**3. ПРОЕКТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ**

1. **Разработка структуры приложения**

Для разработки приложения использовалась среда Microsoft Visual Studio Code, включающая в себя широкий набор функциональных инструментов.

Для представления принципа работы приложения ниже приведена ее

UML диаграмма вариантов использования (рисунок 3.1).

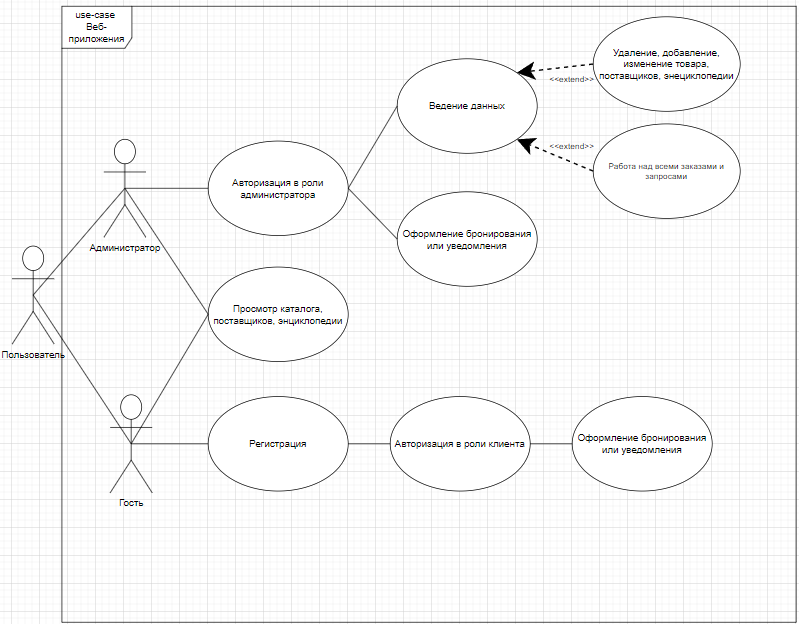


Рисунок 3.1 - Диаграмма использования веб-приложения

При запуске веб-приложения пользователь сам решает, что ему необходимо сделать. Он может, как и просматривать каталог и всю информацию на сайте без входа и регистрации в роли гостя, так и войти на сайт и оформить бронирование или запросить уведомление может выбрать, какое действие ему совершить в роли клиента.

В любой момент работы с веб-приложением, пользователь может выйти из него.

Данные на сайте добавляются, удаляются и обновляются с помощью администратора. У администратора есть все необходимые для этого возможности. Помимо этого, администратор также может работать над существующими заказами и запросами.

Общее назначение программного средства – предоставление возможности просмотра информации о попугаях, клетках, поставщиках, а также произведение бронирования необходимых товаров.

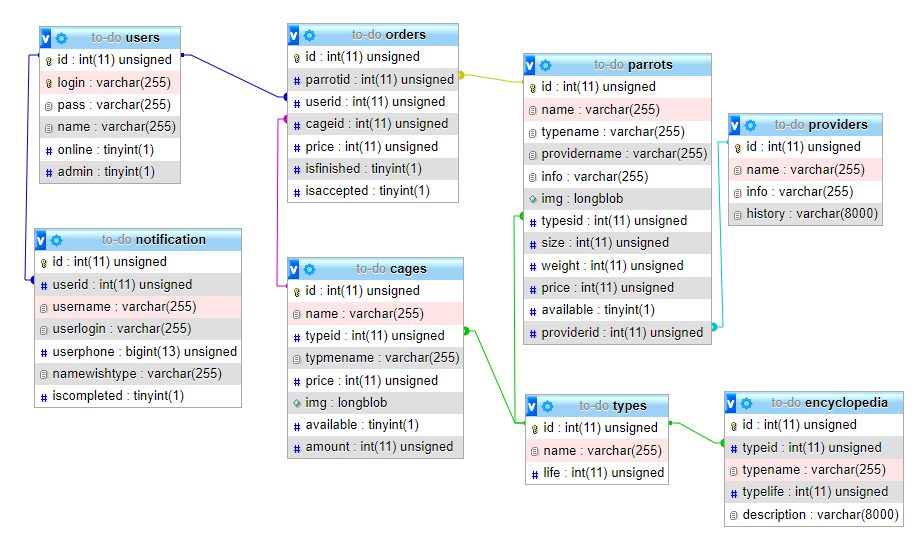
Реализуемая задача состоит в том, чтобы хранение всей информации, необходимой для существования веб-приложения осуществлялось в базе данных, а также производилось удобное взаимодействие с пользовательским интерфейсом.

Рисунок 3.2 – Физическая модель данных

1. **Логическая схема базы данных**

На рисунке 3.3 представлена логическая схема проектируемой базы данных.

Таблица «users» содержит информацию о пользователе (логин, пароль, имя, онлайн он или нет, является ли он администратором или нет).

Таблица «encyclopedia» содержит информацию о энциклопедии. Здесь осуществляется хранение видов, средней продолжительности жизни, описания попугаев.

Таблица «providers» содержит информацию о поставщиках. Здесь происходит хранение имени, описание, история поставщика.

Таблица «types» содержит информацию о видах попугаев. Здесь хранится имени и средней продолжительности жизни попугая.

Таблица «parrots» содержит информацию о попугаях. Здесь осуществляется хранение имени, вида, поставщика, описания, цены, изображения, доступности, размера и веса попугаев.

Таблица «cages» содержит информацию о клетках. Здесь происходит хранение имени, цены, изображения, доступности и количества клеток и вида попугая для этой клетки.

Таблица «orders» хранит информацию о заказе (бронировании). Здесь хранится id товар (либо id попугая, либо id клетки, чтобы не было конфликтов при создании заказов, в структуру базы данных был добавлен следующий запрос рис. 3.3), пользователя, который сделал заказ, цену на товар, состояние заказа (завершен ли он, успешно ли принят или отклонен).

Таблица «notification» содержит информацию о уведомлениях, которые сделали пользователи (например, запрос об уведомлении при появлении в наличии требуемого вида попугая). Здесь хранится информация пользователе, которой делал запрос уведомления, также имя желаемого (запрашиваемого) попугая и завершен или нет запрос (уведомление).

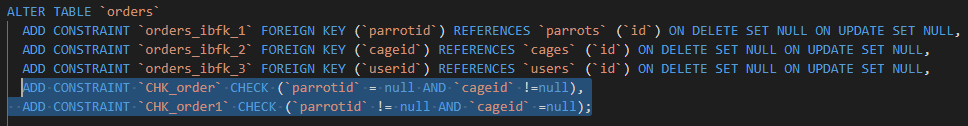


Рисунок 3.3 – Проверка на правильность входных данных

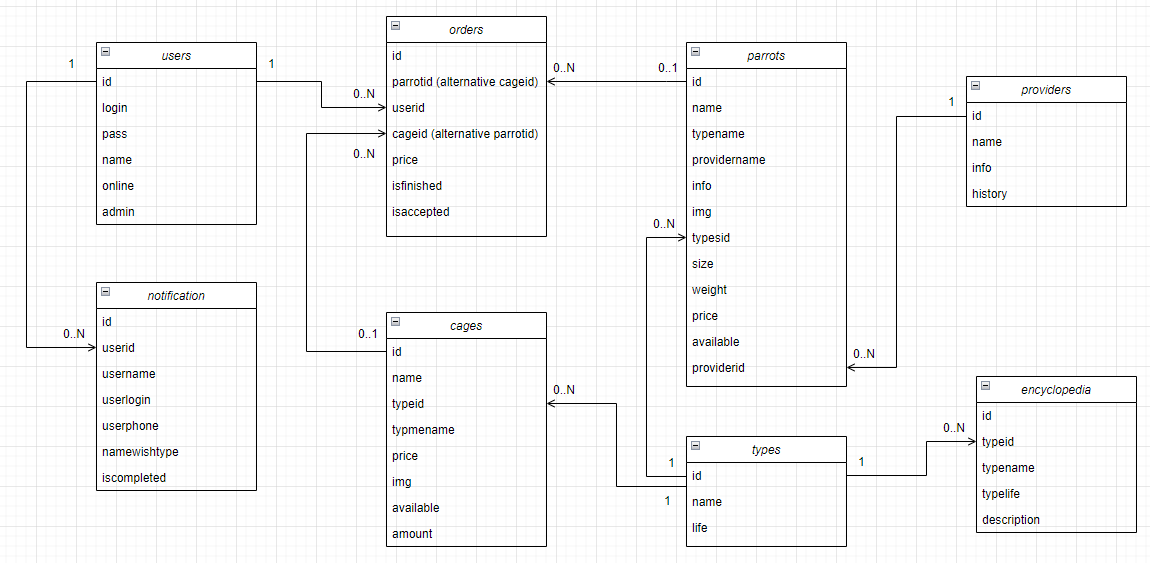


Рисунок 3.4 — Логическая схема проектируемой БД

1. **Разработка архитектуры приложения**

Для разработки данного веб-приложения в курсовой работе использовался паттерн MVC.

Паттерн MVC (Model-View-Controller или «модель-вид-контроллер») – это шаблон архитектуры ПО, который подразумевает разделение программы на три слабосвязанных компонента, каждый из которых отвечает за свою сферу деятельности.

Паттерн MVC включает ряд компонентов:

1. **Model** – модель, предоставляющая доступ к данным. Позволяет извлекать данные и менять их состояние;
2. **View** – представление, отображающее данные клиенту. В веб-программировании существует в виде конечных данных, которые получает клиент;
3. **Controller** – контроллер, отслеживающий различные события (действия пользователя) и по заданной логике оповещающий модель о необходимости изменить состояние системы.
4. **Система маршрутизации** – дополнительный компонент, который сопоставляет запросы с маршрутами и выбирает определенный контроллер для обработки запросов.

Ключевая идея MVC состоит в том, что любое приложение с пользовательским интерфейсом в первом приближении можно разбить на два модуля:

1. модуль, отвечающий за реализацию бизнес-логики приложения;
2. пользовательский интерфейс.

В первом модуле будет реализован основной функционал приложения. Данный модуль будет ядром системы, в котором реализуется модель предметной области приложения. В концепции MVC данный модуль будет обозначать модель (Model).

Во втором модуле будет реализован весь пользовательский интерфейс, включая отображение данных пользователю и логику взаимодействия пользователя с приложением.

Основная цель паттерна MVC — отделить реализацию бизнес-логики приложения (модели) от ее визуализации (вида). Такое разделение повысит возможность повторного использования кода.

Польза применения MVC наиболее наглядна в случаях, когда пользователю нужно предоставлять одни и те же данные в разных формах. Например, в виде таблицы, графика или диаграммы (используя различные виды). При этом, не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью по кнопке, ввод данных).

Если следовать паттерну MVC, можно упростить написание программ, повысить читаемость кода, сделать легче расширение и поддержку системы в будущем.

Приложению приходит запрос и система маршрутизации в данном случае необходима для выбора конкретного контроллера, который нужен для обработки запроса. После чего контроллер запускает процесс обработки запроса. Пока контроллер обрабатывает запрос, он может обращаться к данным, используя модели (Model), а как визуализацию ответа использовать представления (View).

Обработка контроллера имеет конечный результат, который и отправляется в качестве ответа клиенту. Ответ представляет собой html-страницу, которую пользователь видит в своем браузере.

Связь приложения и сервера осуществляется через протокол HTTP и происходит по принципу REST.

HTTP («HyperText Transfer Protocol» или «протокол передачи гипертекста») – протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

REST («Representational State Transfer» или «передача репрезентативного состояния») – это архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

Свойства архитектуры REST:

1. Масштабируемость для обеспечения большого числа компонентов и взаимодействий компонентов;
2. Открытость компонентов к возможным изменениям;
3. Прозрачность связей между компонентами системы для сервисных служб;
4. Переносимость компонентов системы путем перемещения программного кода вместе с данными;
5. Надёжность, выражающаяся в устойчивости к отказам на уровне системы при наличии отказов отдельных компонентов, соединений или данных.

Система REST работает на уровне HTTP и использует следующие методы для осуществления работы с ресурсами на сервере:

* POST – для создания ресурса;
* GET – для получения ресурса;
* PUT – для обновления ресурса;
* DELETE – для удаления ресурса.

В случае веб-приложения про попугаев необходима возможность добавления новых попугаев, их поставщиков, клеток для них, просмотра всех уже существующих данных на сайте и просмотра всех заказов, редактирование уже существующей информации.

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

Для удобного взаимодействия пользователя был разработан следующий интерфейс.

Если пользователь является гостем, ему предоставляется все те же возможности, как и зарегистрированному пользователю, за исключением функции бронирования попугаев или клеток, а именно – просмотр попугаев (рис. 3.4) и клеток (рис. 3.5), просмотр поставщиков (рис. 3.6) и информации о них (рис. 3.7), а также просмотр энциклопедии (рис. 3.8). Если же, не зарегистрированный пользователь захочет сделать бронирование (клетка или попугай, не важно), ничего не получится (рис 3.9). Для успешного бронирования необходимо либо зарегистрироваться под новым пользователем (рис. 3.10), либо зайти уже под существующим (рис. 3.11). После авторизации, пользователь сможет совершить бронирование (рис. 3.12). Также произойдет автоматическое заполнение полей имени и логина пользователя, под которым осуществилась авторизация.

Также для обновления информации на сайте, необходим администратор с возможностями управления веб-приложением. У администратора есть доступ к странице администрирования, при помощи нажатия на специальную кнопку, которая доступна только ему (рис. 3.13). Он обладает рядом возможностей для успешного управления сайтом, а именно просмотр заказов, запрашиваемых попугаев, редактирование энциклопедии, изменение количества клеток, добавление или удаление попугаев, клеток, а также поставщиков (рис. 3.14). Также стоит привести пример написания и обработки запроса для вывода попугаев с определенным id (рис. 3.15).

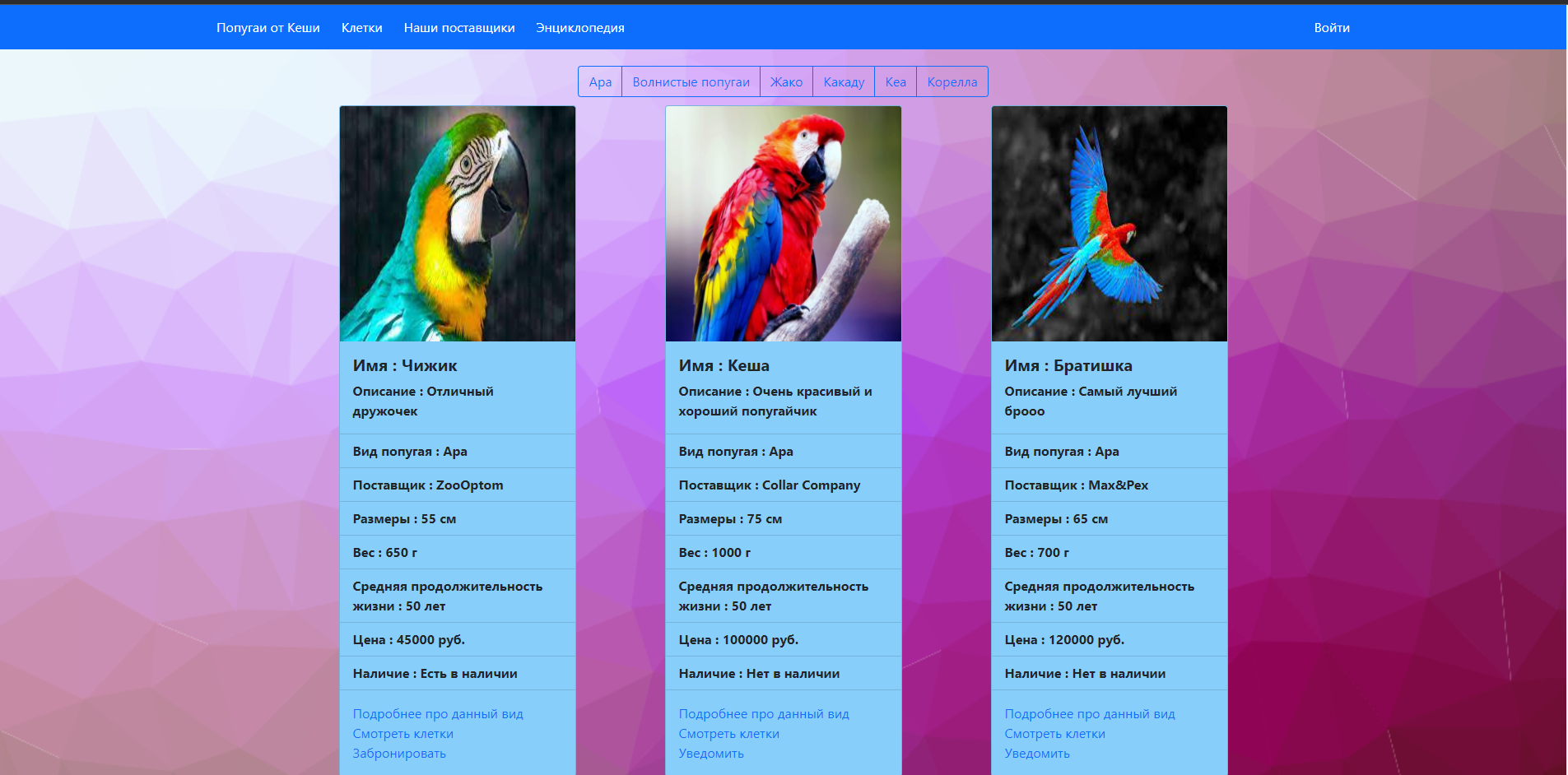


Рисунок 3.5 – Просмотр попугаев

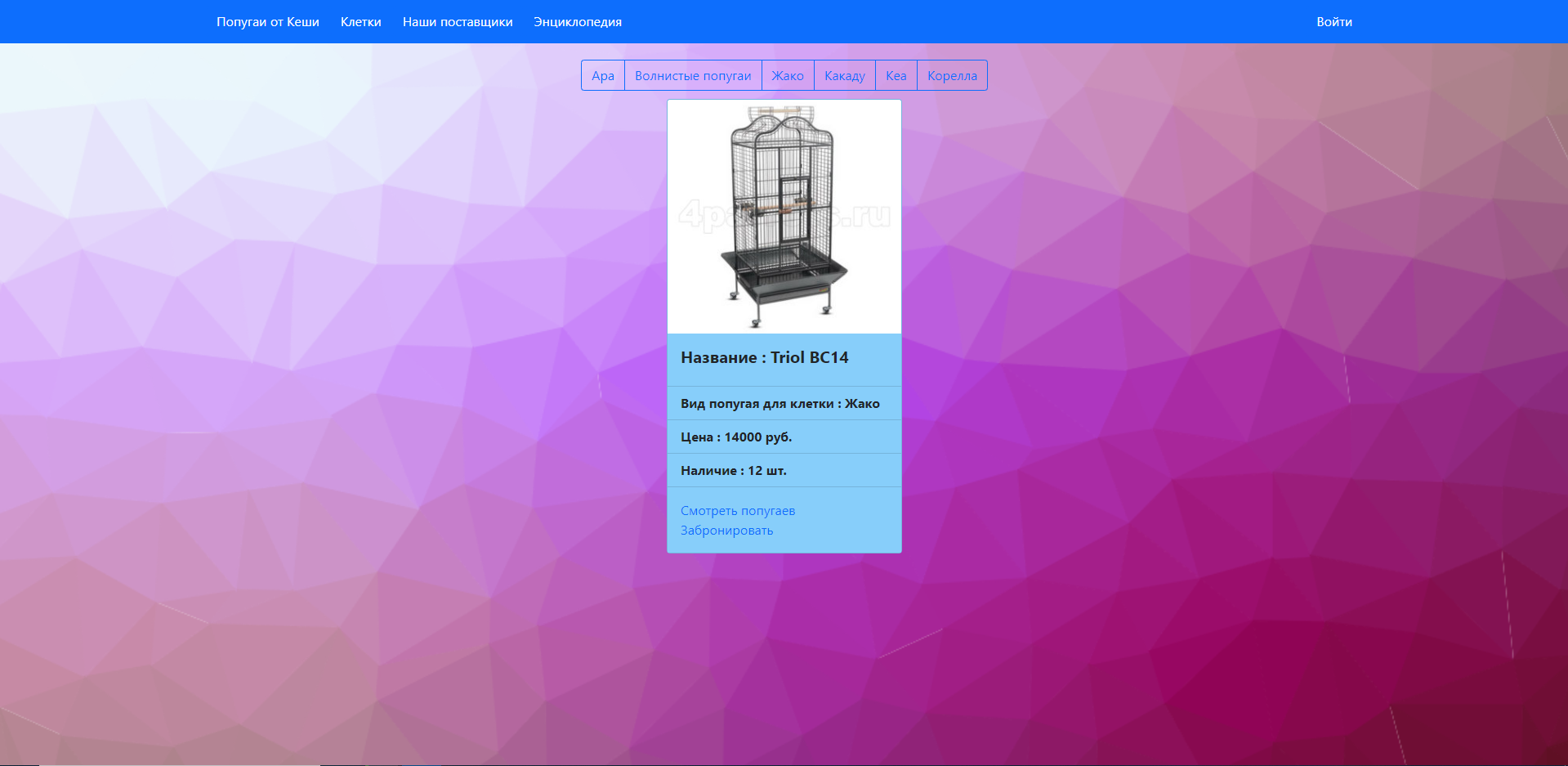


Рисунок 3.6 – Просмотр клеток

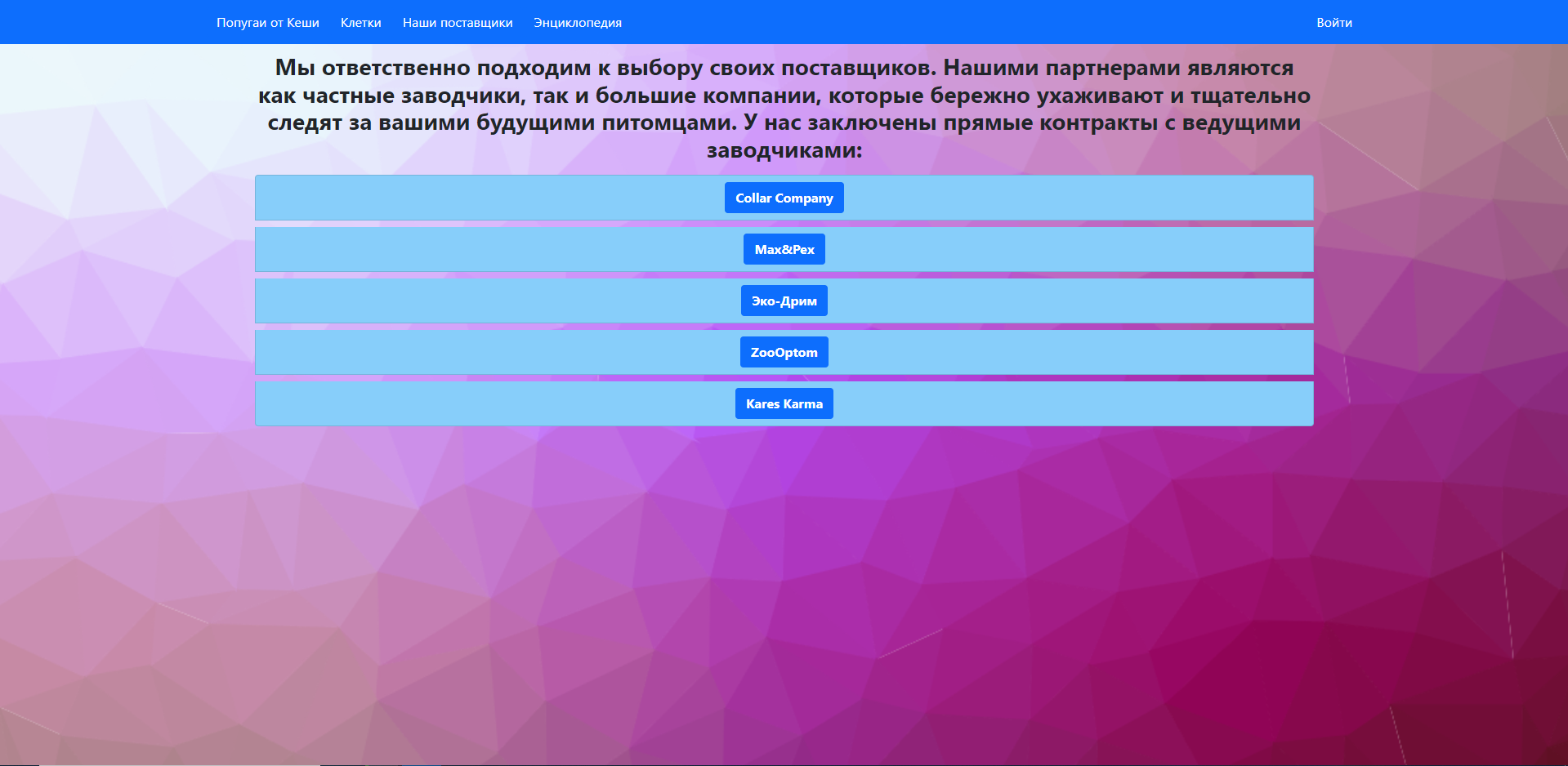


Рисунок 3.7 – Просмотр поставщиков

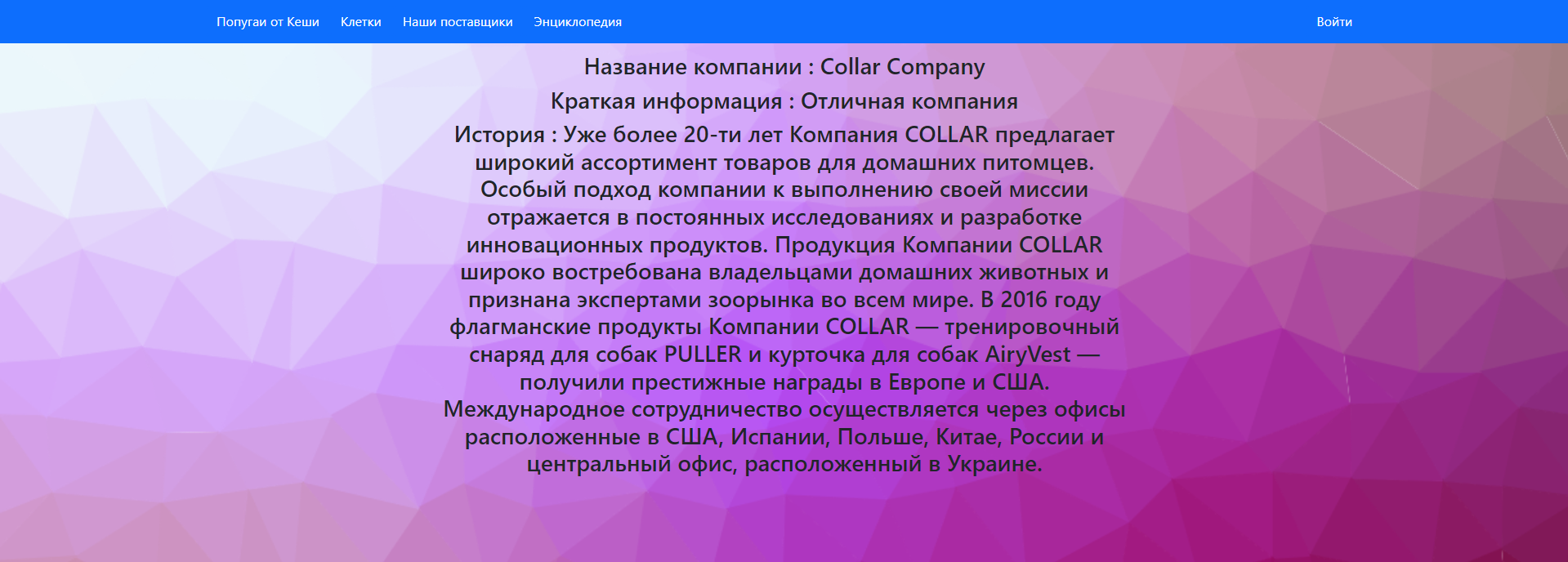


Рисунок 3.8 – Просмотр информации о поставщике

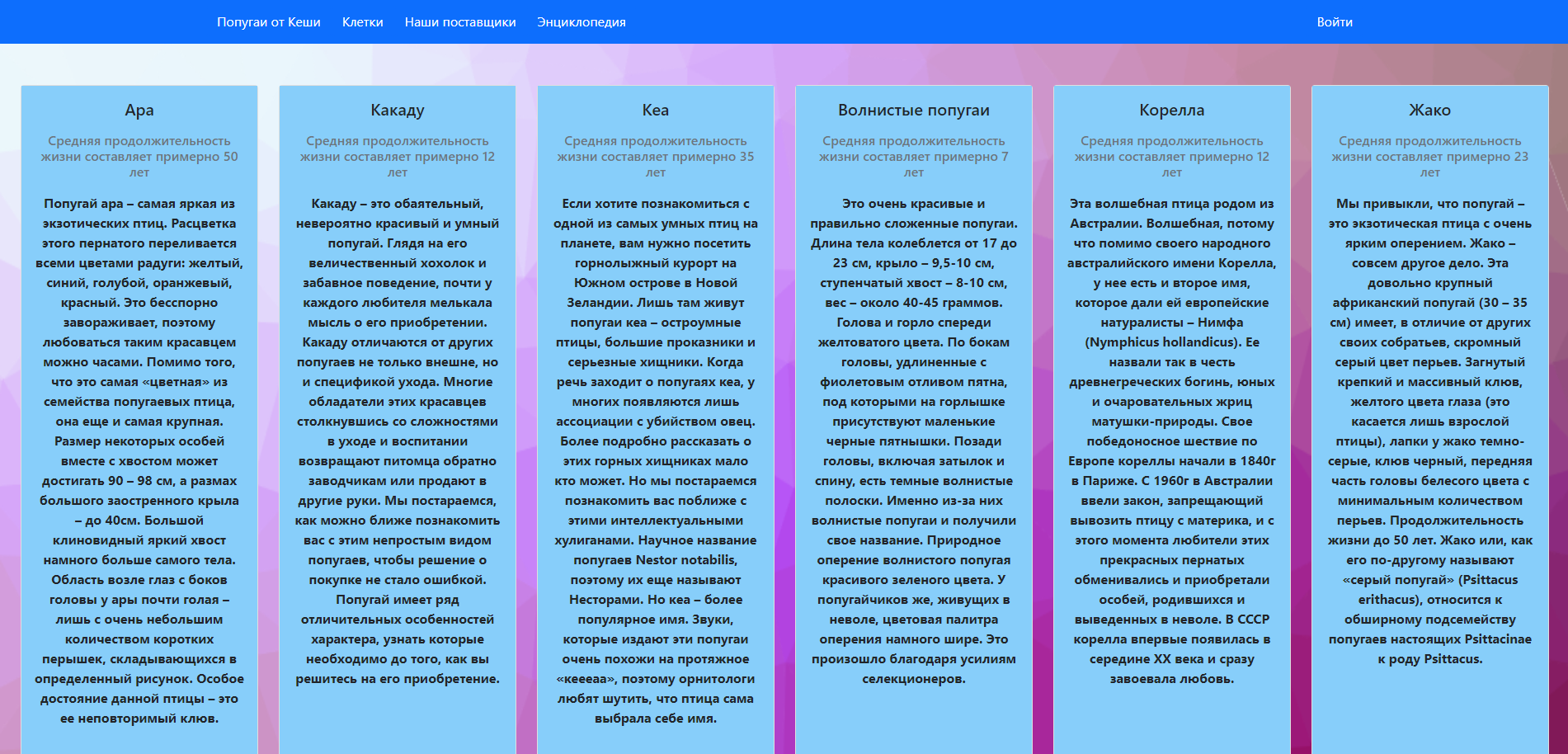


Рисунок 3.9 – Просмотр энциклопедии

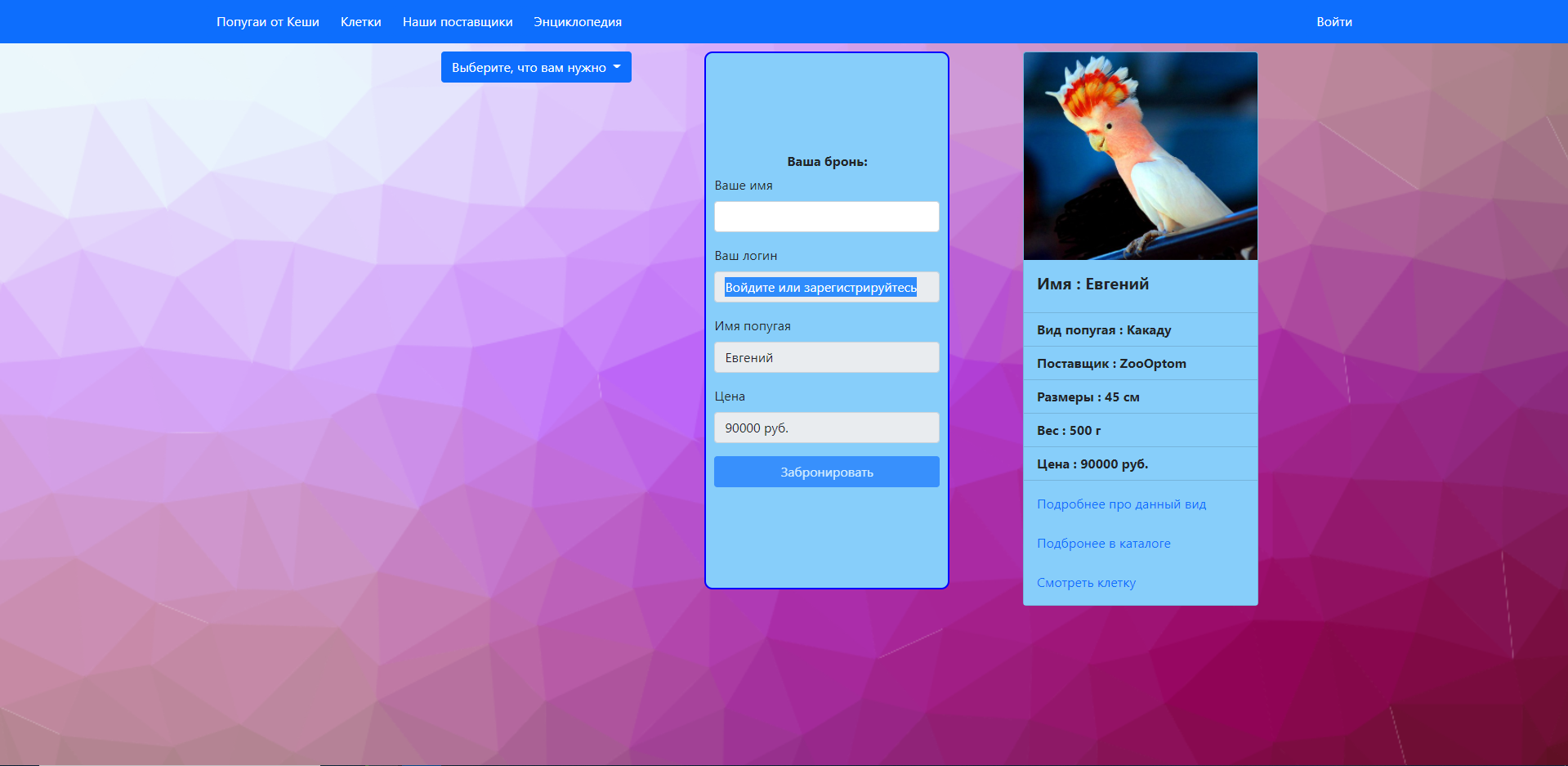


Рисунок 3.10 – Невозможность бронирования незарегистрированному пользователю

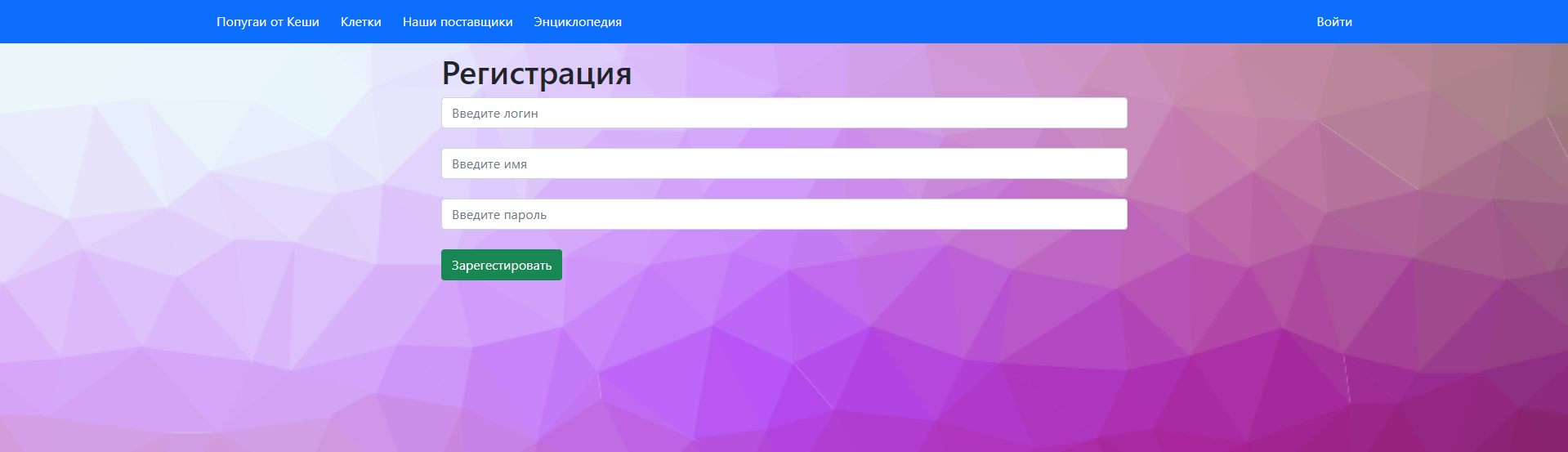


Рисунок 3.11 – Регистрация нового пользователя

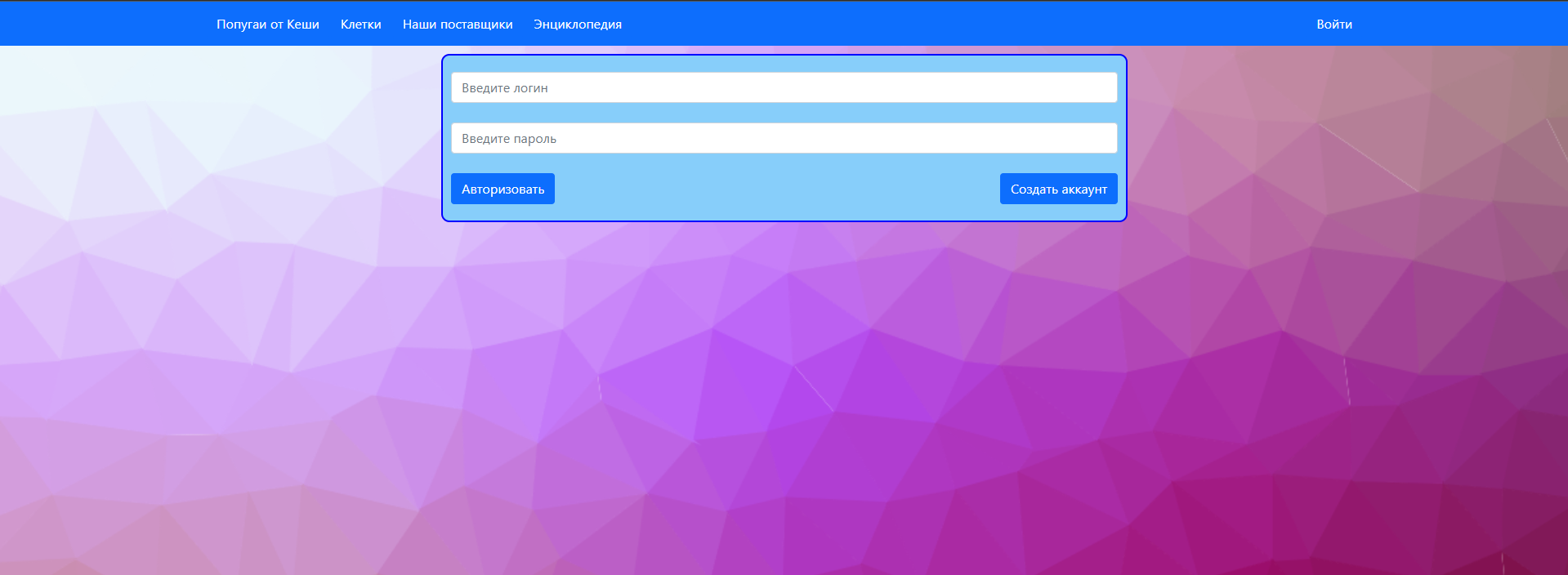


Рисунок 3.12 – Авторизация существующего пользователя

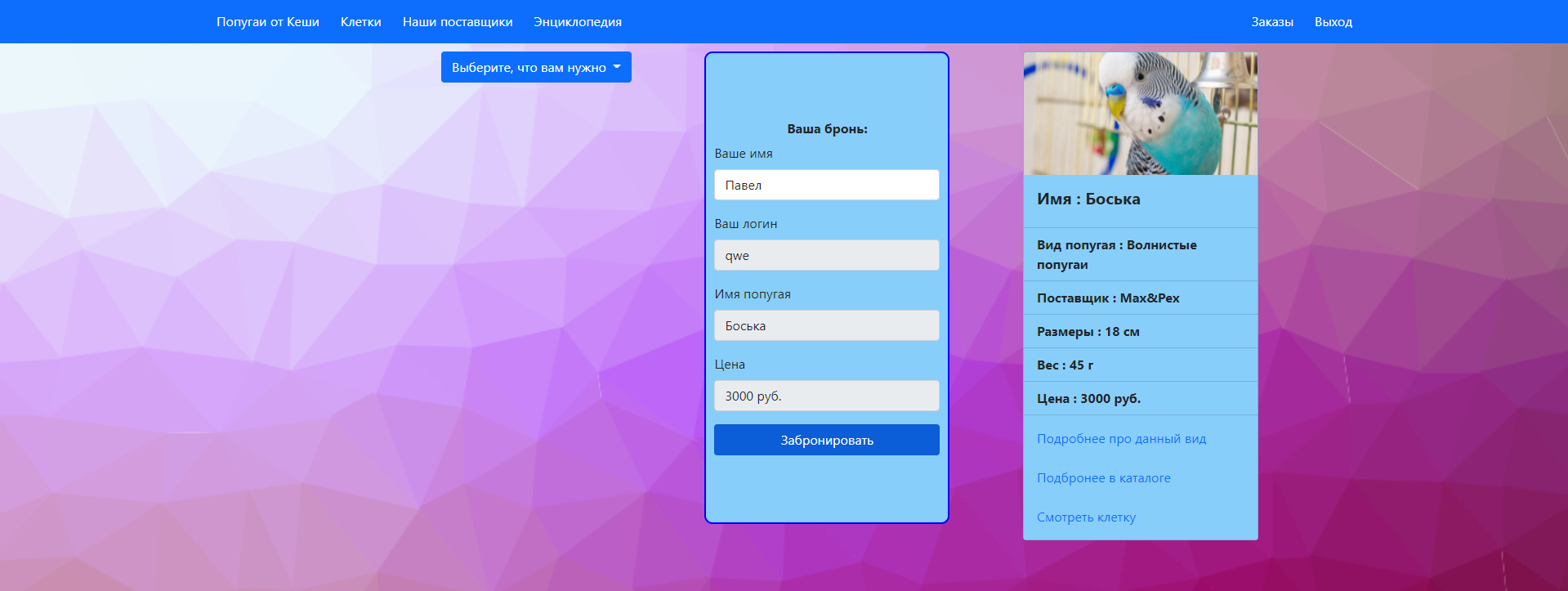


Рисунок 3.13 – Бронирование

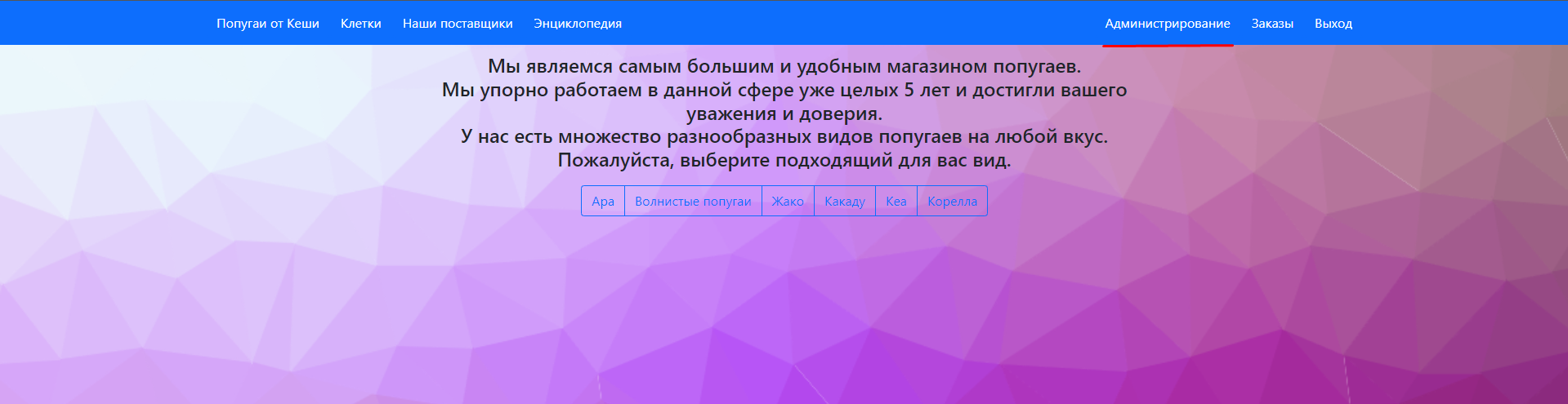


Рисунок 3.14 – Кнопка администрирования

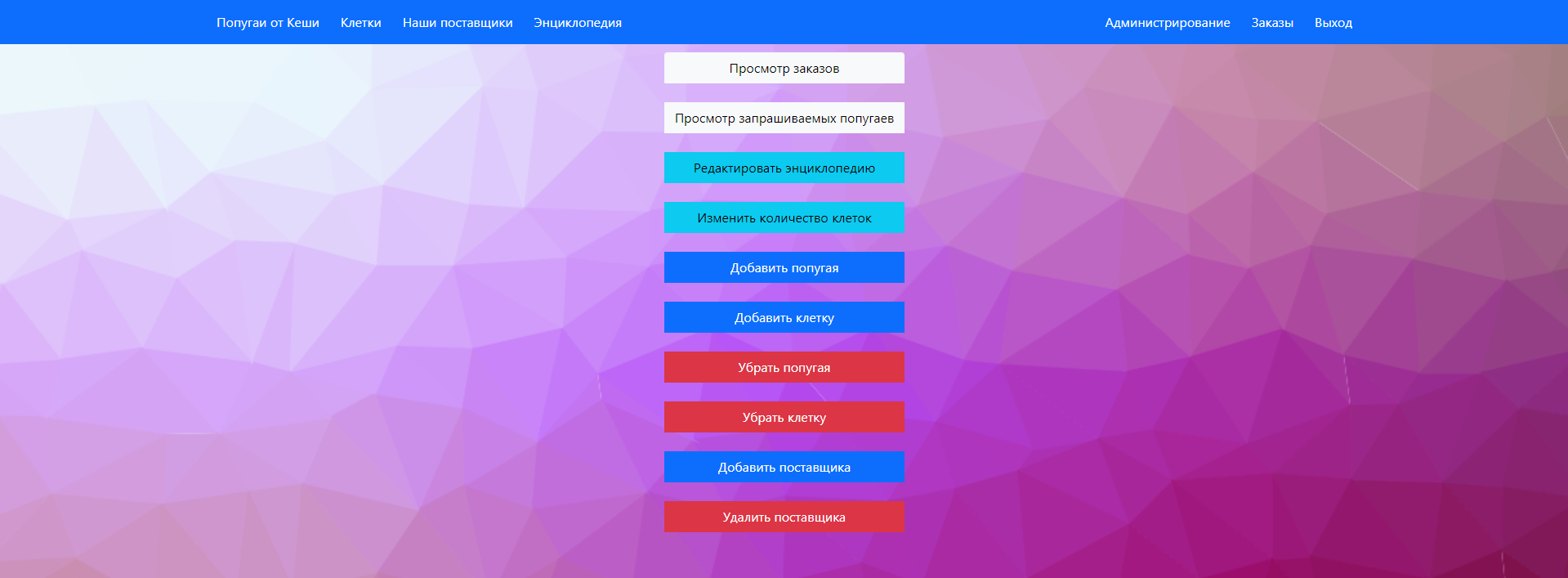
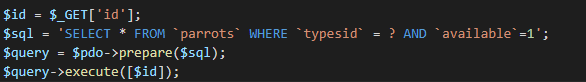


Рисунок 3.15 – Возможности администратора



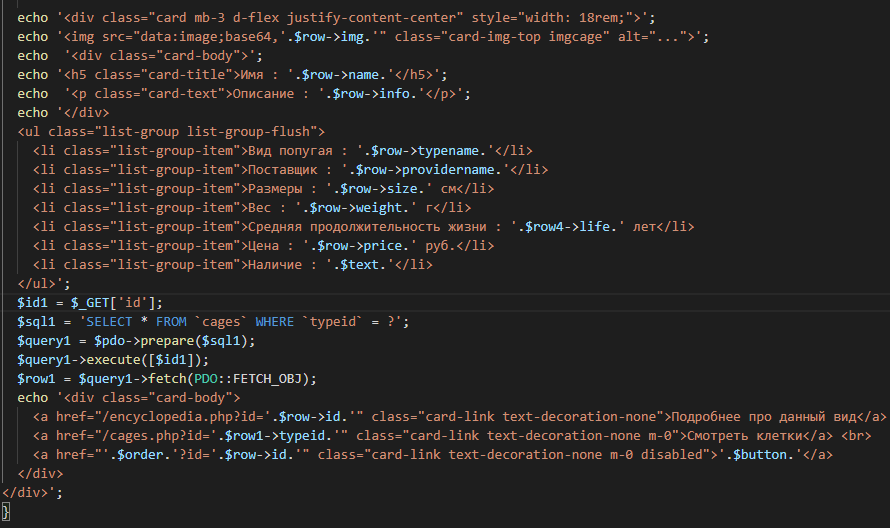


Рисунок 3.16 – Пример запроса получения и вывода попугаев с определенным id