Тестовое задание по направлению «Backend-разработка. Golang»

Линтер для проверки лог-записей

Описание задания:

Необходимо разработать линтер для Go, совместимый с golangci-lint, который будет анализировать лог-записи в коде и проверять их соответствие установленным правилам.

Требования к линтеру

Линтер должен проверять следующие правила для лог-записей:

1. Лог-сообщения должны начинаться со строчной буквы

```
// ❌ Неправильно
log.Info("Starting server on port 8080")
slog.Error("Failed to connect to database")

// ✅ Правильно
log.Info("starting server on port 8080")
slog.Error("failed to connect to database")
```

2. Лог-сообщения должны быть только на английском языке

```
// ❌ Неправильно
log.Info("запуск сервера")
log.Error("ошибка подключения к базе данных")

// ✅ Правильно
log.Info("starting server")
log.Error("failed to connect to database")
```

3. Лог-сообщения не должны содержать спецсимволы или эмодзи

```
// ❌ Неправильно
log.Info("server started! 🚀")
log.Error("connection failed!!!")
log.Warn("warning: something went wrong...")

// ✅ Правильно
log.Info("server started")
log.Error("connection failed")
log.Warn("something went wrong")
```

4. Лог-сообщения не должны содержать потенциально чувствительные данные

```
// ❌ Неправильно
log.Info("user password: " + password)
log.Debug("api_key=" + apiKey)
log.Info("token: " + token)

// ✅ Правильно
log.Info("user authenticated successfully")
log.Debug("api request completed")
log.Info("token validated")
```

Технические требования:

1. Язык разработки: Go 1.22+
2. Совместимость: Линтер должен работать как плагин для golangci-lint

3. Поддерживаемые логгеры:
    - log/slog
    - go.uber.org/zap

Этапы выполнения:

Этап 1: Базовая структура

- Создать структуру проекта
- Настроить go.mod с необходимыми зависимостями
- Реализовать базовый анализатор с использованием golang.org/x/tools/go/analysis

Этап 2: Реализация правил

- Реализовать проверку на строчную букву в начале сообщения
- Реализовать проверку на английский язык (можно использовать unicode ranges)
- Реализовать проверку на спецсимволы и эмодзи
- Реализовать проверку на чувствительные данные (по ключевым словам)

Этап 3: Тестирование

- Написать unit-тесты для каждого правила
- Создать тестовые файлы в testdata/
- Проверить работу на реальных проектах

Этап 4: Интеграция с golangci-lint

- Создать плагин для golangci-lint
- Написать документацию по установке и использованию

Полезные ресурсы:

- Документация по созданию анализаторов
- Руководство по созданию плагинов для golangci-lint
- Пример простого линтера

- Исходный код staticcheck — для вдохновения

Бонусные задания:

1. **Конфигурация**: Добавить возможность настройки правил через конфигурационный файл
2. **Авто-исправление**: Реализовать SuggestedFixes для автоматического исправления ошибок
3. **Кастомные паттерны**: Добавить возможность указывать свои паттерны для проверки чувствительных данных
4. **CI/CD**: Подготовить CI Gitlab/GitHub для автоматической сборки и тестирования

Формат сдачи:

1. Ссылка на публичный репозиторий (GitLab/GitHub)
2. README с инструкцией по сборке и запуску
3. Примеры использования