# Техническая документация по поддержанию жизненного цикла ПО "SWP"

## 1. Общие сведения

### 1.1. Введение

Настоящий документ содержит описание процессов, обеспечивающих поддержание жизненного цикла программного обеспечения "SWP" (далее — Система), включая устранение неисправностей, выявленных в ходе эксплуатации, совершенствование программного обеспечения, а также информацию о персонале, необходимом для обеспечения такой поддержки.

### 1.2. Область применения

Данная документация применяется ко всем компонентам Системы "SWP", включая серверную часть (backend), клиентскую часть (frontend), базу данных и интеграционные компоненты.

## 2. Жизненный цикл программного обеспечения

### 2.1. Этапы жизненного цикла

Жизненный цикл ПО "SWP" включает следующие этапы:

1. **Планирование разработки** — определение требований, функциональных возможностей и архитектуры системы.
2. **Разработка** — программирование компонентов, создание пользовательского интерфейса, интеграция с внешними системами.
3. **Тестирование** — проверка функционала на соответствие требованиям, выявление и устранение ошибок.
4. **Внедрение** — установка, настройка и запуск системы в эксплуатацию.
5. **Эксплуатация** — использование системы конечными пользователями.
6. **Сопровождение** — мониторинг работы, устранение возникающих проблем, обновление компонентов.
7. **Модернизация** — добавление новых функций, улучшение существующего функционала.
8. **Вывод из эксплуатации** — завершение использования системы, архивация данных.

### 2.2. Модель жизненного цикла

Для ПО "SWP" используется итеративная модель жизненного цикла с элементами гибкой (Agile) методологии, что позволяет оперативно реагировать на изменения требований и внешней среды. Длительность одной итерации (спринта) составляет 2 недели.

## 3. Процессы поддержания жизненного цикла

### 3.1. Устранение неисправностей

#### 3.1.1. Классификация неисправностей

Неисправности в системе классифицируются по следующим уровням критичности:

1. **Критический уровень** — неисправности, делающие невозможным использование системы, приводящие к потере данных или нарушению безопасности.
2. **Высокий уровень** — неисправности, существенно затрудняющие использование основных функций системы.
3. **Средний уровень** — неисправности, затрагивающие неосновные функции или имеющие обходные пути решения.
4. **Низкий уровень** — косметические дефекты, не влияющие на функциональность.

#### 3.1.2. Процесс выявления неисправностей

Неисправности могут быть выявлены следующими способами: - Автоматический мониторинг системы - Обращения пользователей через систему технической поддержки - Плановое тестирование - Аудит безопасности - Анализ производительности

#### 3.1.3. Процесс устранения неисправностей

1. **Регистрация неисправности**

* Фиксация в системе отслеживания ошибок
* Присвоение идентификатора и уровня критичности

1. **Анализ неисправности**

* Диагностика проблемы
* Определение причины и оценка влияния

1. **Устранение неисправности**

* Разработка и тестирование решения
* Проверка отсутствия регрессий

1. **Внедрение исправления**

* Для критических ошибок — оперативное обновление
* Для некритических — включение в плановое обновление

1. **Верификация исправления**

* Подтверждение устранения проблемы
* Обновление статуса в системе отслеживания

#### 3.1.4. Временные рамки устранения неисправностей

В зависимости от уровня критичности, установлены следующие временные рамки: - Критический уровень — начало работ незамедлительно, устранение в течение 4 часов - Высокий уровень — начало работ в течение 8 часов, устранение в течение 1-2 рабочих дней - Средний уровень — начало работ в течение 2 рабочих дней, устранение в течение 5 рабочих дней - Низкий уровень — включение в план следующих обновлений

### 3.2. Совершенствование программного обеспечения

#### 3.2.1. Планирование совершенствования

Планирование осуществляется на основе: - Обратной связи от пользователей - Анализа рынка и конкурентной среды - Новых бизнес-требований - Технологических возможностей - Требований безопасности и соответствия нормативным актам

#### 3.2.2. Процесс разработки новых функций

1. **Формирование требований**

* Сбор и анализ потребностей пользователей
* Определение функциональных и нефункциональных требований
* Приоритизация требований

1. **Проектирование**

* Разработка технического задания
* Создание прототипов интерфейса
* Проектирование архитектуры изменений

1. **Разработка**

* Программирование
* Создание unit-тестов
* Документирование кода

1. **Тестирование**

* Функциональное тестирование
* Интеграционное тестирование
* Нагрузочное тестирование
* Тестирование безопасности

1. **Внедрение**

* Сборка релиза
* Тестирование релиза
* Развертывание в продуктивной среде
* Миграция данных (при необходимости)

1. **Поддержка после внедрения**

* Мониторинг работы
* Сбор обратной связи
* Оперативное устранение возникающих проблем

#### 3.2.3. Управление версиями

Система версионирования основана на принципе Semantic Versioning: - Мажорная версия (X.0.0) — несовместимые изменения API - Минорная версия (X.Y.0) — добавление функциональности с сохранением обратной совместимости - Патч-версия (X.Y.Z) — исправление ошибок, не затрагивающее API

Все изменения фиксируются в документе "История изменений" (Changelog).

#### 3.2.4. Обновление системы

1. **Подготовка к обновлению**

* Создание резервных копий данных
* Уведомление пользователей о планируемом обновлении
* Подготовка инструкций по миграции

1. **Процесс обновления**

* Развертывание новой версии в среде тестирования
* Тестирование основных функций
* Развертывание в продуктивной среде
* Выполнение скриптов миграции данных

1. **Действия после обновления**

* Проверка работоспособности системы
* Мониторинг производительности
* Сбор обратной связи от первых пользователей

### 3.3. Техническая поддержка

#### 3.3.1. Уровни технической поддержки

1. **Первая линия поддержки**

* Прием и регистрация обращений пользователей
* Консультации по базовым вопросам работы системы
* Эскалация сложных вопросов на вторую линию

1. **Вторая линия поддержки**

* Решение сложных проблем пользователей
* Диагностика технических неисправностей
* Эскалация на третью линию при необходимости

1. **Третья линия поддержки**

* Решение критических проблем
* Исправление ошибок в коде
* Восстановление системы после сбоев

#### 3.3.2. Каналы поддержки

* Email-поддержка
* Система тикетов (Service Desk)
* Телефонная поддержка
* Документация и база знаний

#### 3.3.3. Режим работы технической поддержки

* Первая линия: 5x8 (рабочие дни с 9:00 до 18:00)
* Вторая линия: 5x8 (рабочие дни с 9:00 до 18:00)
* Третья линия (для критических инцидентов): 24x7

### 3.4. Управление конфигурациями

#### 3.4.1. Система контроля версий

Для хранения исходного кода используется Git. Репозитории организованы по компонентам системы: - Backend (API) - Frontend (Admin) - Frontend (Client) - Общие библиотеки

#### 3.4.2. Управление ветками

В репозитории определены следующие основные ветки: - main — стабильная версия, соответствующая текущему релизу - develop — ветка разработки, интеграция функциональности - feature/[название] — ветки для разработки новых функций - bugfix/[номер-ошибки] — ветки для исправления ошибок - release/[версия] — ветки подготовки релиза

#### 3.4.3. Процесс применения изменений

1. Разработка выполняется в отдельных ветках для функций или исправлений
2. Изменения интегрируются в основную ветку разработки через механизм Pull Request
3. Перед слиянием проводится код-ревью и автоматическое тестирование
4. Для подготовки релиза создается отдельная ветка
5. После финального тестирования релиз публикуется в основной ветке с присвоением версии

#### 3.4.4. Управление зависимостями

* Backend-зависимости управляются через requirements.txt
* Frontend-зависимости управляются через package.json
* Фиксация версий зависимостей для обеспечения стабильной сборки

### 3.5. Тестирование

#### 3.5.1. Уровни тестирования

1. **Unit-тестирование**

* Тесты отдельных компонентов и функций
* Автоматизированное выполнение при каждом коммите
* Измерение покрытия кода тестами

1. **Интеграционное тестирование**

* Тестирование взаимодействия компонентов
* Проверка корректности API
* Тестирование интеграций с внешними системами

1. **Системное тестирование**

* Проверка полного цикла работы функций
* Тестирование производительности
* Тестирование безопасности

1. **Приемочное тестирование**

* Проверка соответствия требованиям
* Пользовательские сценарии
* Валидация бизнес-процессов

#### 3.5.2. Автоматизация тестирования

* Для Backend используются соответствующие фреймворки тестирования
* Для Frontend применяются инструменты тестирования JavaScript-приложений
* CI/CD система автоматически запускает тесты при изменениях в репозитории

#### 3.5.3. Тестовые окружения

* Локальное окружение разработчика
* Тестовый сервер (staging)
* Предпродуктивное окружение
* Продуктивное окружение

### 3.6. Документирование

#### 3.6.1. Типы документации

1. **Техническая документация**

* Архитектура системы
* API документация
* Инструкции по развертыванию
* Руководство по разработке

1. **Пользовательская документация**

* Руководство пользователя
* Руководство администратора
* Обучающие материалы

1. **Проектная документация**

* Техническое задание
* Спецификации требований
* Планы тестирования

#### 3.6.2. Управление документацией

* Хранение в системе контроля версий вместе с кодом
* Использование Markdown для удобства поддержки
* Автоматическая генерация API-документации из кода
* Регулярное обновление при выпуске новых версий

## 4. Персонал для обеспечения поддержки

### 4.1. Требуемые роли

#### 4.1.1. Разработчики

* **Backend-разработчик**
* Требования: опыт работы с Python, FastAPI, PostgreSQL
* Обязанности: разработка и поддержка серверной части, API, интеграций
* Количество: минимум 2 специалиста
* **Frontend-разработчик**
* Требования: опыт работы с React, Next.js, TypeScript
* Обязанности: разработка и поддержка пользовательского интерфейса
* Количество: минимум 2 специалиста

#### 4.1.2. DevOps-инженер

* Требования: опыт работы с Linux, Docker, CI/CD, AWS/S3
* Обязанности: настройка и поддержка инфраструктуры, мониторинг, безопасность
* Количество: минимум 1 специалист

#### 4.1.3. Тестировщик

* Требования: опыт функционального и автоматизированного тестирования
* Обязанности: разработка тест-кейсов, выполнение тестирования, отчетность
* Количество: минимум 1 специалист

#### 4.1.4. Технический писатель

* Требования: опыт создания технической документации
* Обязанности: подготовка и обновление документации
* Количество: минимум 1 специалист (возможно совмещение)

#### 4.1.5. Специалисты технической поддержки

* **Первая линия поддержки**
* Требования: базовые знания системы, коммуникативные навыки
* Обязанности: прием обращений, консультирование пользователей
* Количество: 1-2 специалиста
* **Вторая линия поддержки**
* Требования: углубленные знания системы, опыт диагностики
* Обязанности: решение сложных проблем