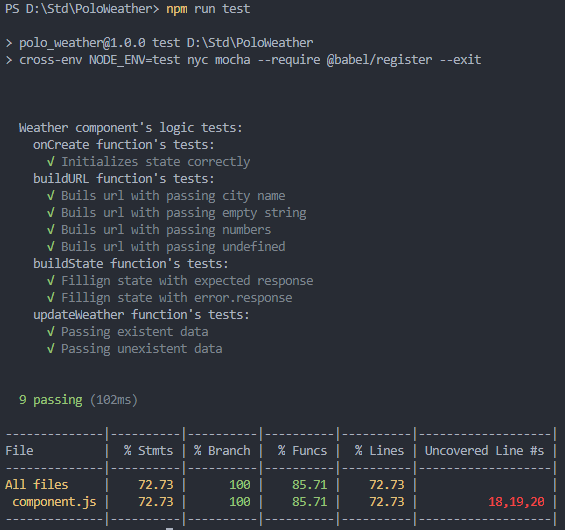
IstanbulJS – библиотека для отслеживания покрытия проекта тестами.

Jest — это фреймворк для тестирования JavaScript, разработанный для обеспечения уверенности в правильной работе любого JavaScript кода. Он позволяет вам писать тесты с приемлемым, знакомым и функциональным API, и быстро достигать желаемых результатов. Использует систему снэпшотов, позволяющую отслеживать все изменения в компонентах.

## **Демонстрация тестирования первой части**

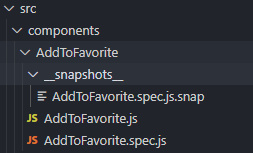
****

*Рисунок 11. Структура файловой системы.*

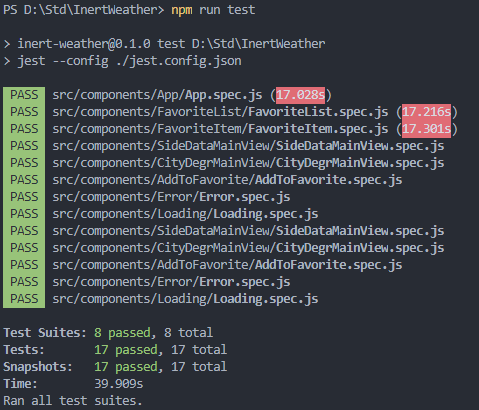
****

*Рисунок 12. Результат выполнения тестов.*

## **Демонстрация тестирования второй части**

****

*Рисунок 13. Структура файловой системы. Расположение теста и снэпшота относительно компонента.*

****

*Рисунок 14. Результат выполнения тестов.*

# **Результаты выполнения четвертого этапа**

## **Задание**

Необходимо доработать лабораторную работу №2, добавив реализацию серверной части приложения. Серверная часть реализуется на NodeJS, допустимо использовать фреймворки вроде Express или Sails.

Приложение становится клиент-серверным, запросы данных о погоде к внешнему API и хранение данных об избранных городах переносятся на сервер.

Для получения данных о погоде из внешнего API по городу используется запрос на GET-endpoint /weather (например: /weather?city=Moscow), по координатам – /weather/coordinates (например: /weather/coordinates?lat=123&long=456)

Данные об избранных городах хранятся в базе данных, можно использовать любое SQL/NoSQL решение. Для работы с избранными городами на сервере должен быть реализован endpoint /favourites, обрабатывающий POST-запросы на добавление города и DELETE-запросы на удаление конкретного города из списка. GET-запрос на /favourites возвращает список избранных городов.

Клиентская логика должна быть адаптирована с учетом этих изменений, включая обработку возможных ошибок.

Ссылка на репозиторий -

## **Теоретическая информация**

Express — это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений.

MongoDB — документно-ориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

Эндпоинты:

GET: /favorites – возвращает список городов в избранном.

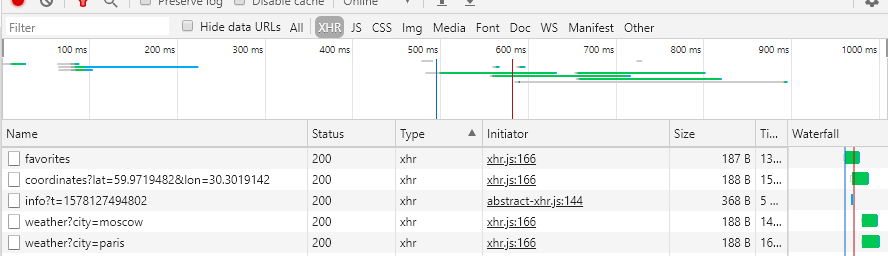
POST: /favorites – добавляет город в избранное.

DELETE: /favorites – удаляет город из избранного.

GET: /weather – возвращает данные о погоде в запрошенном городе.

GET: /weather/coordinates – возвращает данные о погоде по координатам.

## **Демонстрация работы приложения**

****

*Рисунок 15. Взаимодействие с серверной частью.*