**🔎 1. КРИТИЧЕСКИЕ МОМЕНТЫ И УЯЗВИМОСТИ** по backend

**1.1. Права доступа (Roles & Guards)**

* Во всех модулях отсутствует чёткая проверка ролей.  
  **Кто может что делать:** любой аутентифицированный пользователь может создавать, изменять, удалять (users, equipment, news, requests, software, notifications).  
  ➡️ **Рекомендация:**
  + Внедрить Guards по ролям (@Roles('admin'), @Roles('user') и т.д.)
  + Чётко разделить: кто имеет право добавлять, изменять, удалять (например, только админ может удалять или изменять чужие объекты).

**1.2. Валидация данных (class-validator)**

* В большинстве контроллеров используются interface или any для входных данных, но не используются DTO с class-validator.  
  ➡️ **Рекомендация:**
  + Создать для каждого метода отдельный DTO-класс с декораторами (например, @IsString, @IsOptional, @Length, @IsEnum и т.д.).
  + Использовать эти DTO в контроллерах через @Body().

**1.3. Проверка “принадлежности” объектов**

* Нет проверки, что пользователь изменяет или удаляет только свои заявки, профили, оборудование и т.д.  
  ➡️ **Рекомендация:**
  + В сервисах и контроллерах добавить проверку: id пользователя должен совпадать с ownerId объекта (или роль должна быть admin).

**1.4. Безопасность хранения данных**

* Пароли пользователей хэшируются (👍), но пароли оборудования — нет.
* JWT-секрет по умолчанию 'supersecretkey' — **опасно**!  
  ➡️ **Рекомендация:**
  + Для паролей оборудования, если это чувствительная инфа — шифровать или хэшировать.
  + Проект не должен запускаться, если не указан JWT\_SECRET через переменные окружения!

**1.5. Загрузка файлов**

* Нет проверки типов и размера файлов (можно загрузить вредоносный файл).  
  ➡️ **Рекомендация:**
  + Для Multer добавить fileFilter (по MIME-типу) и ограничение по размеру.

**1.6. Логирование**

* В коде есть логирование чувствительных данных (snils, user.id).  
  ➡️ **Рекомендация:**
  + Убрать все персональные данные из логов для production-окружения.

**🛠 2. ФУНКЦИОНАЛЬНЫЕ БАГИ И НЕДОЧЁТЫ**

**2.1. Пагинация**

* В контроллерах, возвращающих списки (news, requests, equipment, software, users), нет пагинации.  
  ➡️ **Рекомендация:**
  + Добавить параметры limit, offset (или page/size) во все такие методы.

**2.2. Ошибка при работе с несуществующими объектами**

* Методы часто не выбрасывают NotFoundException, если объект не найден, а возвращают null.  
  ➡️ **Рекомендация:**
  + Всегда возвращать NotFoundException при отсутствии объекта.

**2.3. Нет ограничения на частоту запросов (rate limit)**

* Любой пользователь может спамить login/register.  
  ➡️ **Рекомендация:**
  + Включить глобальный rate limit (например, через пакет @nestjs/throttler).

**2.4. Нет бизнес-валидации**

* Например, нет проверки, что заявку нельзя удалить/закрыть в определённом статусе.  
  ➡️ **Рекомендация:**
  + Вынести статусы заявок и т.д. в enum, добавить проверки переходов статусов.

**2.5. Нет проверки уникальности**

* Для ключевых полей (snils, inventoryNumber, название ПО и др.) на уровне контроллеров и базы.  
  ➡️ **Рекомендация:**
  + Добавить проверки уникальности на уровне Prisma и базы.

**⚙️ 3. СТИЛЬ И КАЧЕСТВО КОДА**

**3.1. DTO всегда классами**

* Сейчас часто используются interface и any — нужно классы с декораторами.

**3.2. Ошибки должны быть осмысленны**

* Везде использовать HttpException с понятным сообщением.

**3.3. Переиспользование кода**

* Для авторизации и проверки прав доступа вынести декораторы/Guards, чтобы не дублировать код в каждом контроллере.

**💡 4. ЧТО СДЕЛАТЬ КОНКРЕТНО (по папкам/модулям):**

**users:**

* DTO на регистрацию, обновление, создание.
* Guard по ролям (например, только админ может создавать).
* Проверка: может ли пользователь менять только свой профиль.

**auth:**

* DTO вынести в отдельные файлы.
* Убрать дефолтный секрет JWT, обязательна переменная окружения.
* Добавить rate limit.
* Убрать логи персональных данных.

**equipment:**

* DTO на CRUD.
* Проверка прав на действия (например, только ответственный/админ может изменять).
* Ограничить типы и размер загружаемых файлов.
* Проверка уникальности inventoryNumber.

**news:**

* Только админ/модератор может создавать, менять и удалять новости (guard по ролям).
* DTO и валидация для заголовка/текста.
* Пагинация.

**notifications:**

* Только админ может отправлять уведомления.
* DTO на создание уведомления.
* Проверка получателей (существует ли юзер/роль/департамент).

**requests:**

* Только владелец/ответственный/админ может менять и удалять заявку.
* DTO для создания, изменения, комментариев.
* Проверка бизнес-логики перехода статусов.
* Пагинация.

**software:**

* DTO на CRUD.
* Проверка уникальности по названию/версии.
* Только админ может создавать/удалять/менять.
* Пагинация.

**telegram:**

* Проверка, что взаимодействует только авторизованный пользователь (связка telegramId с userId).
* Проверка входящих данных.
* Логирование ошибок.

# 🔎 **Детальный разбор ошибок, багов и недочётов frontend**

## 1. ****Валидация форм и обработка пользовательского ввода****

**Ошибка/Баг:**

* Нет или слабая валидация во всех формах (регистрация, создание заявок, редактирование пользователя/ПО).
* Можно отправить пустые или невалидные значения (например, неправильный SNILS, короткий/пустой телефон, пустой пароль и др.).
* Нет масок и проверки формата для телефонов, SNILS, дат.

**Риски:**

* Мусорные данные в базе, баги при отображении/фильтрации, ошибки на backend, UX-страдания пользователя.

**Рекомендация:**

* Использовать валидацию на клиенте через библиотеки типа **yup** или **zod**.
* Добавлять маски для телефонных номеров и SNILS (например, через react-input-mask).
* Делать все поля с явным required (где нужно), указывать ограничения по длине, формату.
* Подсвечивать ошибочные поля и показывать текст ошибки рядом с инпутом.

## 2. ****Обработка ошибок (от backend и валидации)****

**Ошибка/Баг:**

* В большинстве компонентов ошибки приходят либо в виде alert, либо просто игнорируются (console.log).
* Не выводятся текстовые ошибки под соответствующими полями.
* Нет централизованного уведомления пользователя о критичных ошибках.

**Риски:**

* Пользователь не понимает, что случилось, форма кажется “сломана”.
* Ошибки backend могут не дойти до пользователя.

**Рекомендация:**

* Все ошибки формы отображать под соответствующим полем (например, “Это поле обязательно” или “Такой SNILS уже зарегистрирован”).
* Важно для UX: общий toast/snackbar для глобальных ошибок.
* Использовать try/catch на отправку формы и catch ошибок API.
* Сразу отображать, если backend вернул ошибку — например, конфликт, слишком длинное значение, невалидный формат.

## 3. ****Права доступа и UI на основе роли****

**Ошибка/Баг:**

* Во многих компонентах (карточки, таблицы, кнопки “редактировать/удалить”) нет проверки, кто именно сейчас может это видеть/делать.
* Кнопки могут быть доступны всем пользователям, даже без прав.
* Role берётся из localStorage/context — это легко подделать.

**Риски:**

* Критичные действия могут выполнить любые пользователи (например, удалить чужую заявку).
* Некорректный UI (кнопки доступны всегда).

**Рекомендация:**

* В компонентах всегда проверять права (role) пользователя.
* Показывать только те кнопки/элементы, которые доступны роли (например, admin, manager, user).
* Но **все права обязательно дублировать проверкой на backend**!

## 4. ****Нет loading/skeleton состояний****

**Ошибка/Баг:**

* Нет визуального индикатора загрузки при получении данных (например, список заявок, пользователей, ПО).
* Нет скелетонов — при загрузке UI “прыгает”, пользователь видит “пустой экран”.

**Риски:**

* Плохой UX, ощущение “подвисания” или “неработающего” интерфейса.

**Рекомендация:**

* Добавить Skeleton/Spinner для всех асинхронных операций (fetch пользователей, заявок, ПО, уведомлений).
* Пока данные не пришли — показывать скелетоны или лоадеры.

## 5. ****Нет ограничения/валидации для загрузки файлов****

**Ошибка/Баг:**

* Можно загрузить любой тип и размер файла (например, .exe, .mp4, огромный .zip).
* Нет проверки на тип и размер до отправки.

**Риски:**

* Потенциальная уязвимость (опасные файлы), баги при обработке.
* Перегрузка сервера и долгие загрузки.

**Рекомендация:**

* Ограничивать типы файлов (например, только .pdf, .jpg, .png и т.д.).
* Лимитировать размер (например, до 10 МБ).
* Показывать ошибку, если не проходит проверка (ещё до отправки на сервер).

## 6. ****Нет debounce/pagination для больших списков и поиска****

**Ошибка/Баг:**

* В компонентах типа UserAutocomplete, UserSelect, фильтры и автокомплиты — нет задержки (debounce) на отправку запросов.
* Нет пагинации — если пользователей/ПО/заявок много, UI тормозит или даже падает.

**Риски:**

* Перегрузка сервера, лаги, баги с “зависанием” UI.

**Рекомендация:**

* Везде где поиск — делать debounce (300-500 мс).
* В таблицах — внедрить пагинацию или “ленивую подгрузку”.
* Ограничивать количество элементов на странице (например, до 50).

## 7. ****Длинные строки “ломают” UI, нет placeholder-ов****

**Ошибка/Баг:**

* ФИО, название ПО, описания заявок, длинные сообщения — могут растягивать карточки/таблицы, разрывать верстку.
* Если данных нет, поле просто пустое (непонятно, ошибка это или просто нет данных).

**Риски:**

* Сломанная верстка, плохой UX.

**Рекомендация:**

* Для всех длинных строк — использовать truncate/ellipsis.
* Для пустых значений — показывать заглушку (“—”, “нет данных”).

## 8. ****Нет адаптивности/доступности (Accessibility)****

**Ошибка/Баг:**

* Кнопки, формы, инпуты не имеют aria-label, alt, описаний.
* Нет поддержки клавиатурной навигации, нет выделения фокуса.

**Риски:**

* Неудобно для пользователей с особенностями, не проходят accessibility-тесты.

**Рекомендация:**

* Добавлять aria-label, role, alt-описания.
* Следить за поддержкой Tab и фокуса на всех интерактивных элементах.
* Проверить основные сценарии навигации без мышки.

## 9. ****SVG-иконки не оптимизированы****

**Ошибка/Баг:**

* SVG-иконки не всегда через ReactComponent, не оптимизированы по размеру, нет поддержки currentColor (цвета через css).
* Нет alt для иконок, если они используются как <img>.

**Риски:**

* Большой размер, неуправляемость, проблемы с цветами и доступностью.

**Рекомендация:**

* Использовать SVGR для импорта SVG как ReactComponent.
* Оптимизировать через SVGO.
* currentColor — для управления цветом через стили.

## 10. ****Типизация и совместимость с backend****

**Ошибка/Баг:**

* В types/\*.ts есть расхождения с backend (enum’ы, структура объектов).
* Много any, нет shared-типов (например, через openapi-typescript или tRPC).

**Риски:**

* Баги при отправке/получении данных, падения UI из-за несовпадения структур.

**Рекомендация:**

* По возможности генерировать типы из backend (OpenAPI/tRPC).
* Все типы держать в актуальном состоянии при изменении API.

# 📋 **Сводный чек-лист для исправления**

1. Фронтовая валидация (yup/zod) во все формы.
2. Подсветка и пояснения ошибок под каждым полем.
3. Loading/Skeleton на время ожидания данных.
4. Дебаунс в автокомплитах/поисках.
5. Пагинация в таблицах/списках.
6. Ограничение/валидация файлов на загрузку.
7. Только актуальные права (role) — и на UI, и на backend.
8. Truncate длинных строк, заглушки для пустых.
9. SVG — через ReactComponent, оптимизация, currentColor.
10. Доступность: aria-label, alt, Tab-навигация.
11. Согласованность типов frontend ↔ backend.