***Проектно-конструкторская часть***

1. **Разработка структуры приложения**

Исходя из описания предметной области и функционала, необходимого заказчику, составим физическую схему базы данных.



Рисунок 8. Физическая схема базы данных.

Рассмотрим каждую таблицу(сущность):

1. CarWashes

Назначение: хранение и представление списка автомоек.

Атрибуты: id – уникальный номер для однозначной идентификации автомойки; name(address) – название автомойки, т.к. все автомойки – филиалы одной сети, уникальных названий у них нет, поэтому вводится просто адрес филиала; display – отвечает за отображение филиала в пользовательском интерфейсе (по умолчанию равен “true”), при “удалении” филиала автомойки значение становится false и филиал автомойки перестаёт отображаться в пользовательском интерфейсе.

1. Services

Назначение: хранение и представление списка всех услуг.

Атрибуты: id – уникальный номер для однозначной идентификации услуги; name – наименование услуги; time – длительность оказания услуги; price – стоимость услуги в рублях.

1. CarWashServices

Назначение: реализация связи многие-ко-многим, то есть в одном филиале может оказываться несколько услуг, и каждая услуга может оказываться в разных филиалах; хранение и представление соответствия услуг и филиалов автомойки.

Атрибуты: carWashId и serviceId – уникальный составной ключ для обеспечения семантической целостности, где carWashId – это id автомойки, а serviceId – id услуги; display – отвечает за отображение соответствия филиала и услуги в пользовательском интерфейсе (по умолчанию равен “true”), при “удалении” услуги из филиала значение становится false и данная услуга перестаёт отображаться в данном филиале в пользовательском интерфейсе.

1. Users

Назначение: хранение и представление списка пользователей.

Атрибуты: id – уникальный номер для однозначной идентификации пользователя; name – может быть, как полным ФИО, так и просто именем; phone – номер телефона пользователя, используется для авторизации, и как контактные данные пользователя; password – пароль, указанный пользователем при регистрации, используется для авторизации; role – право доступа для пользователя, имеет тип enum со значениями 'ADMIN' (администратор сети автомоек), 'WORKER' (работник), 'USER' (клиент).

1. Reservations

Назначение: хранение и представление списка автомоек.

Атрибуты: id – уникальный номер для однозначной идентификации записи; carWashId – id автомойки; userId – id пользователя, который записан в автомойку; dateTime – дата и время записи в автомойку; fullPrice – итоговая стоимость всех выбранных пользователем услуг, с учётом индивидуальной скидки; duration – суммарная длительность оказания всех выбранных услуг; status – статус записи имеет тип enum со значениями 'создана' (статус по умолчанию после создания записи), 'отменена' (пользователь, администратор или работник автомойки отменил запись), 'пропущена' (пользователь не явился в назначенное время), 'выполнена' (все выбранные услуги оказаны).

1. ServicesInReservations

Назначение: реализация связи многие-ко-многим, то есть в одной записи может быть выбрано несколько услуг, и каждая услуга может быть выбрана в разных записях; хранение и представление соответствия услуг и филиалов автомойки.

Атрибуты: reservationId и serviceId – уникальный составной ключ для обеспечения семантической целостности, где reservationId – это id записи, а serviceId – id услуги; display – отвечает за отображение соответствия филиала и услуги в пользовательском интерфейсе (по умолчанию равен “true”), при “удалении” услуги из филиала значение становится false и данная услуга перестаёт отображаться в данном филиале в пользовательском интерфейсе.

1. Также в базе данных присутствует ещё одна таблица, она не обозначена на схеме, так как она ни к чему не привязана и данные из неё можно с таким же успехом хранить в конфигурационном файле. Рассмотрим и её.

DefineSales

Назначение: хранение и представление информации о скидках.

Атрибуты: id – уникальный номер для однозначной идентификации скидки; dsale – скидка в %; dsum – общая стоимость записей пользователя со статусом 'выполнена', при которой начинает действовать данная скидка.

1. **Разработка алгоритмов обработки информации**

В общем виде обработка информации происходит так:

1. пользователь в интерфейсе веб-приложения вводит какие-либо данные, нажимает кнопку;
2. отправляется http запрос с вложенными параметрами на сервер;
3. сервер обрабатывает запрос;
4. производится выборка из базы данных согласно параметрам;
5. сервер возвращает строки из базы данных в виде http ответа на клиент;
6. веб-приложение обрабатывает ответ и отображает его в необходимом виде в интерфейсе.

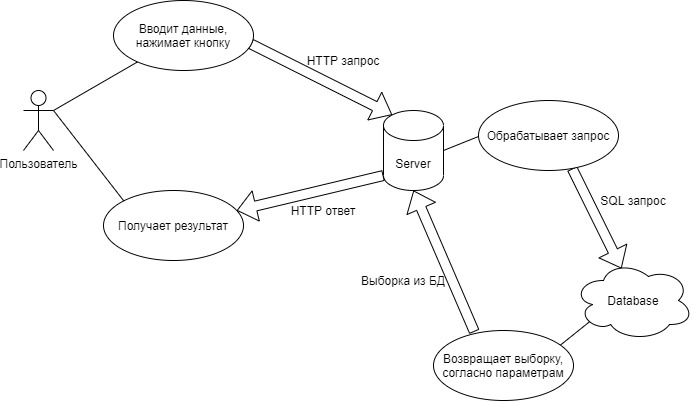


Рисунок 9. Алгоритм запросов.

1. **Разработка архитектуры приложения**

Для каждого обращения к базе данных на клиентской части приложения необходимо объявить http запрос как константу. Все эти константы запросов для приложения пропишем в файле «applicationAPI.js», а для пользователей в файле «userApi.js» и будем экспортировать эти константы запросов при необходимости.

На сервере создадим «mainRouter.js», который будет принимать http запросы и проверяя путь направлять их в соответствующий роутер.

Каждый роутер в зависимости от типа запроса (get/post/put/delete) и возможного добавочного пути будет вызывать соответствующий метод класса из контроллера.

Также для дополнительной безопасности будут использоваться функции промежуточной обработки (middleware). А именно проверка авторизации пользователя и проверка роли. Это очень важно при добавлении или обновлении информации в базе данных. В middleware ещё будет размещён обработчик ошибок, связанных с запросами к базе данных.

Для аутентификации пользователей будут использоваться jwt токены из пакета «jsonwebtoken».

Пароли в базе данных будут храниться в зашифрованном виде с помощью пакета «bcrypt».

Для удобной работы с базой данных, лёгкому переходу на другую базу данных, возможностью работать с данными, как с обычными объектами будет использоваться ORM-библиотека «sequelize».

На клиентской части для хранения результатов запросов из базы данных будут созданы 2 файла: ApplicationStore.js – для хранения информации о приложении (список филиалов автомойки, список услуг и тд.); и UserStore.js – для хранения информации о пользователе (роль, статус).

Интерфейс веб-приложения должен быть модульным, то есть страницы должны состоять из множества частей, собранных в одно целое. Это обеспечит лёгкое исправление ошибок и простоту модернизации веб-приложения в дальнейшем.

1. **Реализация готового приложения**

На сервере создадим файл «db.js», в котором объявим подключение к базе данных с помощью «sequelize». Данные для этого подключения берутся из фала «.env», также в этом файле хранится секретный ключ для создания jwt токена.

Для создания модели базы данных, на сервере была создана папка «models» и в этой папке файл «models.js». В котором с помощью «require» был загружен созданный ранее модуль подключения к базе данных из файла «db.js».

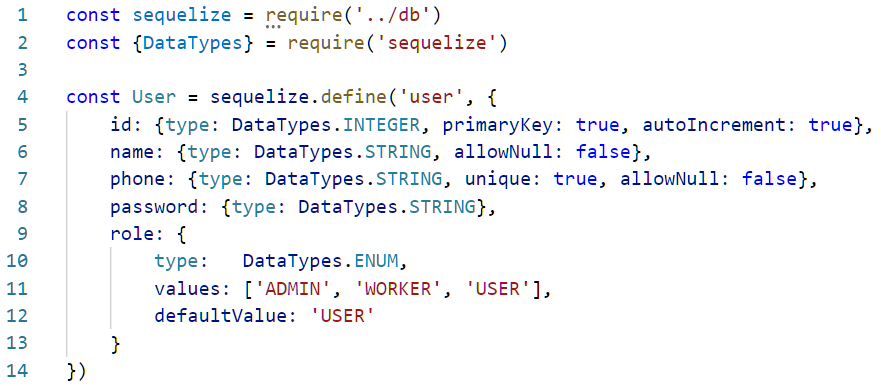


Рисунок 10. Пример создания сущности в базе данных с помощью ORM-библиотеки «sequelize».

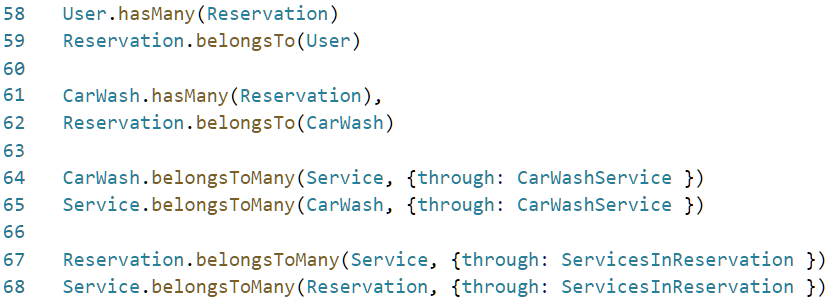


Рисунок 11. Пример описания отношений сущностей в базе данных с помощью ORM-библиотеки «sequelize».

Для удобства и логического разделения кода, каждой сущности будет соответствовать свой роутер и контроллер. Все контроллеры объединены в папку controllers, а все роутеры – соответственно в routes.

В каждом контроллере загружаем соответствующий ему модуль сущности из ранее созданного файла «models.js». И прописываем методы для запросов к базе данных (Рисунок 12). Естественно не все запросы удобно писать через встроенные функции sequelize, а некоторые – просто невозможно. Поэтому в sequelize есть возможность написать запрос вручную (Рисунок 13).

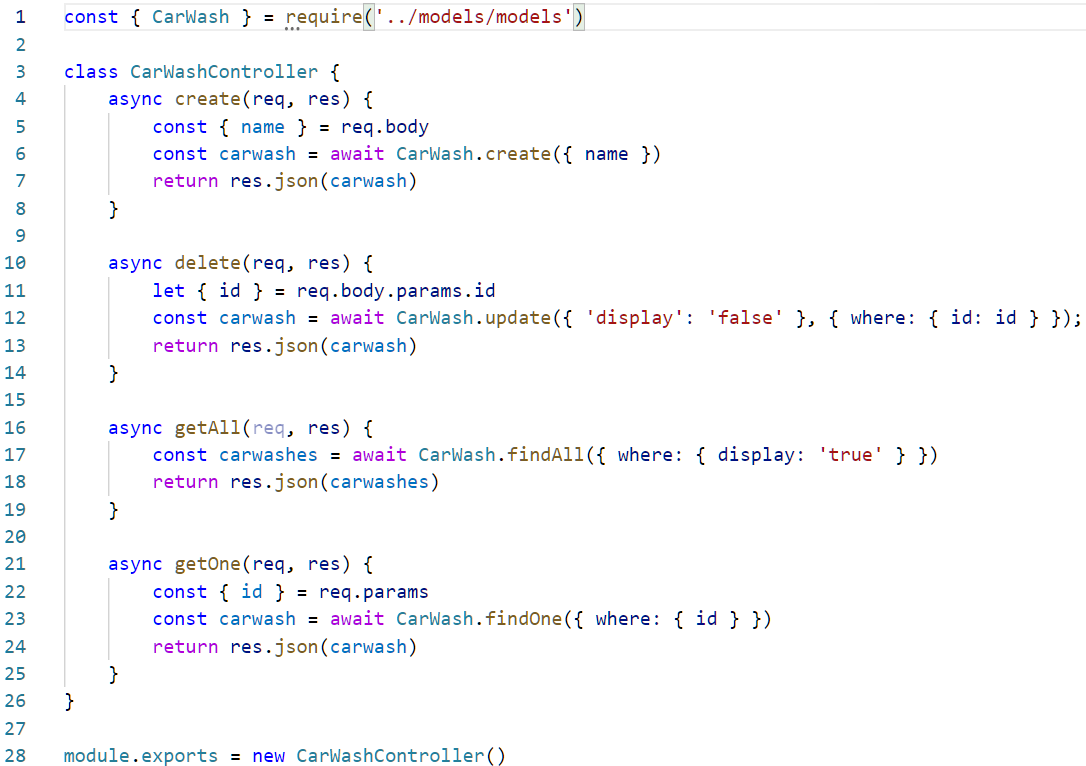


Рисунок 12. Создание контроллера для филиалов автомойки в файле «carWashController.js».

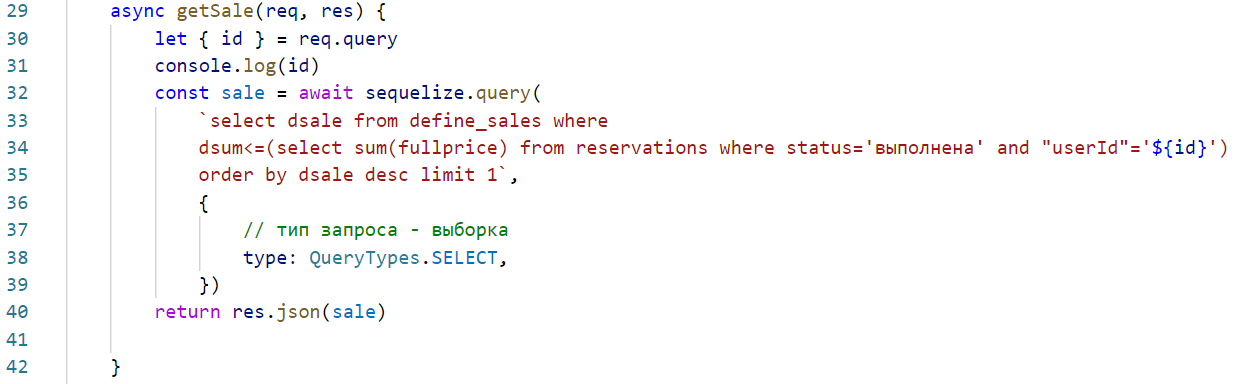


Рисунок 13. Пример написания запроса вручную.

В каждом роутере обрабатываем тип http запроса и добавочный путь, если есть, а также при необходимости используем middleware проверку прав пользователя, отправившего этот запрос. И выбираем подходящий метод из контроллера.



Рисунок 14. Пример обработки http запросов в роутере, файл «carWashRouter.js».



Рисунок 15. Пример первичной обработки http запроса на сервере, и перенаправление в соответствующий ему роутер, файл «mainRouter.js».

На текущем этапе разработки в базе данных не предусмотрено никаких дополнительных проверок и валидаций полей, так как разработка приложения производится одним человеком, который сам и проектировал эту базу данных. Также все необходимые валидации полей, например, формат номера телефона пользователя, незаполненные поля и тд. проверяются с помощью JavaScript. В будущем, при необходимости расширения приложения можно будет добавить дополнительные триггеры, которые будут проверять данные перед записью в таблицу. В случае sequelize это можно сделать с помощью Hooks.

1. **Разработка интерфейса взаимодействия пользователя с системой**

В интерфейсе данного веб-приложения 5 страниц:

1. Главная страница
2. Регистрация и авторизация
3. Личный кабинет администратора
4. Личный кабинет пользователя
5. Личный кабинет работника

Каждая страница состоит из отдельных компонентов, которые хранятся на клиенте в каталоге «components». Каждый компонент имеет свой функционал и назначение, это отражено в его названии. Например: «CarWashBar.js» служит для отображения панели с возможность выбрать филиал автомойки, благодаря этому компоненту пропадает нужда постоянно дублировать код при необходимости отображения панели филиалов на экране. Также в каталоге «components» есть подкаталог «modals», в котором хранятся модальные окна, эти окна отображаются поверх страниц при нажатии соответствующей им кнопке на странице. Их цель также убрать дублирование кода, например: «ChangePassword.js» написанный один раз, легко добавляется на страницы личных кабинетов и позволяет изменять свой пароль, как простым пользователям, так и администратору и работникам сети автомоек.

При разработке интерфейса страниц, их компонентов и модульных окон были использованы такие компоненты react-bootstrap как:

1. Modal
2. Button
3. Dropdown
4. ListGroup
5. ButtonGroup
6. Container
7. Form
8. Row
9. Col
10. Navbar
11. Nav
12. Table
13. OverlayTrigger
14. Tooltip
15. Card
16. Carousel
17. Spinner

Большинство элементов интерфейса обёрнуто в «observer» для автоматической перерисовки их в режиме реального времени без обновления страницы.

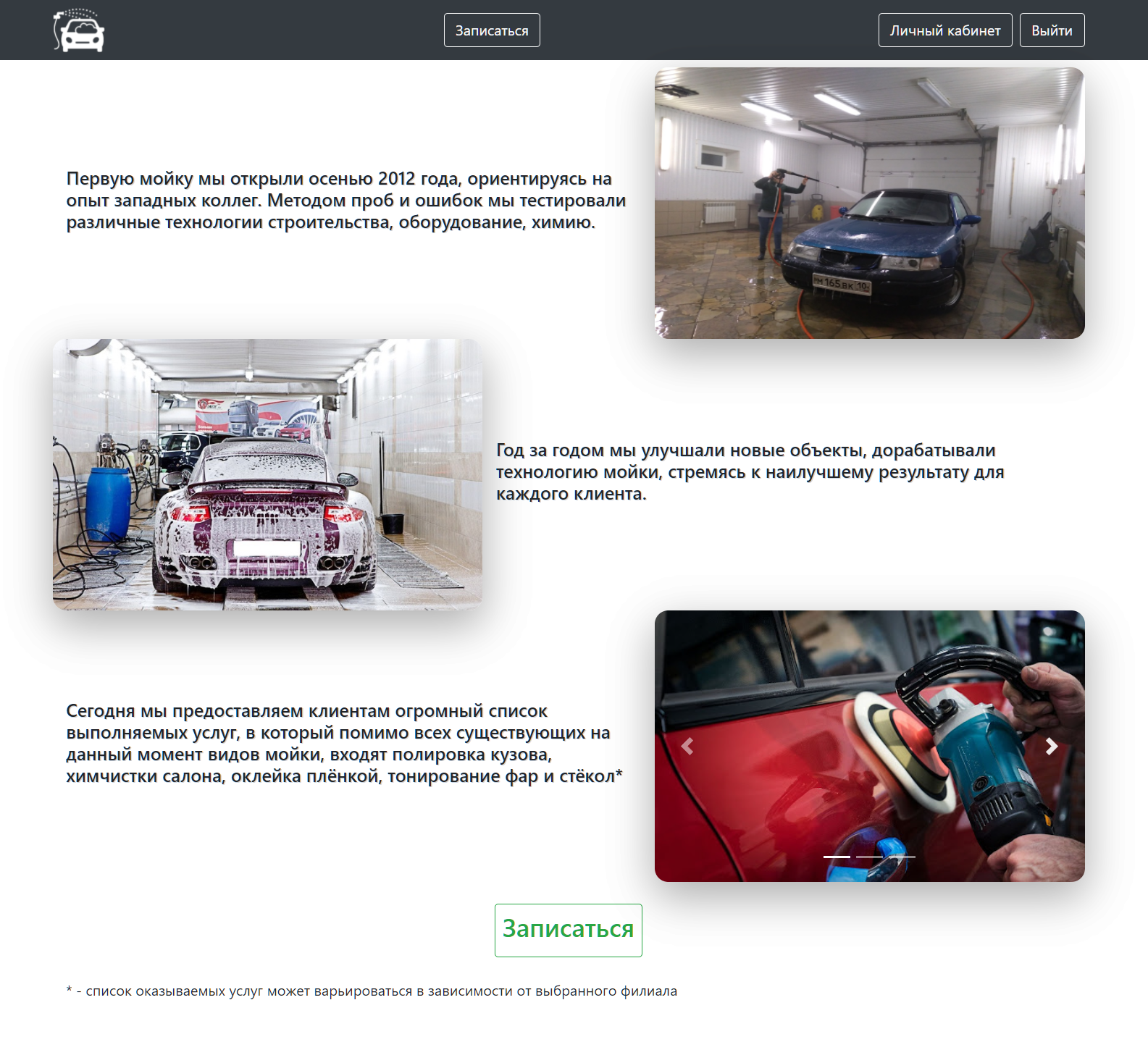


Рисунок 16. Главная страница веб-приложения.

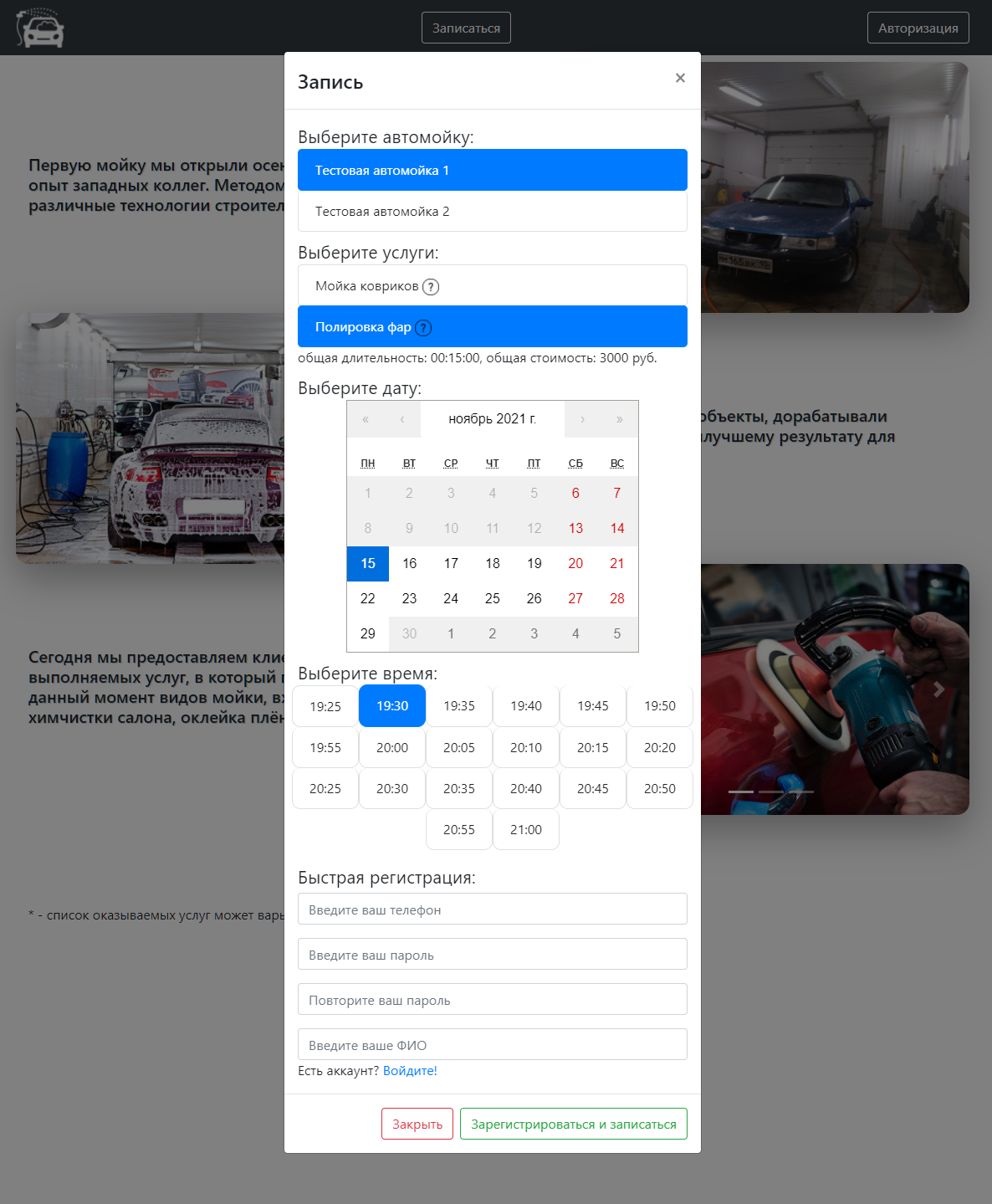


Рисунок 17. Модальное окно записи в автомойку не авторизованного пользователя.

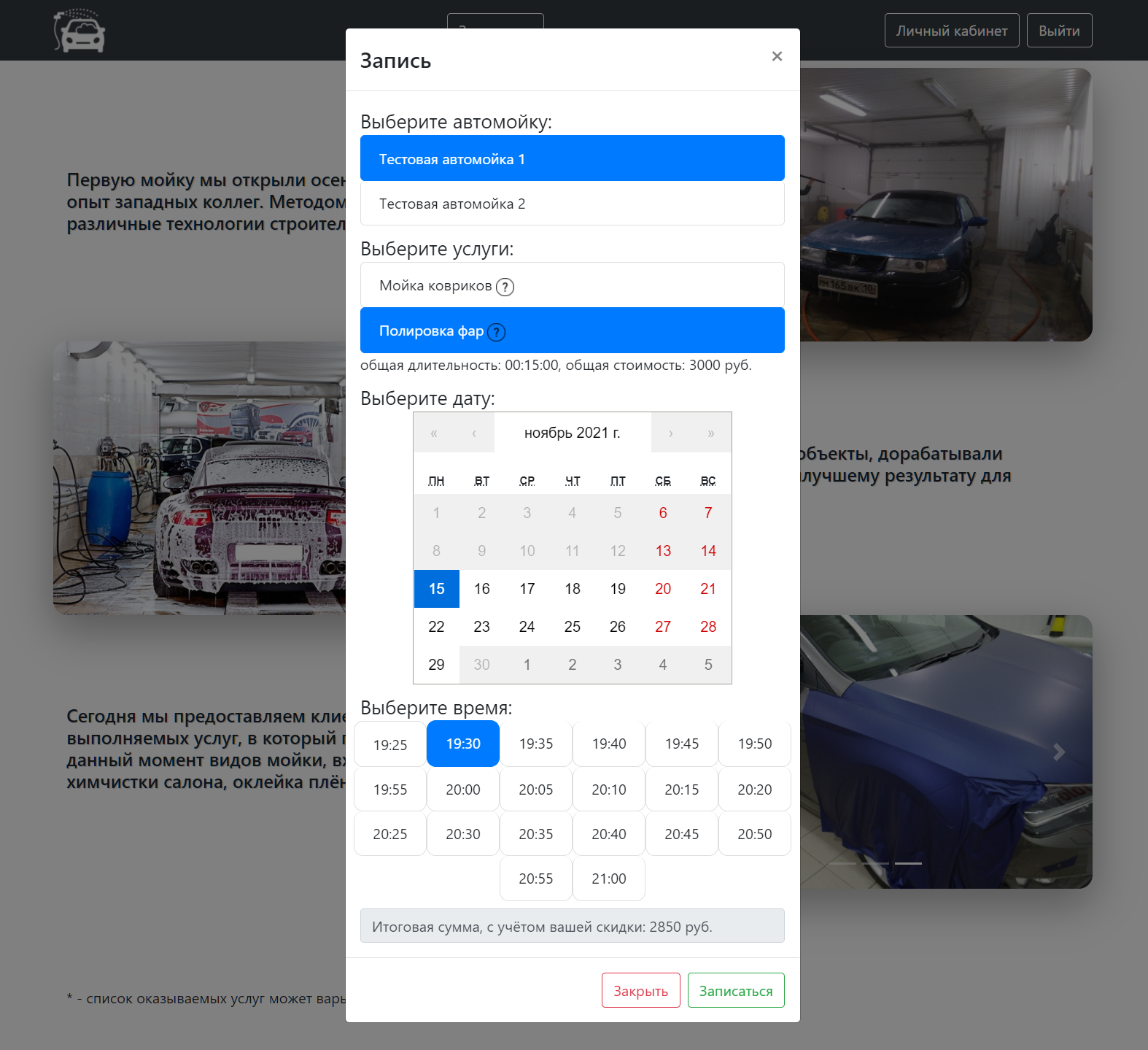


Рисунок 18. Модальное окно записи в автомойку если пользователь авторизовался.

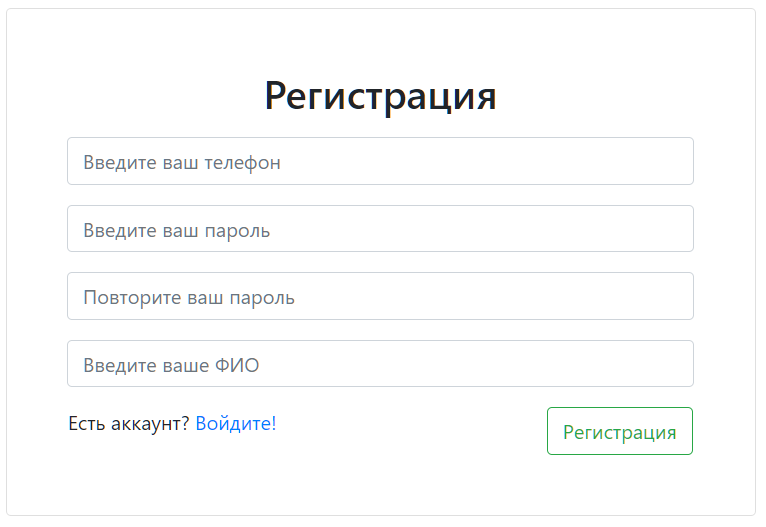
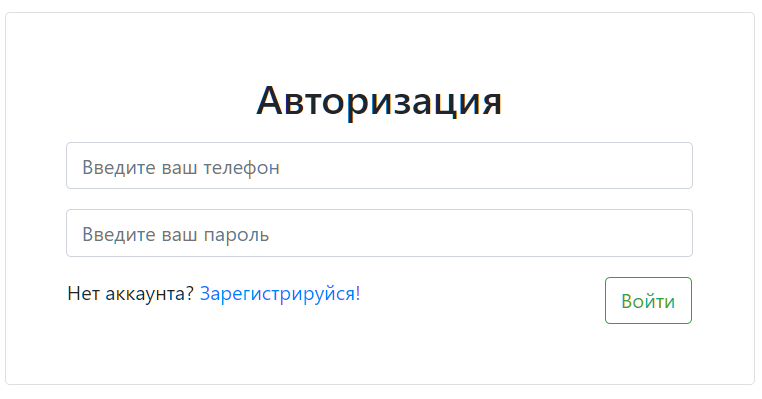


Рисунок 19. Формы регистрации и авторизации.

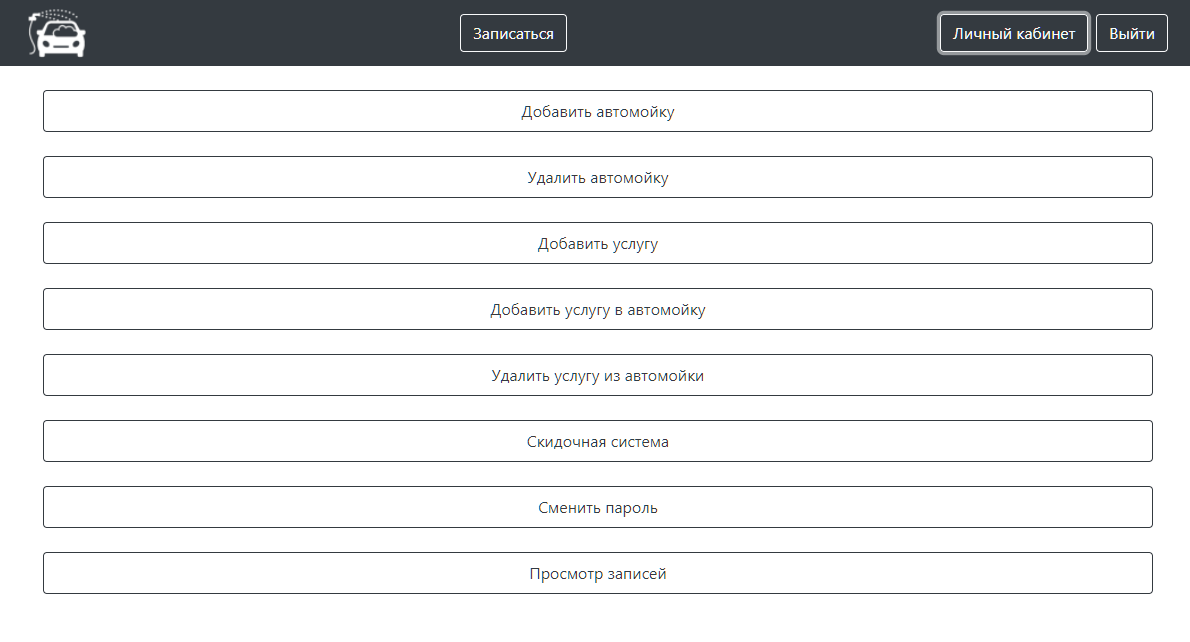
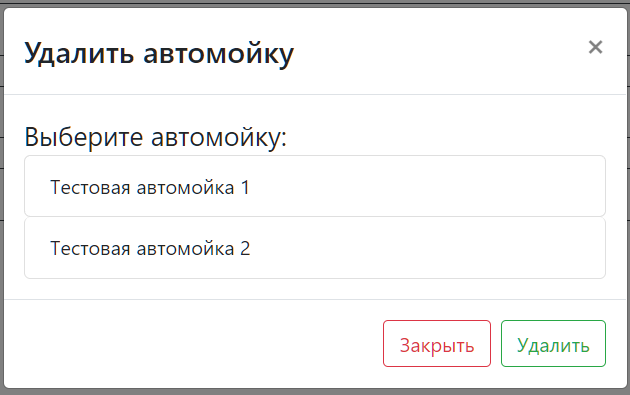
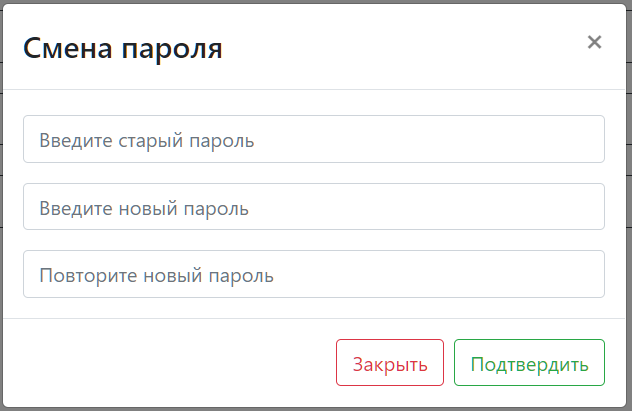
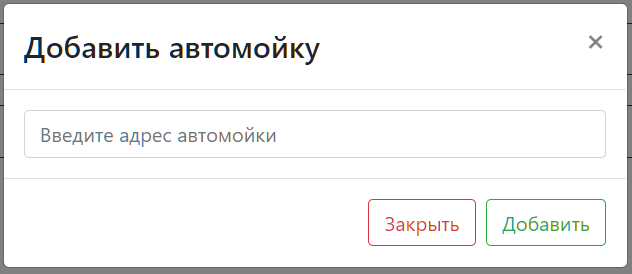


Рисунок 20. Личный кабинет администратора.



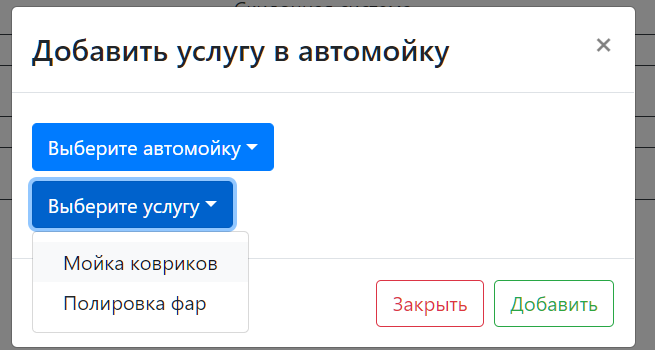
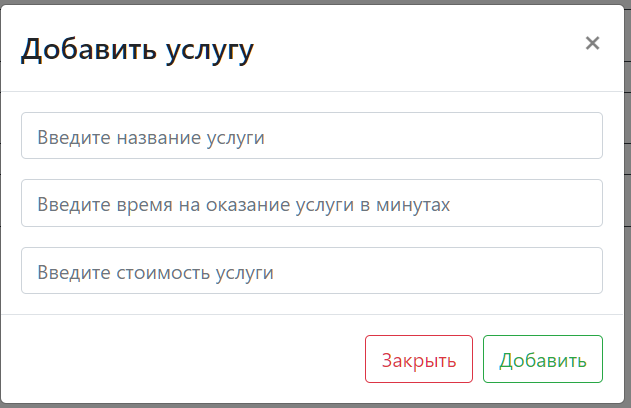
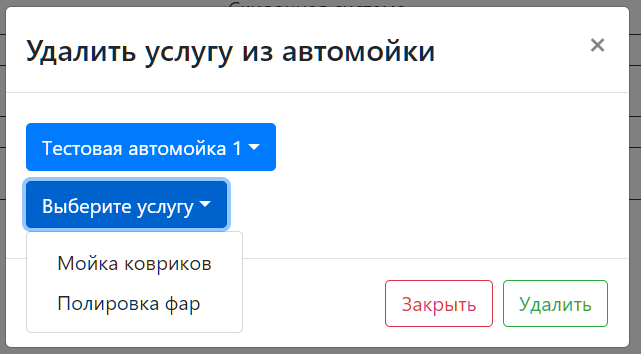
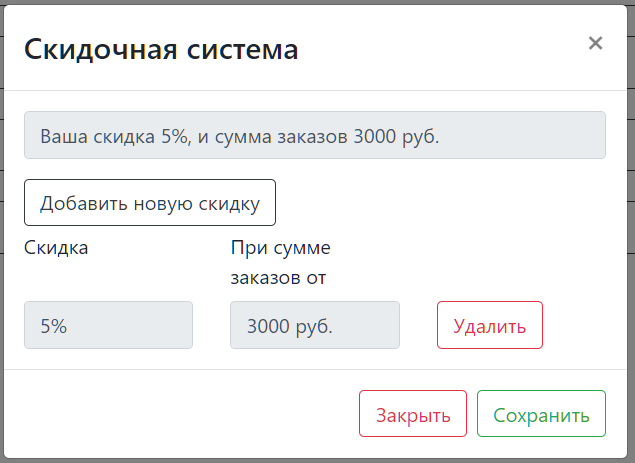
 

Рисунок 21. Все модальные окна из личного кабинета администратора.

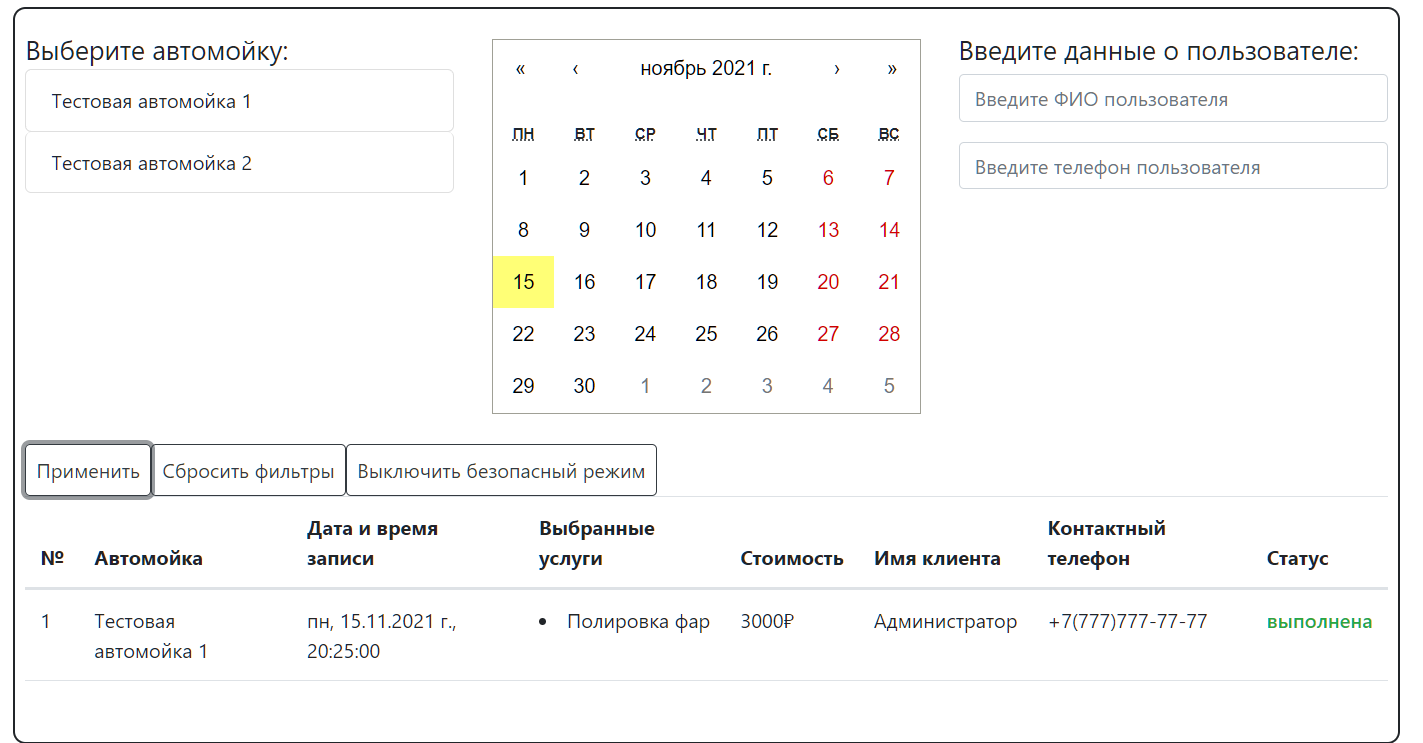


Рисунок 22. Просмотр всех записей с возможностью использования фильтров (нижняя кнопка в панели администратора).

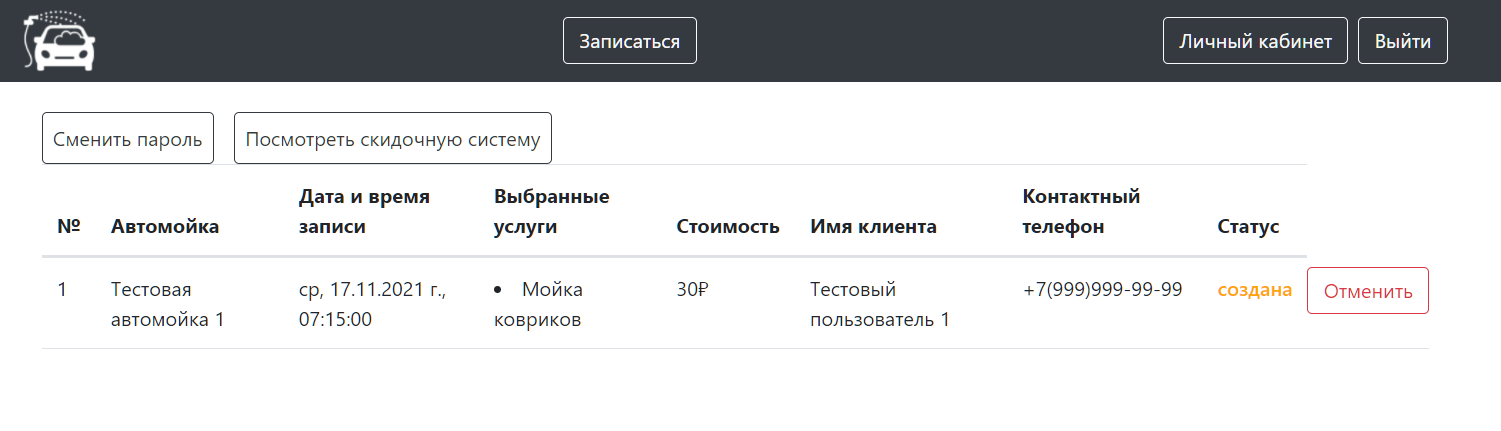


Рисунок 23. Личный кабинет пользователя.

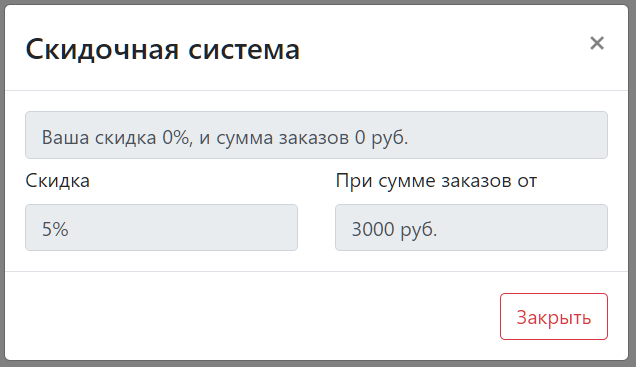


Рисунок 24. Вид скидочной системы для пользователя.

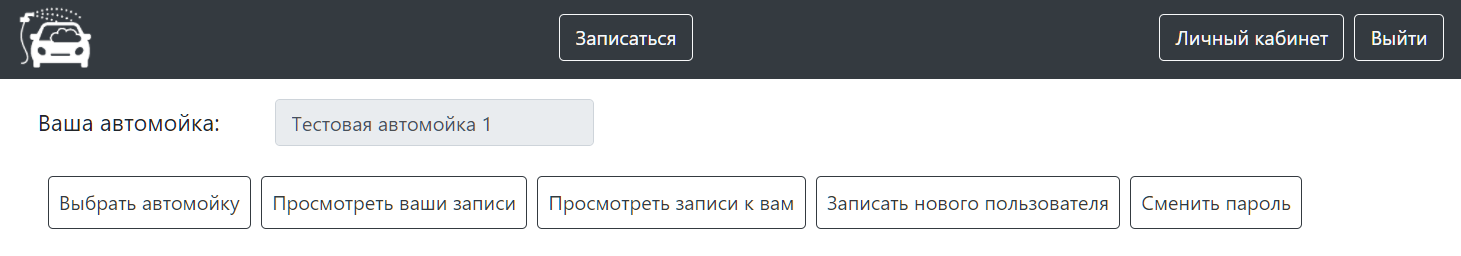


Рисунок 25. Личный кабинет работника.

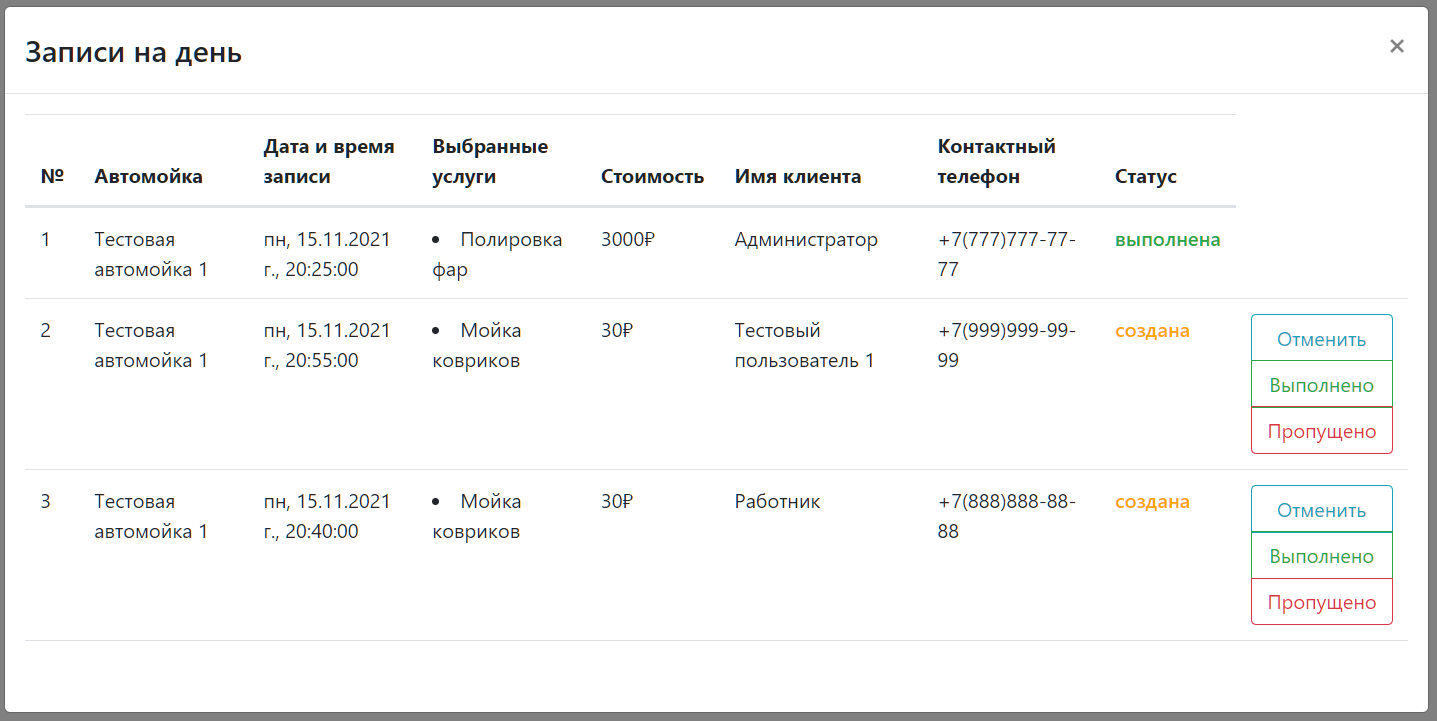


Рисунок 26. Просмотр работником записей на текущий день.

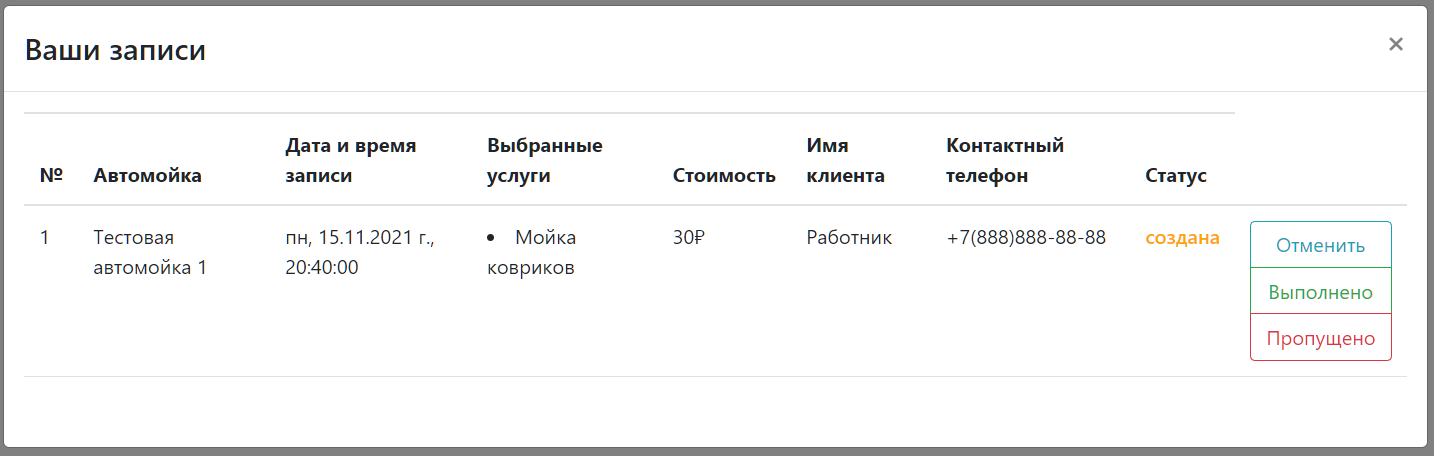


Рисунок 27. Просмотр работником всех своих записей.