1 слайд{

И так, тема нашего сегодняшнего выступления - Тестирование веб-приложений

}

2 слайд {

Введение.

Сейчас мы расскажем на теории и практике о важности, целях и видах тестирования.

Начнем с важности тестирования.

}

3 слайд{

Важность тестирования.

"Представьте себе: вы зашли на сайт, чтобы купить билет на концерт или заказать еду, и вдруг... сайт выдает ошибку, ничего не загружается, или все работает очень медленно. Скорее всего, вы просто закроете этот сайт и перейдете на другой, где все работает без проблем.

Это обычная ситуация, которую можно избежать, если провести хорошее тестирование веб-приложения. Именно поэтому тестирование — важный этап разработки. Оно помогает нам быть уверенными, что пользователи не столкнутся с неприятными ошибками и смогут легко пользоваться продуктом, и вместе с этим, сами разработчики могут быть уверены что исходный код и важные данные не утекут в руки злоумышленников.

}

4 слайд {

Виды тестирований.

Есть много разных видов тестирования. Сегодня мы расскажем лишь о некоторых, с помощью которых нам удалось протестировать свои веб-приложения. Наше обсуждение тестирование разделено на два блока – функционального и нефункционального тестирования, в функциональное входят юнит тесты и метод трех ящиков, а в нефункциональное тестирование – тестирование безопасности и смол юзабилити тест.

* Функциональное тестирование:

1. Юнит тесты: валидация, положительные и негативные тесты.
2. Три ящика

* Нефункциональное тестирование:

1. Тестирование безопасности: Sql и xss иньекции
2. Тестирование юзабилити

}

5 слайд {

1. *Функциональное тестирование — это проверка, выполняет ли веб-приложение все необходимые функции. Этот вид тестирования распространяется на все элементы веб-приложения с которыми можно взаимодействовать, будь то поля для ввода данных или кнопки. Одним словом, это функции страниц веб-приложения.*

*Функциональное тестирование тесно связано с терминами верификации и валидации.*

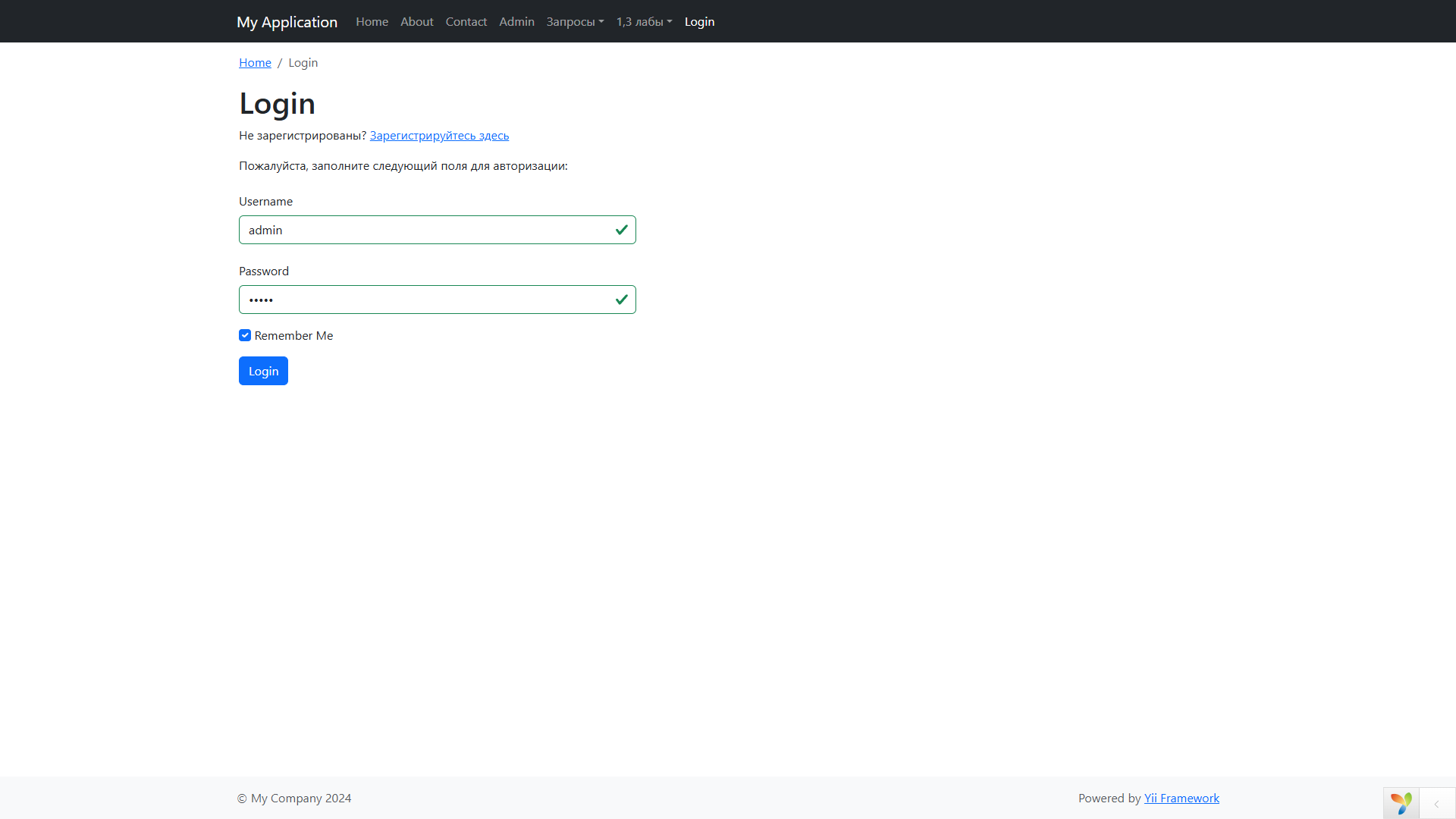
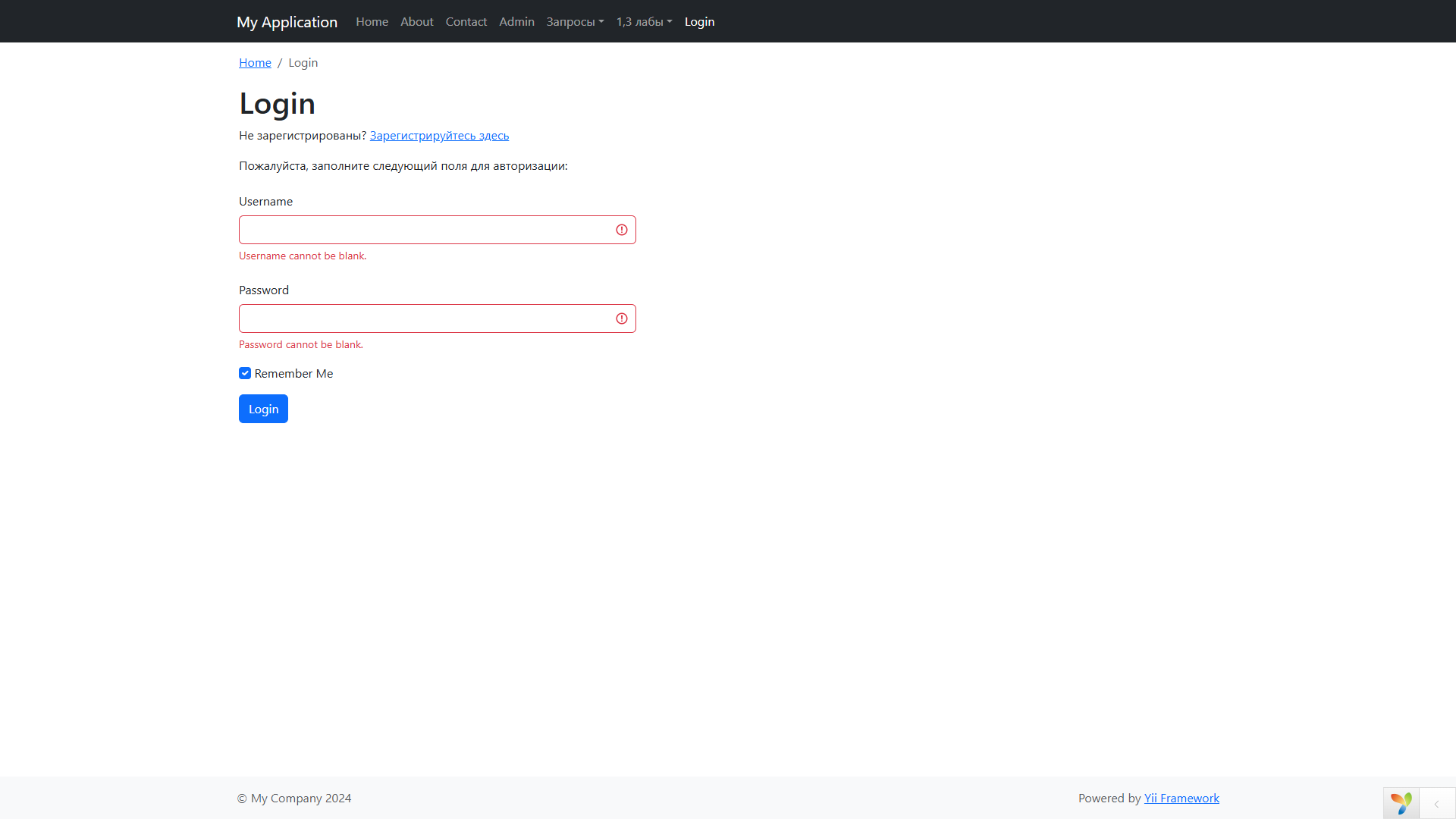
***Верификация****— это проверка соответствия продукта изначально заданным критериям. То есть что в этой форме, так называемые поля username и password действительно отвечают за ввод и проверку данных в бд именно с этими полями.*

***Валидация****— это проверка данных, которые ввёл пользователь. По мере ввода браузер и/или сервер проверяют данные, чтобы определить, соответствуют ли они требуемому формату.*

}

6 слайд {

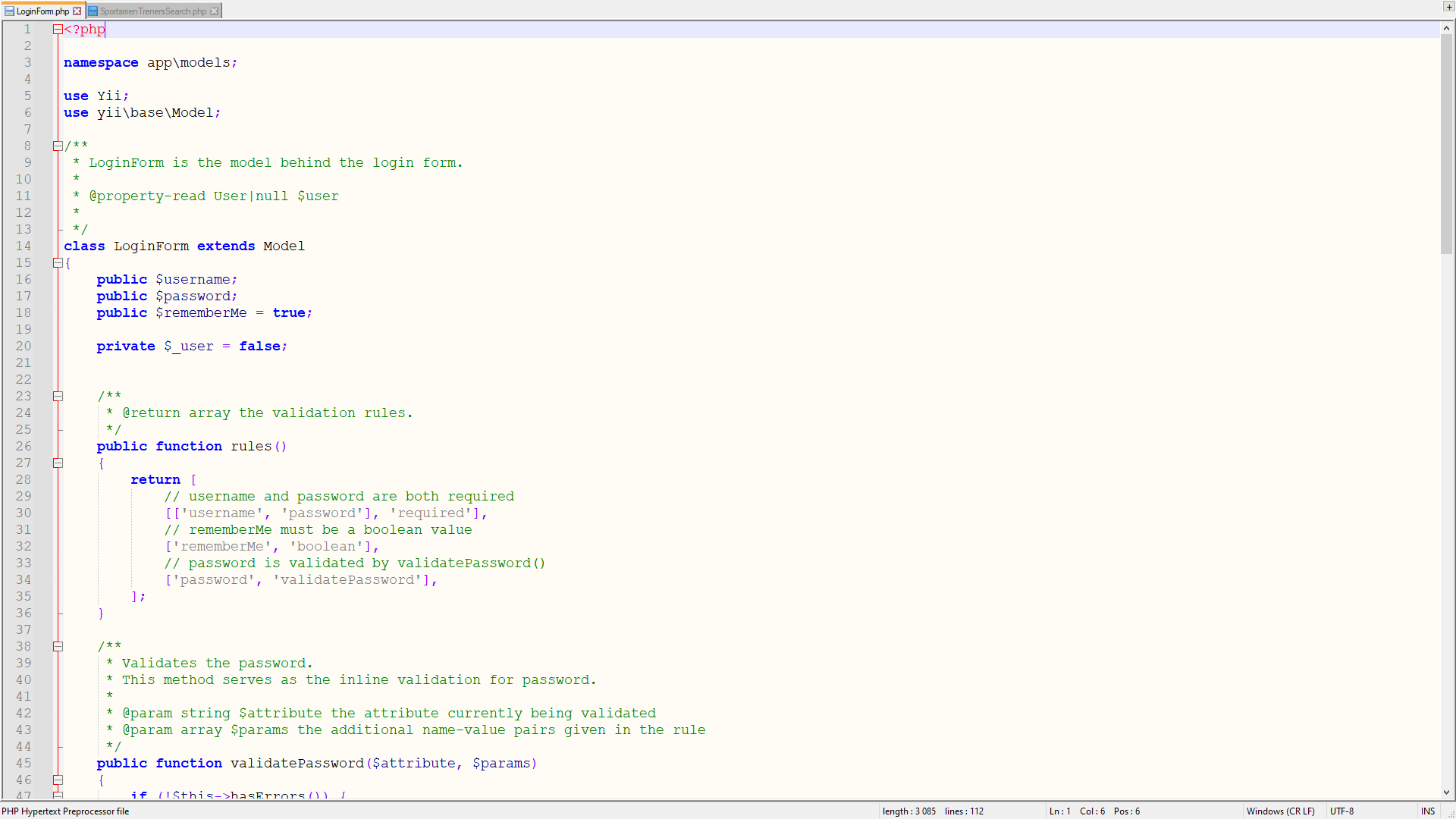
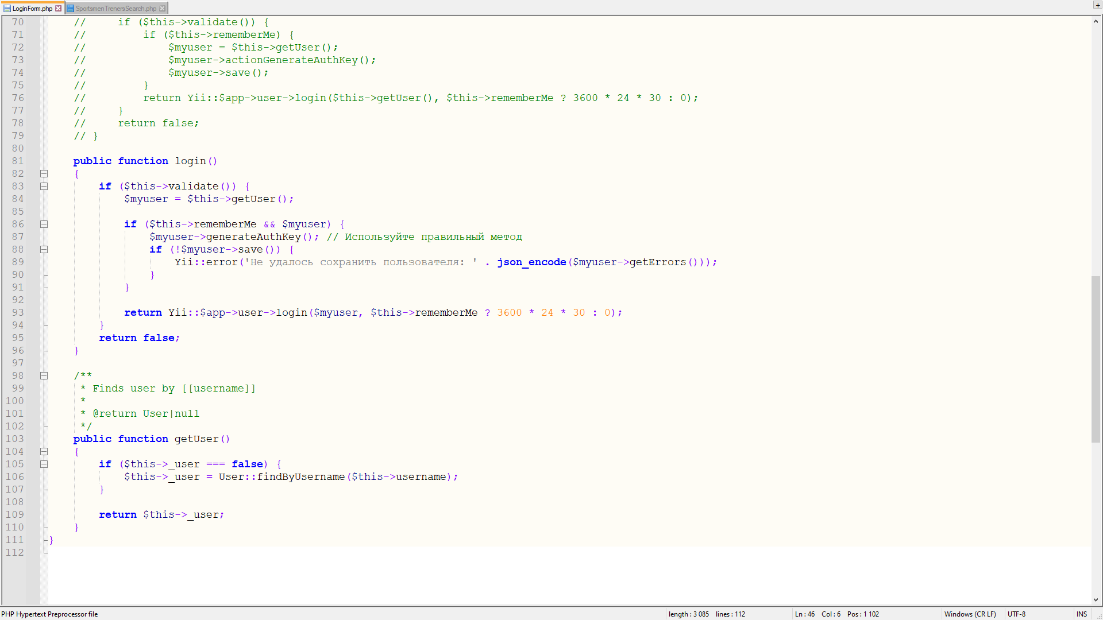
*Например, мы можем протестировать форму регистрации, чтобы убедиться, что все поля работают корректно. В случае если пользователь не заполнил обязательные поля, система должна выдать соответствующее сообщение об ошибке.* Проверка формы регистрации помогает убедиться, что пользователи не смогут зарегистрироваться без выполнения всех необходимых условий, что является критически важным для работы приложения.

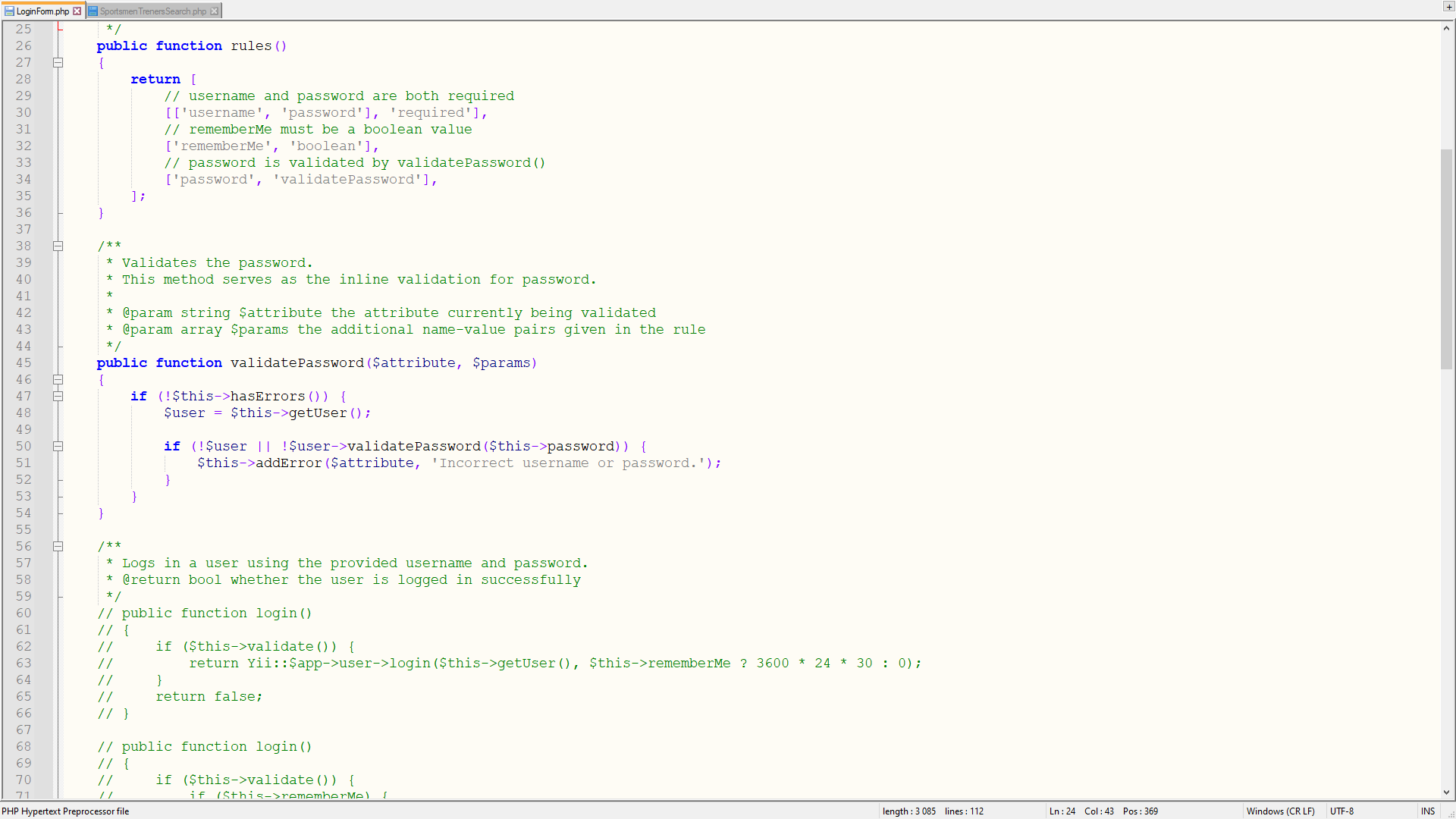


}

7 слайд {

Со стороны кода:



После того как вы заполнили поля уже будет проверка на наличие введенных данных в бд. Если пароль или username будет введен неверно – выдаст соответствующую ошибку.

}

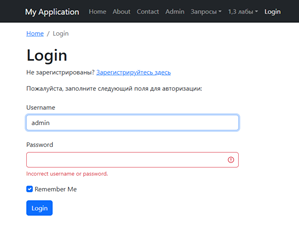
8 слайд {

В этом моменте в кадр входят такие термины как авторизация и аутентификация.

***Авторизация****— на нашем примере это процесс проверки прав пользователя на доступ к информации. Авторизация носит вторичный характер и происходит после успешной аутентификации пользователя в системе.*

}

9 слайд {



}

10 слайд {}

11 слайд {

***Аутентификация****— это процесс (проверки является ли пользователь который вводит данные, тем за кого себя выдает) установления подлинности личности пользователя. В ходе аутентификации пользователь представляет доказательства своей личности, такие как пароль, токен или отпечаток пальца, а система проверяет их. Основная цель аутентификации — убедиться в подлинности пользователя перед предоставлением доступа к системе или приложению.*

}

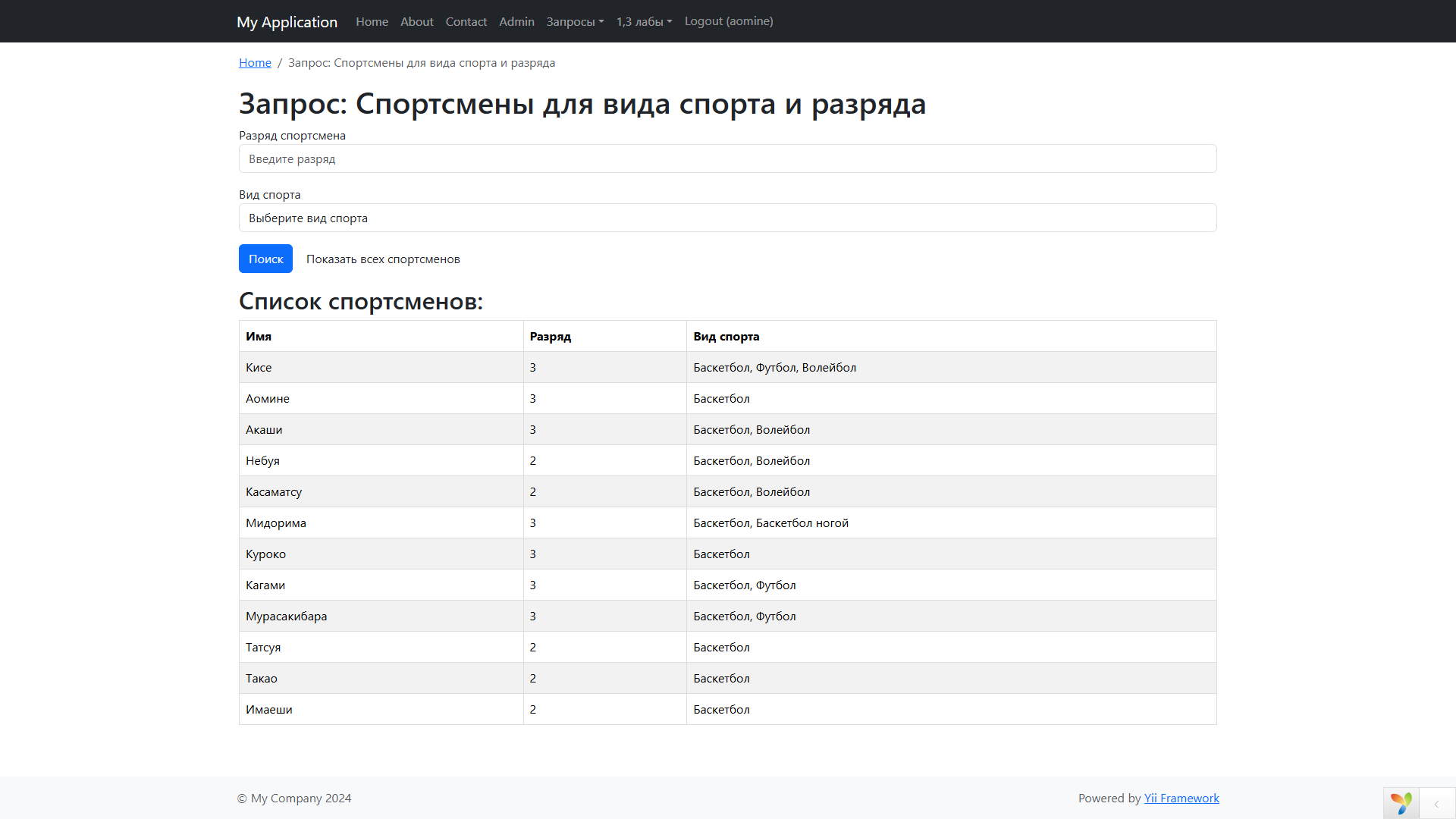
12 слайд{

Следующее ответвление функционального тестирования это позитивные и негативные тестирования полей для ввода данных.

}

13 слайд{

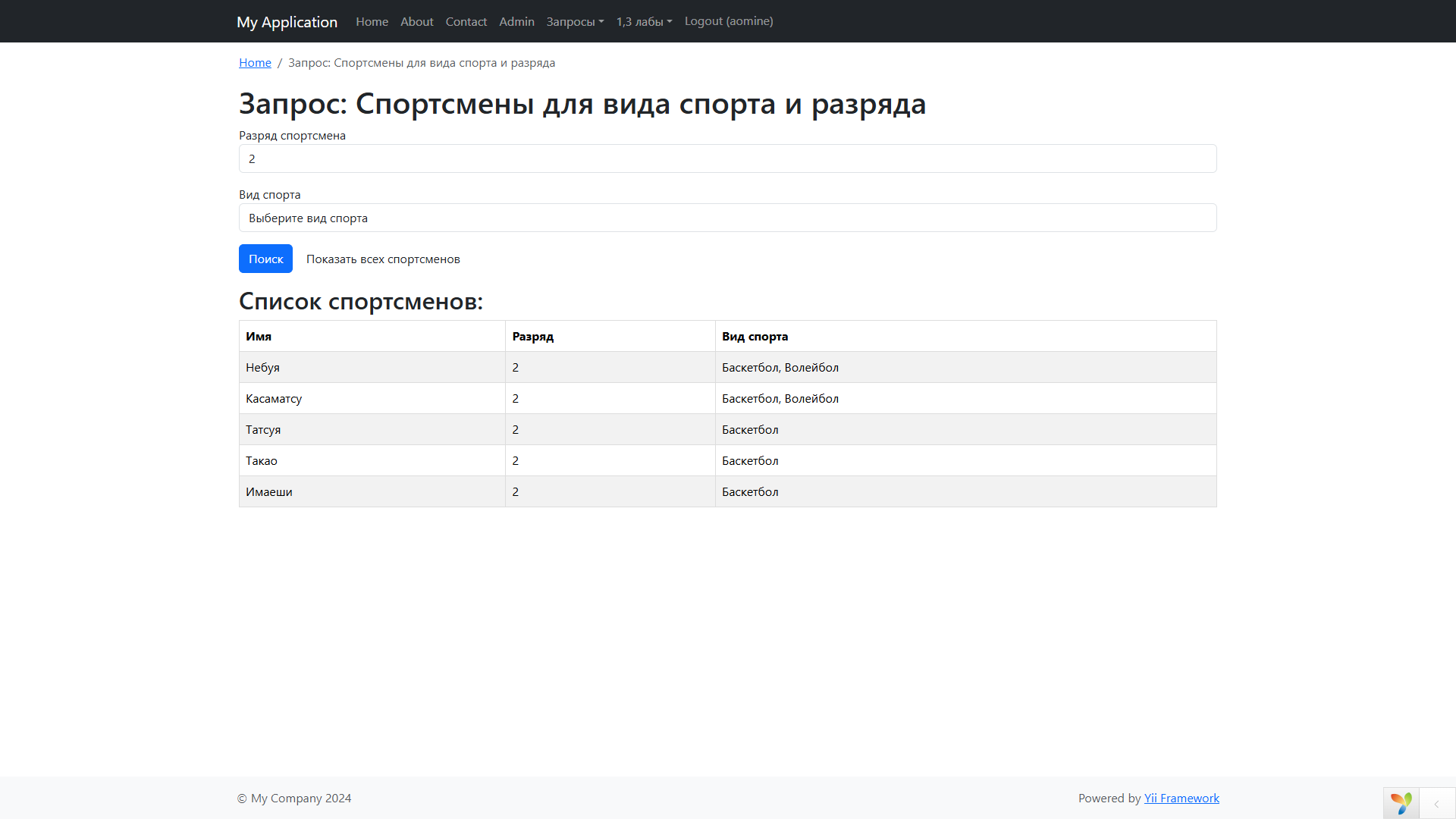
*Сначала всегда необходимо проверить поля на позитивные значения. При стартовой загрузке страницы мы видим список всех имеющихся записей, здесь мы видим что разряды спортсменов у нас 2го и 3го ранга, (на слайде)*

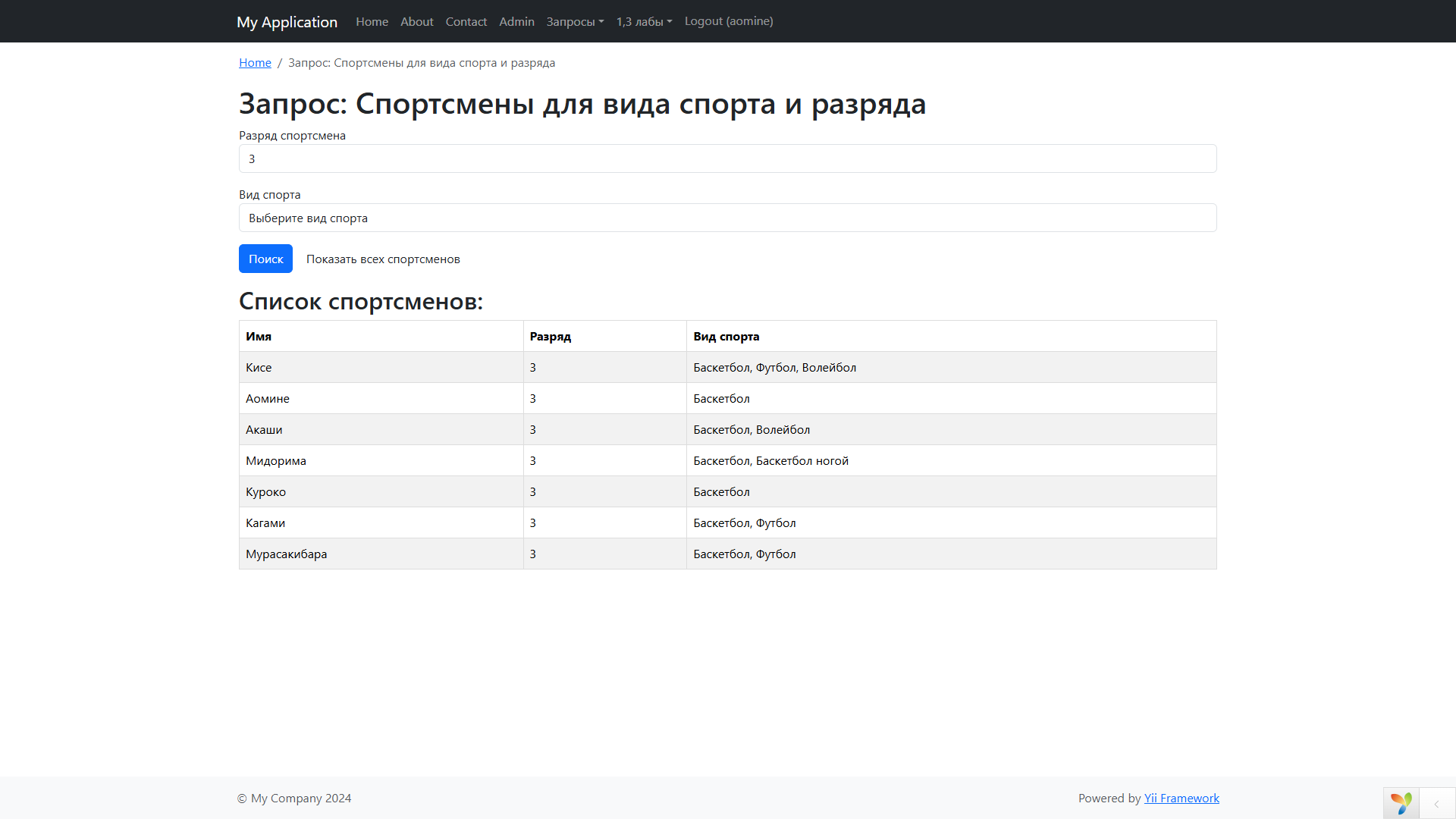


}

14 слайд{

соответственно, проводим тестирование этих значений:



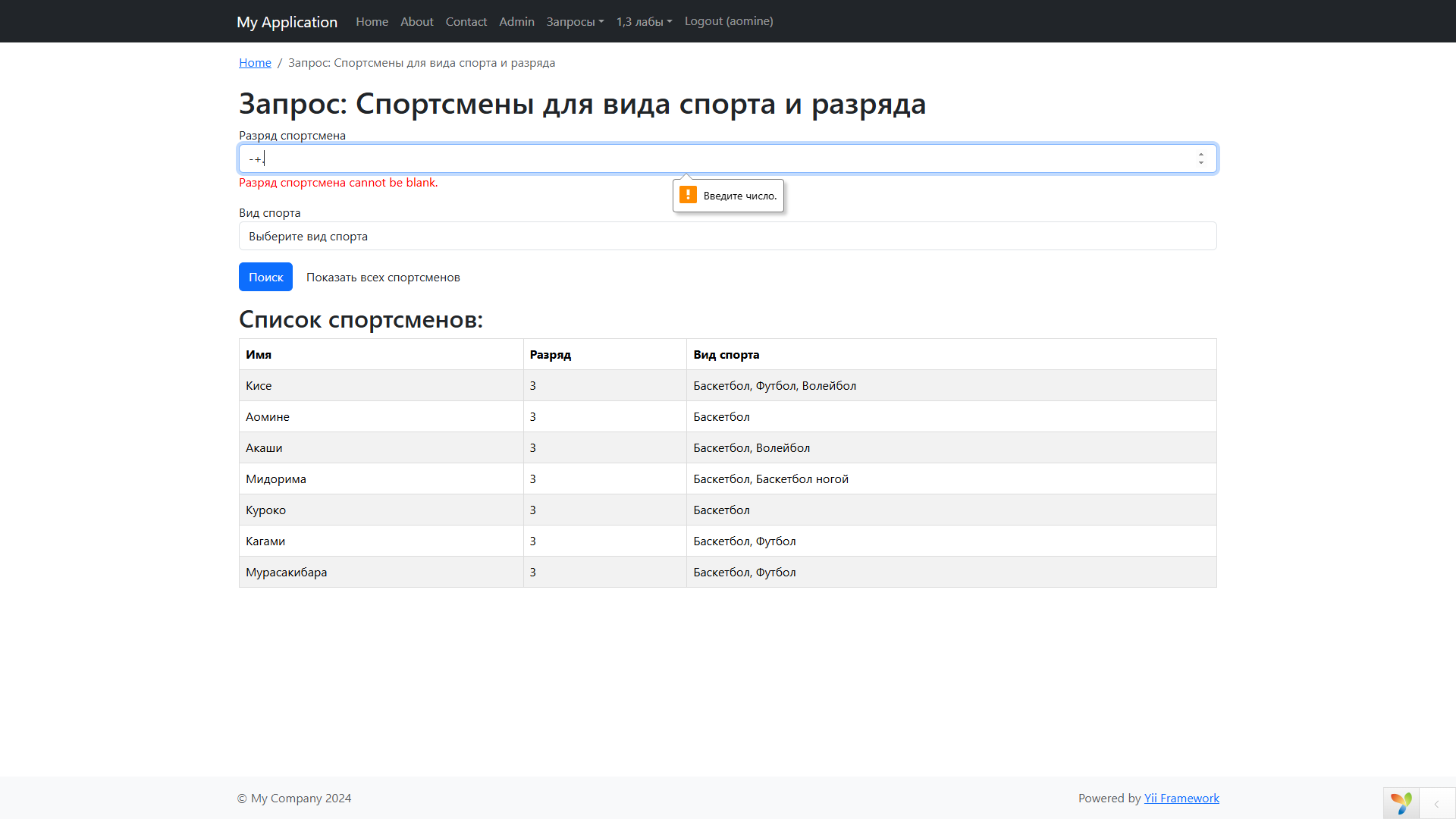


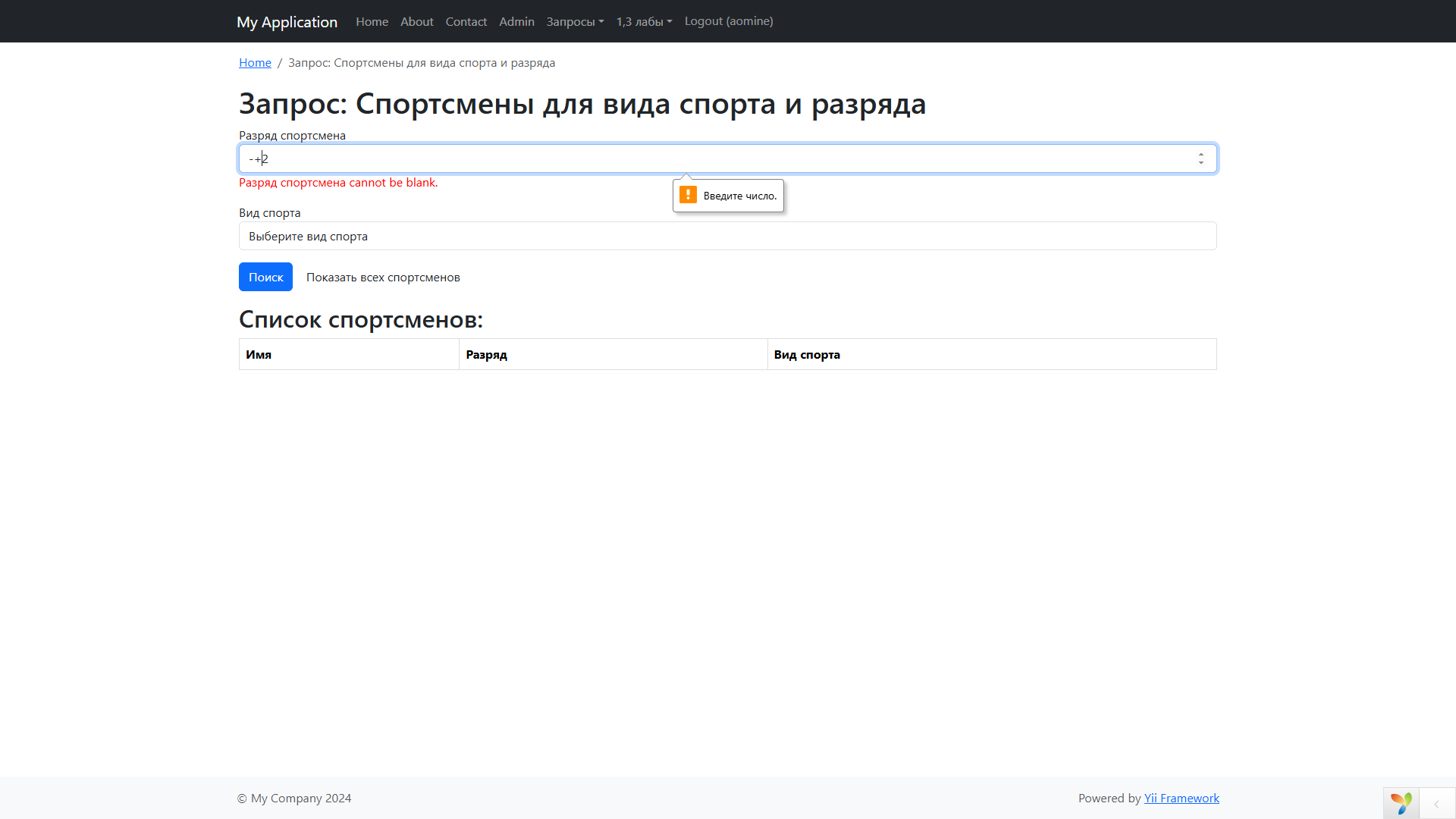
}

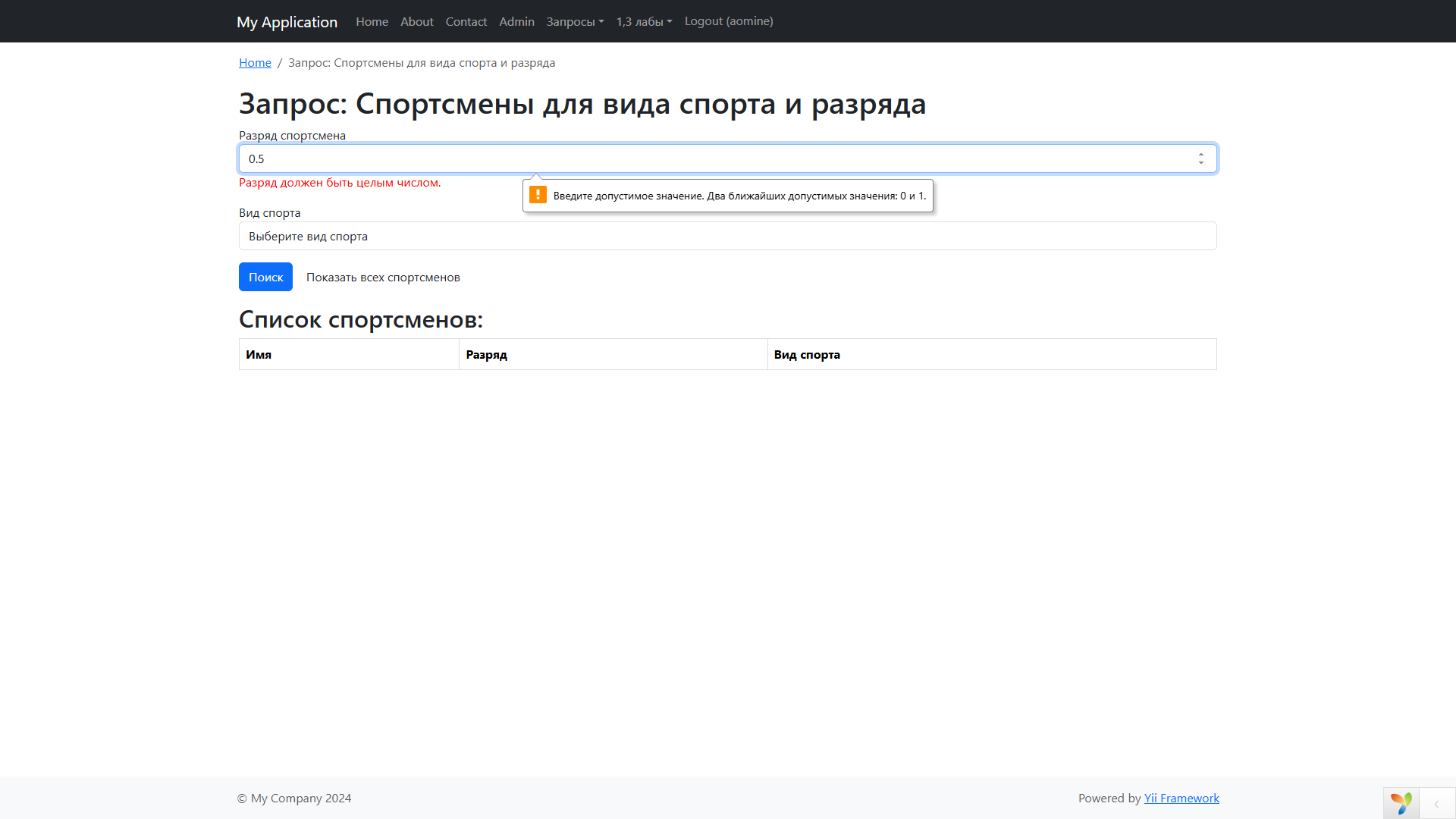
15 слайд{

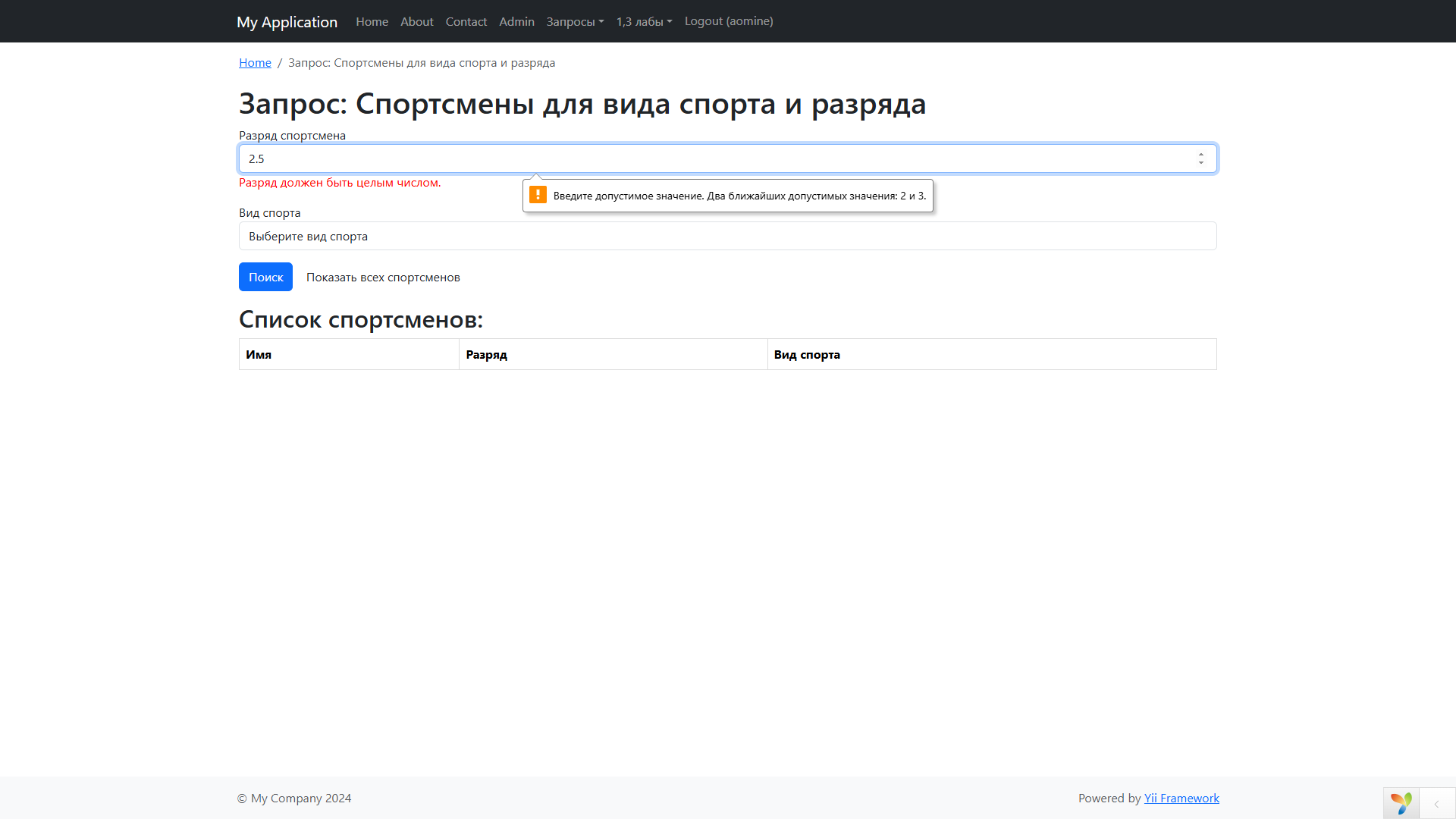
После мы производим тестирование негативных значений:

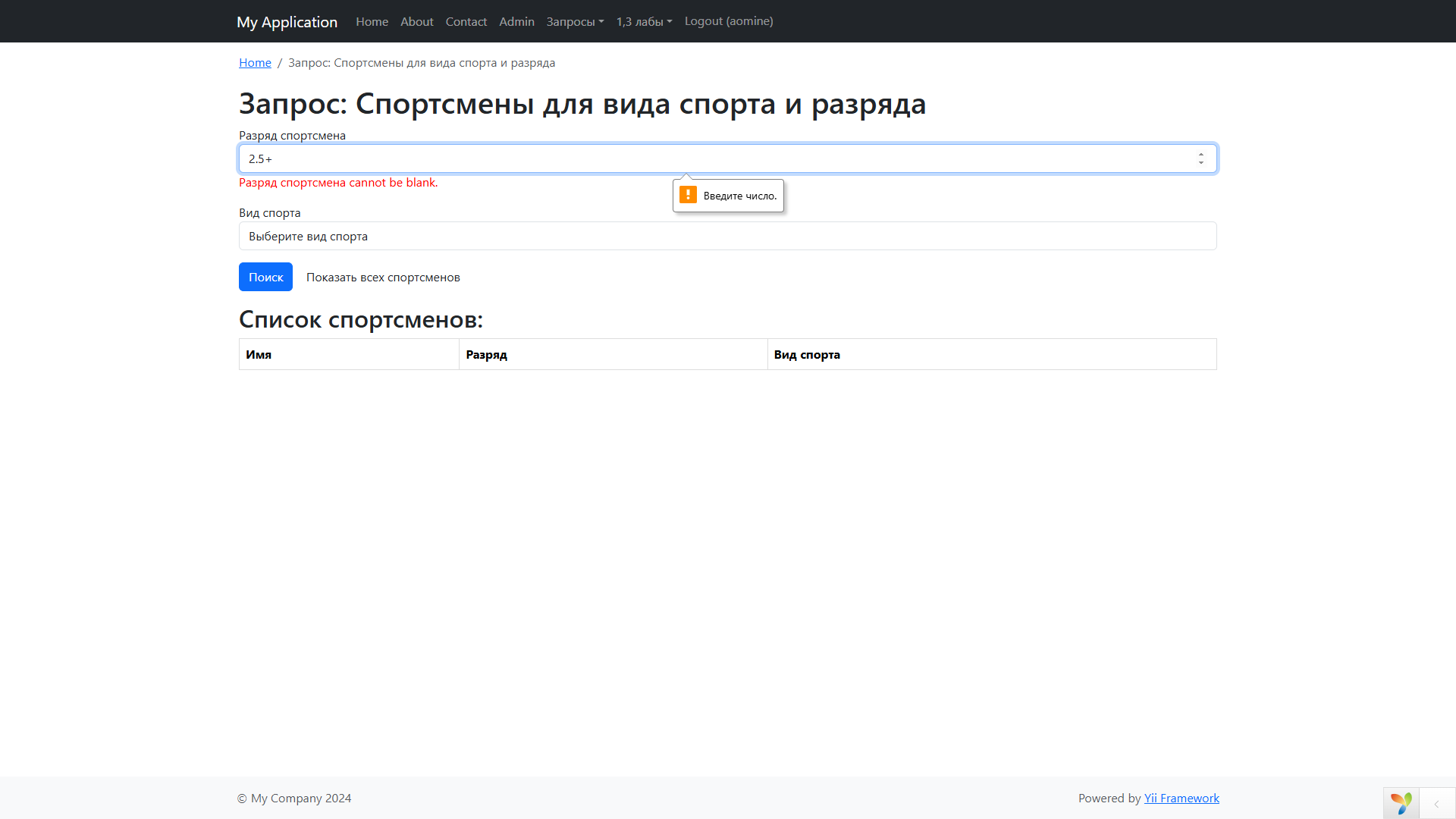
Во время тестирования нам удалось выяснить что поле “Разряд” (означающее общий разряд спортсмена) принимающий только числовые значения может еще поместить в себя символы “-”, “+”, “.”.







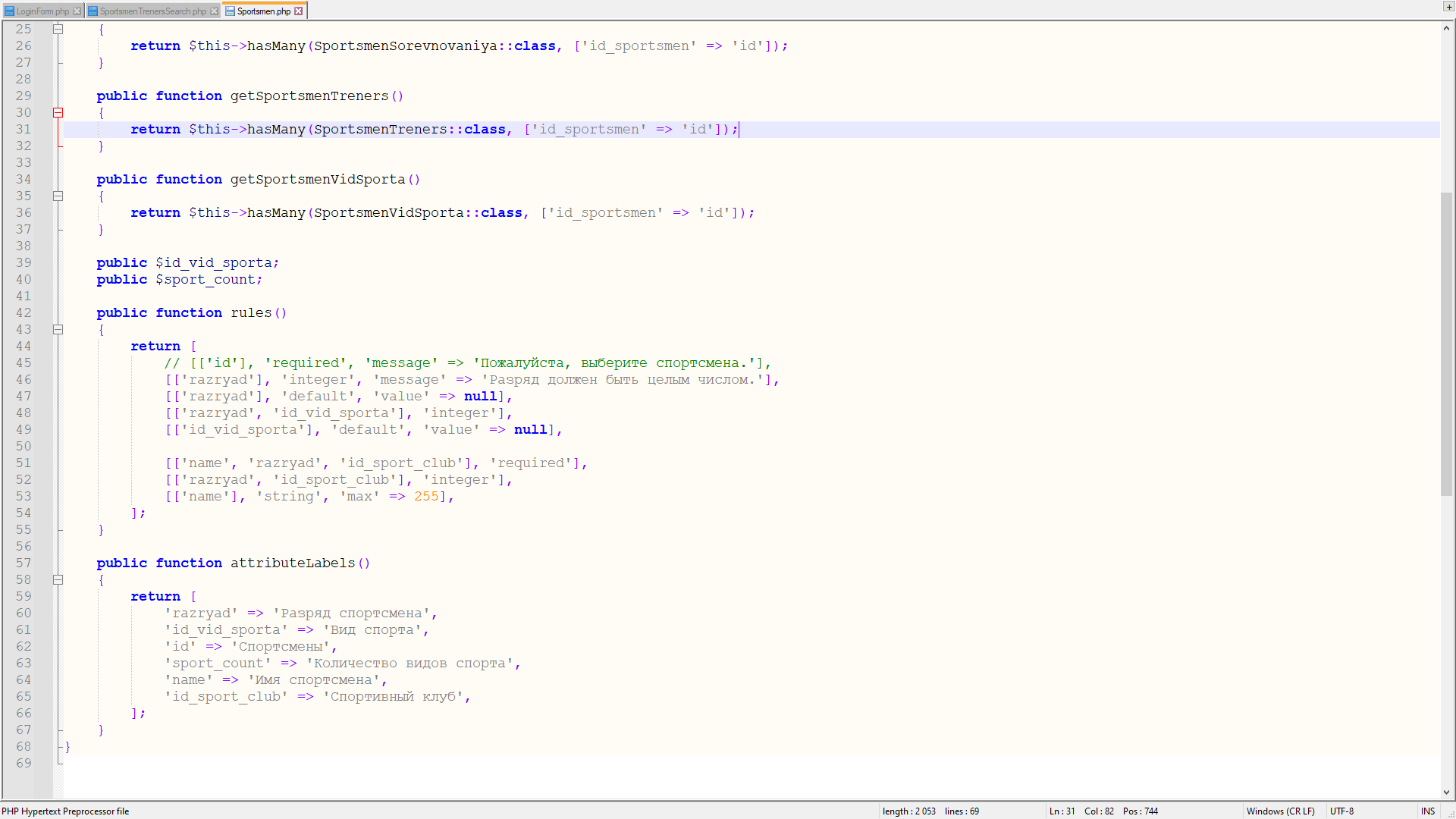




}

16 слайд{

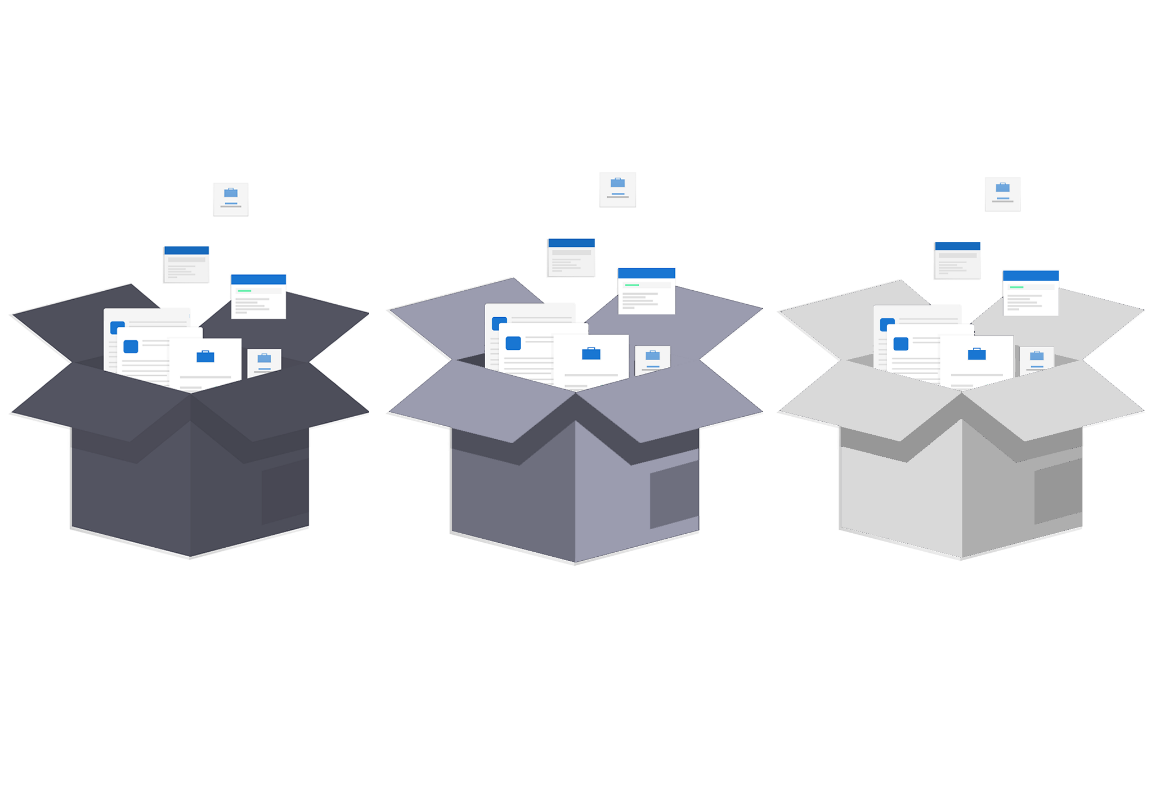
Со стороны кода валидация выглядит следующим образом:



}

17 слайд {

К слову, раз уж мы упомянули функциональное тестирование, есть три метода тестирования веб-приложений это черный ящик, серый ящик и белый ящик.



\*\*обьяснение что это такое\*\*

}

18 слайд {

1. *Нефункциональное тестирование*

*Это тип тестирования веб-приложения для проверки нефункциональных частей приложения, это например – стили, скорость загрузки приложения, адаптивность дизайна, удобная навигация по сайту, ну и конечно безопасность веб-приложения.*

*Нефункциональное тестирование оценивает, как приложение работает, а не что оно делает.*

}

19 слайд {

*Тестирование безопасности.*

*Это тестирование направлено на выявление уязвимостей в веб-приложении, которые могут быть использованы злоумышленниками. Тестирование на безопасность критически важно для защиты данных пользователей и предотвращения утечек информации, что может привести к серьезным последствиям для кампании, включая финансовые потери и ухудшение репутации.*

}

20 слайд {

Пример xss и sql иньекций

Когда вы вводите данные на сайте — например, логин и пароль — приложение передает эту информацию в базу данных для проверки. Но что, если злоумышленник найдет способ "залезть в щель" и отправить такой запрос, который обманет систему и база данных предоставит ему доступ к личным данным пользователей?

**SQL-инъекции** — это именно такой способ взлома, когда злоумышленник использует уязвимость в коде веб-приложения, чтобы вмешаться в запрос к базе данных и изменить его, получив доступ к секретной информации или может даже удалить данные."

"SQL-инъекция — это когда пользовательский ввод на сайте используется неправильно, и вместо обычных данных можно вставить специальный код, который изменяет поведение базы данных.

}

21 слайд {

На пример в моем коде везде были использованы get-запросы, так как нам нужно получить информацию быстро, и мы работаем на локальном сервере, нам боятся нечего). Но в работе он может быть уязвим, так как мы открыто видим весь запрос в URL, при выполнении запроса.

GET-запрос 



POST-запрос же более безопасен, поскольку данные передаются в теле запроса и не видны в URL.



Какой запрос мы бы не строили, нам все равно его не будет видно так в открытую, что уже делает подобный тип передачи данных более безопасным от sql-инъекций.

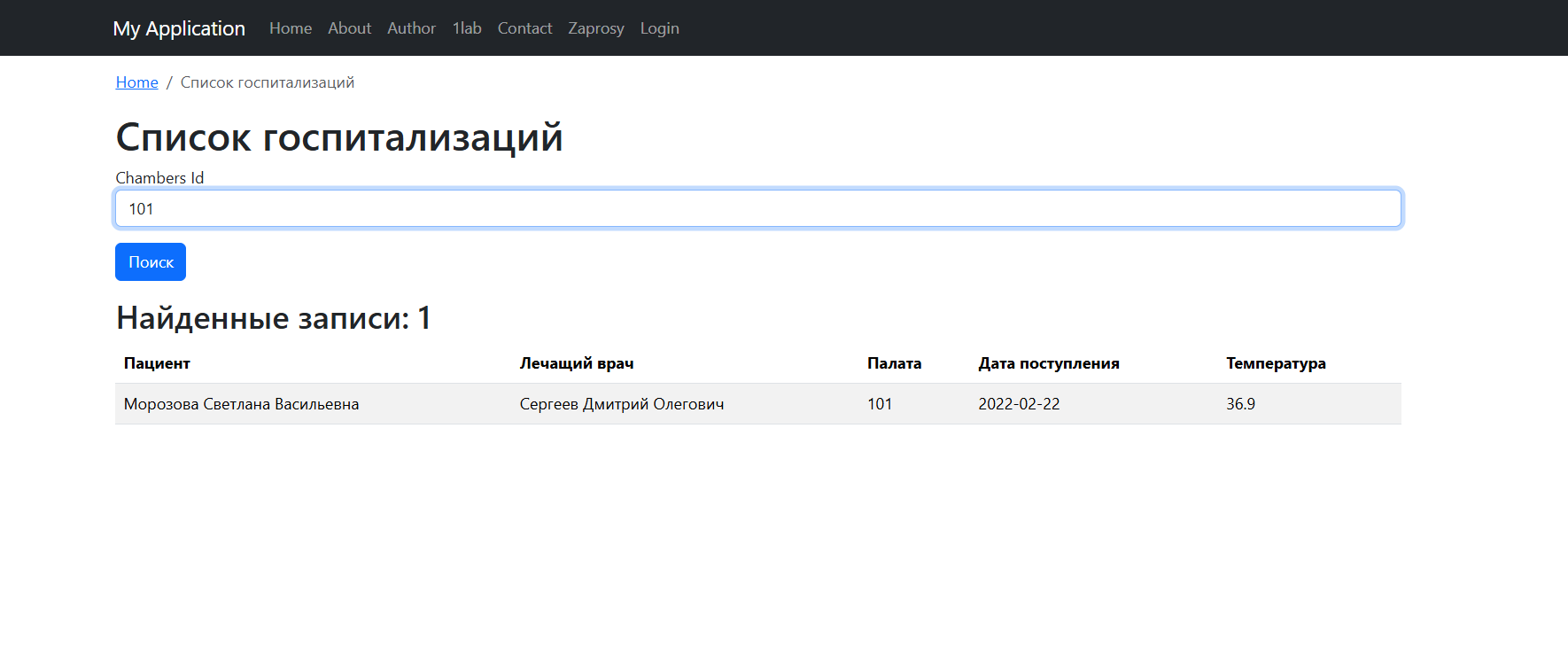
}

22 слайд {

Разберем на примере как выглядит sql-инъекция:

*Допустим у нас есть запрос, в котором исходя из id палаты мы можем узнать кто там находится, его ФИО, ФИО врача, дату поступления и температуру, с которой он поступил. Все данные передаются через GET. (на слайде)*

Например, мы хотим узнать информацию о находящихся в палате 101(её id = 1):



}

23 слайд {

При выполнении данного запроса URL будет выглядить таким образом:





Но давайте попробуем что нибудь сломать. Самый популярный и простой способ получения сторонних данных через sql-инъекцию это SQL запрос OR с очевидным условием



В данном запросе очевидно, что какой бы не был у нас id палаты(chamber\_id), мы все равно получим все нужные данные, т.к. 1=1.

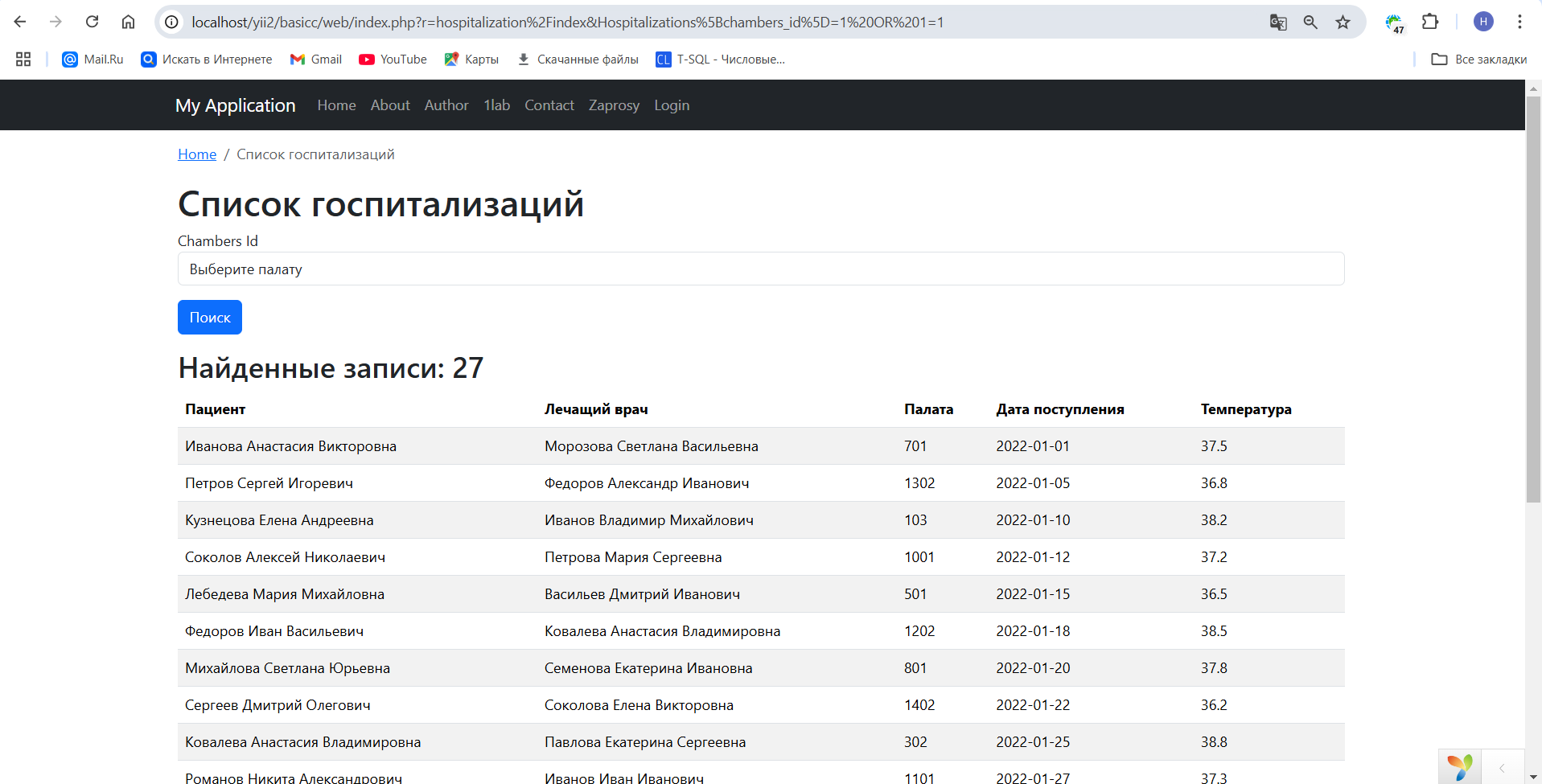
}

24 слайд {

При явном отображении в URL, мы эту ссылку можем менять, и путем различных и не хитрых махинаций получить данные которые нас интересуют(Данные по всем пациентам из всех палат).



В нашей ссылке с поиском пациентов по палатам просто дописав OR 1=1, неважно каким будет id палаты. Мы все равно получим выборку из всех палат.



}

25 слайд {

*Юзабилити тестирование.*

*Следующий вид нефункционального тестирования это юзабилити тесты.*

}

26 слайд {

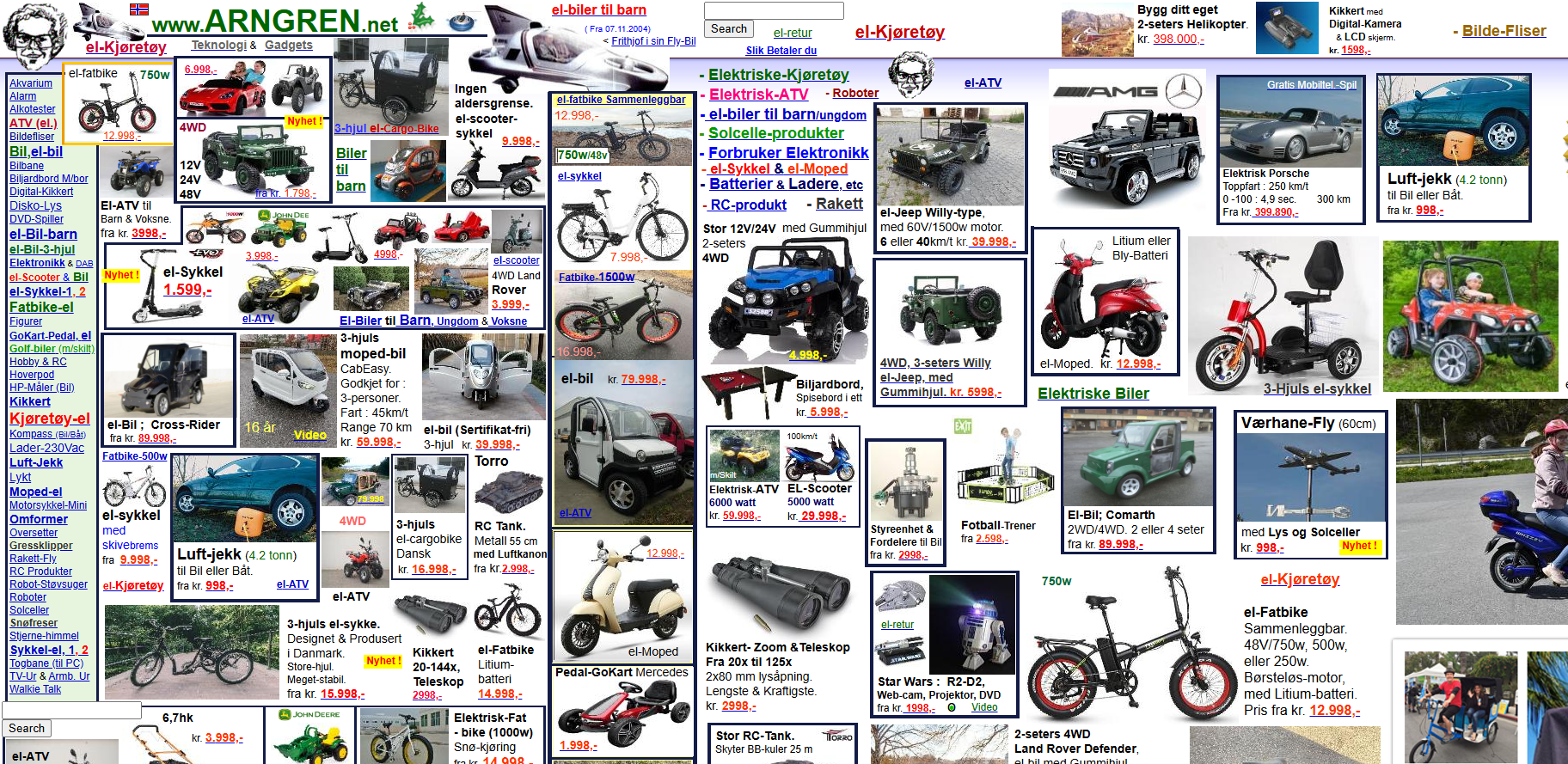
*Данный вид тестирования необходим, чтобы понять, насколько удобно пользователю взаимодействовать с интерфейсом.*

*Если иметь ввиду что YII это продукт, а я пользователь, могу с уверенностью сказать, что с юзабильностью у них все в порядке))*

}

27 слайд {

*В общем главное, что бы ваш сайт не выглядел так:*

**

}

28 слайд {

1. *Заключение*

*Тестирование веб-приложений — это ключевой этап разработки, который помогает обеспечить качество и надежность продукта. Важно внедрять различные виды тестирования на всех этапах разработки, чтобы минимизировать риски и повысить удовлетворенность пользователей. Регулярное тестирование позволяет выявлять и устранять проблемы на ранних стадиях, что в конечном итоге экономит время и ресурсы.*

}

***Дополнительная информация на всякий случай, для понимания:***

# 

# **ВЕРИФИКАЦИЯ и ВАЛИДАЦИЯ**

**Верификация — это проверка соответствия продукта изначально заданным критериям. Пример — регистрация в электронных кошельках, где нужно пройти верификацию для идентификации пользователя в системе.**

**Валидация — это проверка данных, которые ввёл пользователь. По мере ввода браузер и/или сервер проверяют данные, чтобы определить, соответствуют ли они требуемому формату.**

**На примере полей username и password валидация может проверять следующие критерии:**

**Username. Поле обязательно должно быть заполнено.**

**Password. Длина пароля должна быть от 8 до 32 символов, при этом должна быть хотя бы одна буква и одна цифра.**[**1**](https://dev.to/fromwentzitcame/username-and-password-validation-using-regex-2175)

**Верифицированный пользователь — это пользователь, прошедший процедуру верификации, которая подтверждает его личность.**[**1**](https://new-tel.net/blog/kak-kompanii-verifitsiruyut-svoikh-klientov/)[**2**](https://semantica.in/blog/verifikacziya.html)

**Верификация позволяет убедиться в том, что с системой взаимодействует реальный человек, а не бот или мошенник.**[**1**](https://new-tel.net/blog/kak-kompanii-verifitsiruyut-svoikh-klientov/)

**Некоторые способы верификации:**

**Доступ с паролем. Для доступа к системе нужно пройти регистрацию своего аккаунта, придумать логин и закрепить его паролем. При верификации система сличает пароль с логином и совершает тем самым верификацию.**[**1**](https://new-tel.net/blog/kak-kompanii-verifitsiruyut-svoikh-klientov/)

**Доступ через соцсети. Достаточно подтвердить свой аккаунт, например в социальных сетях.**[**1**](https://new-tel.net/blog/kak-kompanii-verifitsiruyut-svoikh-klientov/)

**Доступ через код. Для этого клиент вводит в форму верификации свой номер телефона, на который приходит сообщение с кодом.**[**1**](https://new-tel.net/blog/kak-kompanii-verifitsiruyut-svoikh-klientov/)

**Если аккаунт успешно прошёл верификацию, то в профиле пользователя, справа от никнейма, видна зелёная галочка (в ПК-версии). В мобильной версии и Android-приложении подпись верификации показывается сразу под ником.**

# **ПРИНЦИПЫ ТЕСТИРОВАНИЯ**

1. **Тестирование демонстрирует наличие дефектов а не их отсутствие**.  
   (даже если мы дефекты не нашли, мы не можем сказать, что их вообще нет.)
2. **Исчерпывающее тестирование недостижимо**. ( мы конечно будем использовать техники тест дизайна, которыми мы сможем оптимизировать наше тестирование, но сказать что мы сможем протестировать всё – нельзя!!!)
3. **Ранее тестирование сохраняет время и деньги**. (Исправить ошибку на этапе разработки требований на много дешевле, чем ошибка поступит к разработчику, он напишет код да и это еще попадет к конечному пользователю и будут критические ошибки ( медицинское оборудование, авиация) на начальных этапах лучше подключиться тестировщику, проанализировать требования, дать заключение)
4. **Кластеризация дефектов**. (В 20% ПО храниться 80% дефектов. Знание этого будет нам необходимо в будущем).
5. **Тестирование зависит от контекста**. (ПО которое отвечает за безопасность, тетстируется иначе чем web приложение для электронной комерции).
6. **Заблуждение о отсутствии ошибок**. (Отсылочка к ВАЛИДАЦИИ).
7. **Парадокс пестицида**.( Если мы вводим всегда одни и те же данные, в одно и то же поле, которое хотим протестировать ( допустим слово СОБАКА), то есть вероятно того что наша система изменится, возникнет новый баг, но мы его не сможем найти, так как необходимо вводить слово КОТ. Необходимо изключить что бы выполнялись всегда одни и те же тест-кейсы, вводились одни и те же тестовые данные, даже тесировали одни и те же ЛЮДИ. Это незначит что людей надо увольнять, их надо просто менять)
8. Тестирование черного ящика – У тестироващика нет доступа к коду. И мы можем только наблюдать frontend. Это больше работа начинающих тестировщиков.
9. Тестирование серого ящика - Если мы можем читать код, видеть сторонние приложения. Тестирование API, видеть что уходит на сервер и приходит с сервреа. Backend тестирование.
10. Тестирование белого ящика - Мы видим программный код, знаем архитектуру. Но это уже ближе к автоматизации тестирования. Ручное тестирование обычно на черном и сером ящике.

**Позитивное тестирование** – исследование приложения в ситуации, когда все действия выполняются по инструкции без каких бы то ни было ошибок, отклонений, ввода неверных данных. (Введи валидный правильный логин, валидный правильный пароль, зарегистрировались)

**Негативное тестирование** – направлено на исследование работы приложения в ситуациях, когда с ним выполняются некоторые (некорректные) операции и/или используются данные, потенциально приводящие к ошибкам.(Начинаем проводить проверки приводящие к ошибкам, как результат должно появиться валидационное сообщение)

**Деструктивное тестирование** – одна из форм негативного тестирования с целью нарушить работоспособность приложения и обнаружить точку отказа.

# **ОТЧЕТЫ ПО ТЕСТРИРОВАНИЮ**

Отчет должен отражать основные достижения, проблемы, выводы и рекомендации. Условно у нас прошла итерация и мы провели тестирование. Тут мы должны написать общее **качество продукта (высокое, низкое, среднее**), либо процентовкой (90%, 80%, 60%), у нас такое-то количество критических багов, команда тестирования советует не проводить релиз сейчас

Прописывается информация о команде тестировщиков, которая проводила это тестирование.

**Описание процесса тестирования** – перечень работ за конкретный период. Сколько кейсов было выполнено, сколько багов было найдено. Желательно сравнение в динамике, допусти в прошлую итерацию обнаружено 10 багов в эту итерацию 8. Условно можем сказать, что качество продукта повышается.

**Статистика по новым дефектам** в виде таблицы. Создаются графики, диаграммы, либо таблицы, с информацией о том, сколько дефектов было найдено.

Список новых дефектов.

Статистика по всем дефектам – таблица, содержащая данные обо всех дефектах за время проекта.

Рекомендации – перечень работа за конкретный период. Тут мы можем предложить, то , что можно улучшить.

**Юнит-тесты** (unit test) — это **вид тестирования в программной разработке, который заключается в проверке работоспособности отдельных функциональных модулей, процессов или частей кода приложения**. [2](https://blog.skillfactory.ru/glossary/unit-testirovanie/)

Простыми словами, юнит-тест — это проверка созданной программы частями. [4](https://kokoc.com/blog/unit-testy/)

**Некоторые виды юнит-тестов**:

* **Позитивное тестирование**. Проверяет, правильно ли функция или метод работает при корректных входных данных. [3](https://tproger.ru/articles/yunit-testy-dlya-novichkov--zachem-nuzhny-i-kak-pisat)
* **Негативное тестирование**. Оценивает, как код реагирует на нестандартные ситуации или введение очевидно некорректных данных. [3](https://tproger.ru/articles/yunit-testy-dlya-novichkov--zachem-nuzhny-i-kak-pisat)
* **Граничное тестирование**. Проверка граничных значений входных данных. Например, если функция принимает числа от 1 до 100, граничное тестирование будет проверять значения 0, 1, 100 и 101. [3](https://tproger.ru/articles/yunit-testy-dlya-novichkov--zachem-nuzhny-i-kak-pisat)
* **Тестирование с использованием моков**. Моками называют фейковые базы данных, серверы и другие сложные системы. Это «дублёры», с помощью которых можно безопасно протестировать систему. Например, если метод обращается к базе данных, используют мок для имитации ответа базы данных — без подключения к реальной базе. [3](https://tproger.ru/articles/yunit-testy-dlya-novichkov--zachem-nuzhny-i-kak-pisat)
* **Тестирование состояния**. Проверка, что функция или метод приводит систему или объект в ожидаемое состояние после выполнения. Например, есть метод, который добавляет элемент в список, — после этого размер списка должен увеличиться на один. [3](https://tproger.ru/articles/yunit-testy-dlya-novichkov--zachem-nuzhny-i-kak-pisat)

Юнит-тесты позволяют избежать ошибок или быстро исправить их при обновлении или дополнении ПО новыми компонентами, не тратя время на проверку программного обеспечения целиком. [2](https://blog.skillfactory.ru/glossary/unit-testirovanie/)