Практическая работа №8. Ролевые модели.

Задание: создать в собственном React-приложении разделение пользователей по ролям. Настроить разные права для каждой группы пользователей.

Примеры ролей:

- Админ - может добавлять, редактировать и удалять существующие сущности всех пользователей.
- Пользователь - может добавлять, редактировать и удалять только свои сущности.

Стек технологий: React, NodeJS, PostgreSQL или MongoDB.

1. Ролевая модель. Что это?

Когда программа работает и имеет в себе определённый функционал, который не должен быть доступен обычному пользователю, возникает вопрос, а как это сделать? Для того, чтобы можно было распределять доступ к функционалу приложения, была придумана ролевая модель.

Самый простой способ ограничения — это когда одному пользователю доступен весь функционал приложения. В таком случае нужна одна таблица пользователей, у которых есть доступ к ресурсу. Соответственно и проверка будет одна.

В других же случаях, когда, к примеру, продавец отвечает только за подтверждение покупки и продажу товара, консультант за чат с клиентами, а у аналитика есть доступ только к данным по проданным товарам, нужно будет разграничивать функционал к отдельным частям и функциям программы. Конечно же и проверок будет больше, к примеру, проверка по должности, по айдишнику пользователя и т. д.

Рассмотрим самые популярные модели RBAC и ABAC.

Role-based access control (RBAC)

Идея подхода заключается в том, чтобы создать роли, которые в действительности будут повторять бизнес-роли в компании. Роли, нужно будет присваивать пользователям. С помощью данных ролей, можно будет

разграничить доступ к функциям сайта и проверить возможность выполнения того или иного действия пользователем.

В моменты, когда одна роль будет соответствовать одной функции, становится возможным просто присвоить каждой одной роли свою одну функцию. К примеру, Бухгалтер, который пользуется только эксель таблицей на ресурсе.

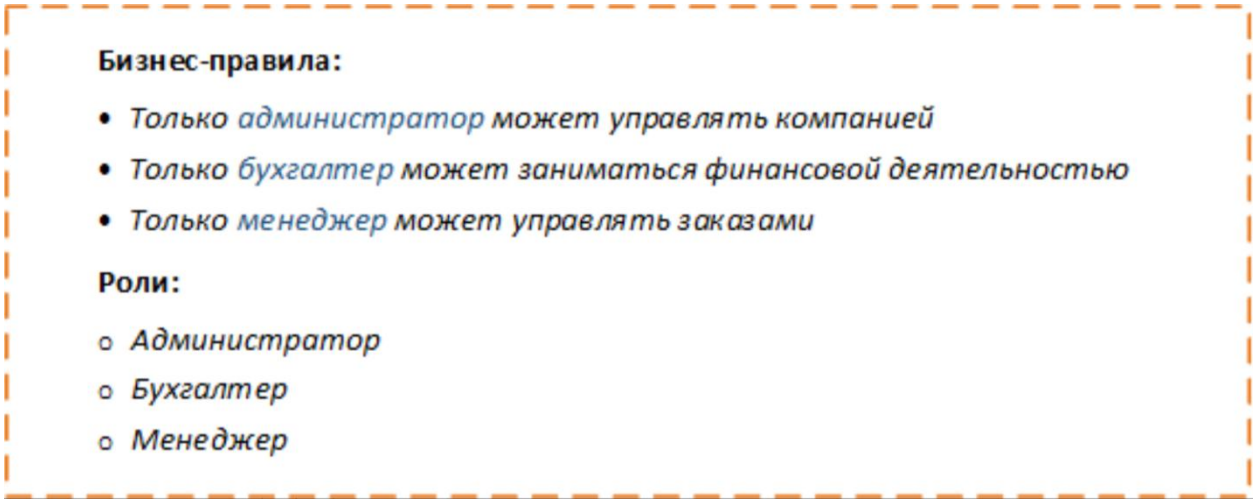

Рисунок 1. Присвоение одного бизнес-правила только для одной роли.

Рисунок 2. Примерная модель приложения, на основе простого распределения ролей.

В моменты, когда одна роль будет соответствовать одной функции, становится возможным просто присвоить каждой одной роли свою одну функцию. К примеру, Бухгалтер, который пользуется только эксель таблицей на ресурсе.

Но такая модель неустойчива, когда бизнес растёт. Приходится создавать новые роли, которые описывают буквально каждые важные различия для пользователей. К примеру, если у нас появляются новые филиалы, в которых так же развита ИТ структура. Придётся создавать для каждого филиала свою новую роль.

В моменты, когда ещё неизвестны точные данные, роль вообще создать будет невозможно.

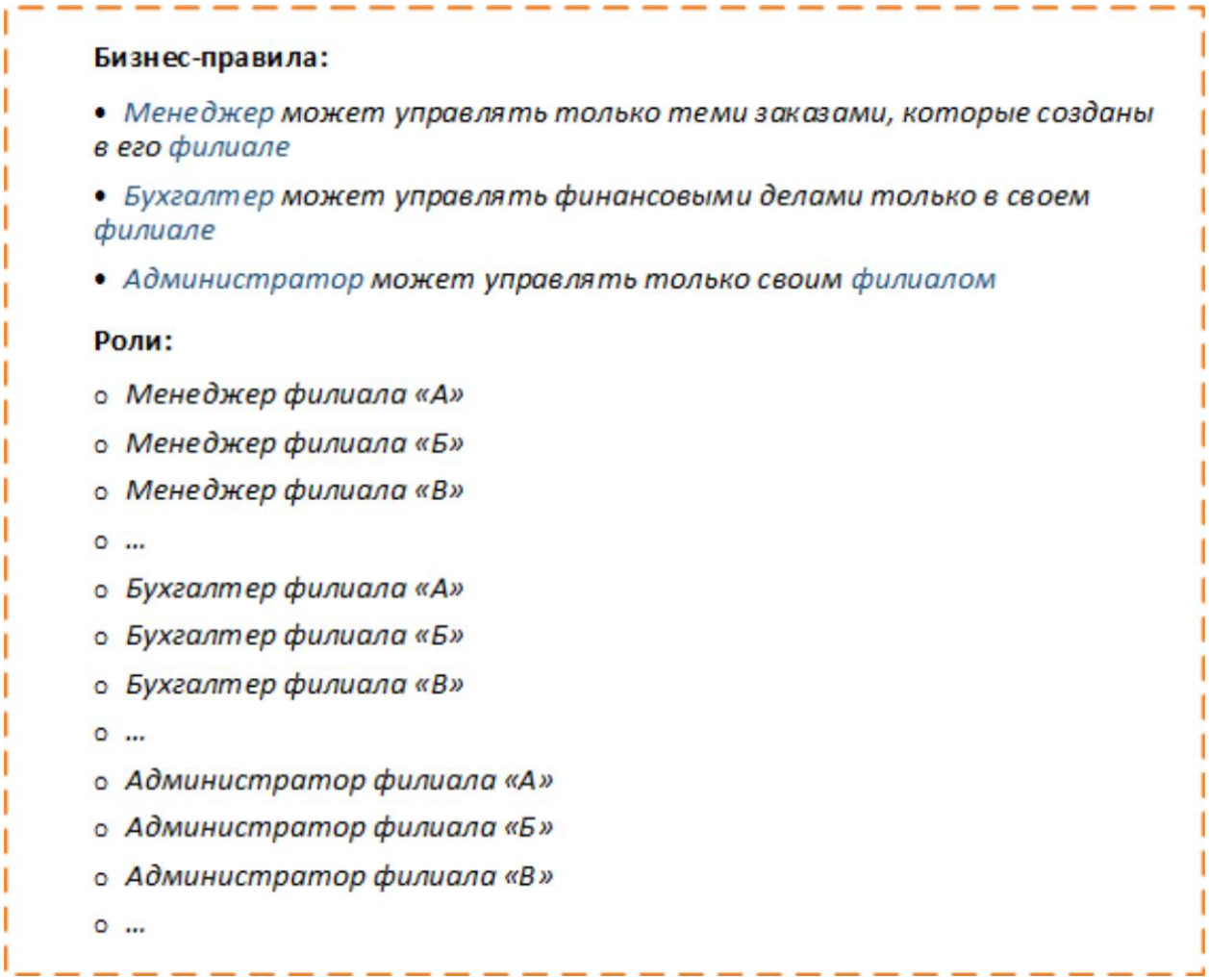

Рисунок 3. Присвоение одного бизнес-правила множеству ролей.

Стоит заметить, что в таком ролевом подходе также не существуют бизнес-правила, которые ограничивают доступ к данным.

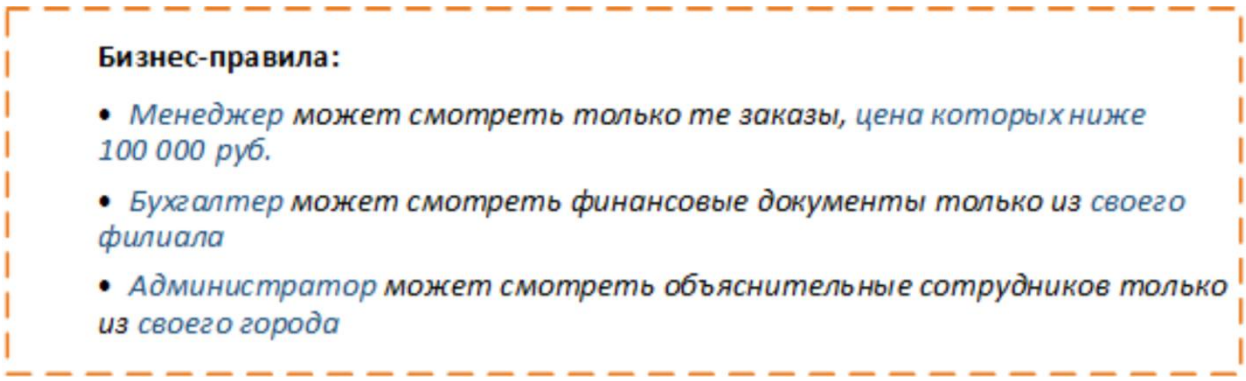

Рисунок 4. Бизнес-правила для ограничения доступа к данным.

Ролевая модель подходит для простых и однозначных бизнес-правил, когда же приходится составлять многозначные и сложные бизнес-правила, в которые может входить ограничение к доступу к данным и без прямого присвоения функционала (бухгалтерам таблички, а аналитикам доступ к статистике сайта), то ролевая модель не подходит, на её поддержание уходит много сил и ресурсов.

Attribute-based access control (ABAC)

Этот подход является заменой RBAC (вышеописанному подходу), когда система становится сложнее. Данный подход отличается тем, что он обращает внимание не на роли пользователя, а на его атрибуты.

Бизнес-правила в свою очередь, в таком подходе, выставляют свои условия с атрибутами, которые должны быть у того, кто это правило собирался выполнить. К примеру, чтобы открыть базу данных филиала Г, у пользователя должен быть статус работника склада и место работы должно быть указано в филиале Г.

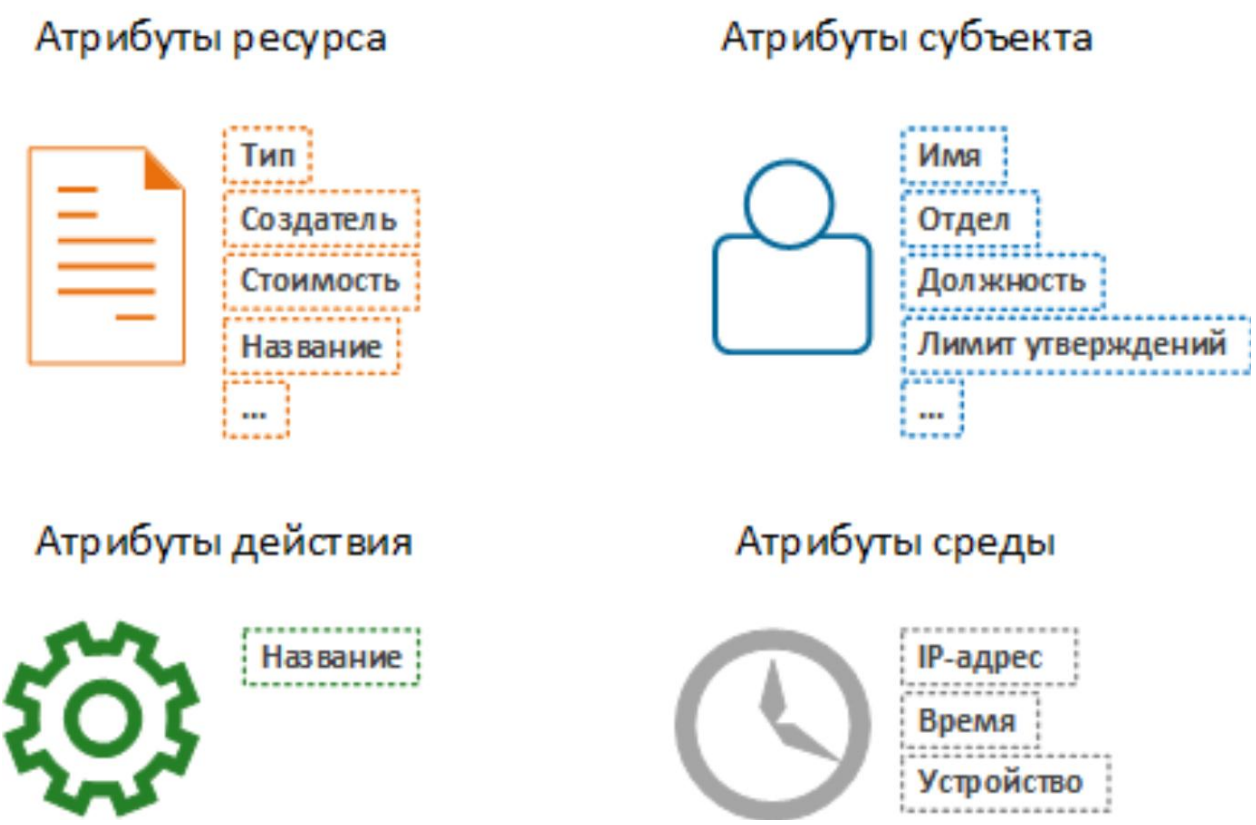

Рисунок 5. Некоторые категории атрибутов.

В момент выполнения авторизации берутся значения всех атрибутов и сравниваются с ожидаемыми значениями. На основе результатов сравнения выдаются права. Если все условия выполнены, то результат положительный и, следовательно, пользователь имеет доступ к данной функции (бизнес-плану).

Бизнес-правило:

- *Только менеджер может управлять заказами*

Состоит из условий:

- *Субъект.Должность = «Менеджер»*
- *Объект.Тип = «Заказ»*

Рисунок 6. Пример условий Бизнес-правила.

Самым главным преимуществом ABAC перед RBAC является то, что данный подход не зависит от сложности и количества условий бизнес-правила. Если появится новый филиал “G”, то всё равно, проверка будет выполняться по атрибуту пользователя, а не по его роли. И с появлением новых филиалов задача не усложнится, ведь проверка по атрибутам разрешает проблему с

неизвестными заранее данными. Будет “F” или “A” филиал, не важно, важно будет только тогда, когда филиал будет “G”.

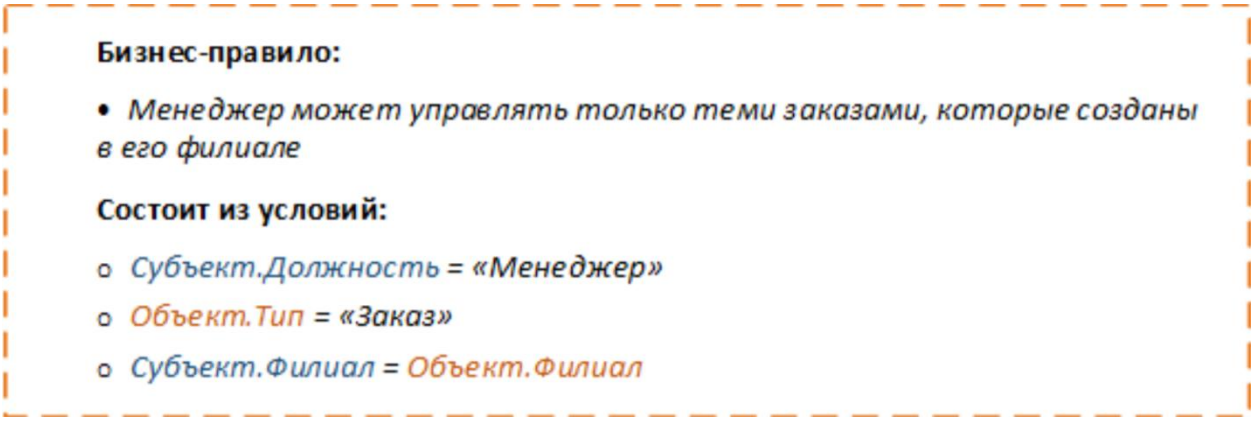

Рисунок 7. Пример условий с Филиалом.

Данный подход ещё примечателен тем, что бизнес-правила, которые представлены, как набор условий, можно использовать для фильтрации данных.

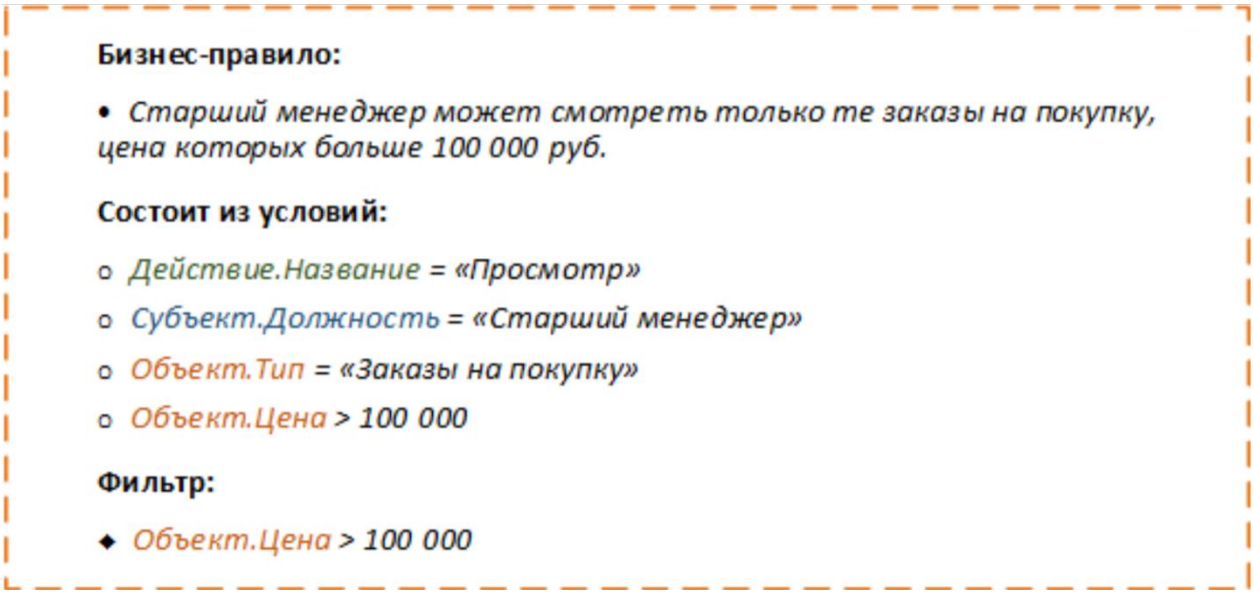

Рисунок 8. Пример бизнес-правила с условием-фильтром.

В данном случае, первые три условия используются для получения доступа к бизнес-правилу, а четвёртое можно использовать как фильтрацию.

Список литературы

1. Node.js® — это кроссплатформенная среда выполнения JavaScript с открытым исходным кодом https://nodejs.org/en

2. Zhang X. et al. The Development and Prospect of Code Clone //arXiv preprint arXiv:2202.08497. – 2022.

3. Deno — A modern runtime for JavaScript and TypeScript https://deno.com/

4. https://ru.reactjs.org/

5. https://habr.com/ru/articles/653553/

6. https://www.npmjs.com/

7. https://vuejs.org/guide/scaling-up/ssr.html

8. Gackenheimer C., Gackenheimer C. Jsx fundamentals //Introduction to React. – 2015. – C. 43-64.

9. https://www.educative.io/answers/what-is-unidirectional-data-flow-in-react

10. Scott Jr E. A. SPA Design and Architecture: Understanding single-page web applications. – Simon and Schuster, 2015.

11. Duldulao D. B., Cabagnot R. J. L. Navigating React Router //Practical Enterprise React: Become an Effective React Developer in Your Team. – Berkeley, CA : Apress, 2021. – C. 55-90.

12. Macrae C. Vue. js: up and running: building accessible and performant web apps. – " O'Reilly Media, Inc.", 2018.

13. Savkin V. Angular router. – Packt Publishing Ltd, 2017.

14. https://reactrouter.com/

15. https://stackoverflow.com/questions/48817305/advantages-of-dynamic-vs-static-routing-in-react

16. https://blog.bitsrc.io/dynamic-vs-static-routing-in-react-49730baaf3e9

17. https://aws.amazon.com/ru/caching/

18. https://blog.logrocket.com/options-caching-react/

19. https://www.tutorialspoint.com/reactjs-shouldcomponentupdate-method

20. https://www.copycat.dev/blog/react-memo/

21. https://www.developerway.com/posts/react-key-attribute

22. https://medium.com/ovrsea/react-re-rendering-purecomponents-and-memoization-553a2cc863c3

23. https://redux.js.org/

24. Bugl D. Learning Redux. – Packt Publishing Ltd, 2017.
25.