Описание технических средств хранения исходного текста и объектного кода программного обеспечения, а также технических средств компиляции исходного текста в объектный код программного обеспечения МБорд

Наименование ПО: МБорд
Версия: 1.0.0
Организация-разработчик: ООО "Метрикор"
Дата: 2026-05-05

---

1. Общие положения

Настоящий документ описывает технические средства, используемые для хранения исходного текста программного обеспечения МБорд, средства компиляции (сборки) исходного текста в объектный (исполняемый) код, а также средства хранения объектного кода.

Программное обеспечение МБорд представляет собой веб-приложение, состоящее из серверной части (Backend, язык Python) и клиентской части (Frontend, язык TypeScript/JavaScript). Серверная часть является интерпретируемой и не требует классической компиляции в машинный код. Клиентская часть проходит этап транспиляции (TypeScript → JavaScript) и бандлинга (объединение модулей в оптимизированные пакеты). Обе части упаковываются в Docker-образы для развёртывания.

Все компоненты ПО разрабатываются и хранятся на территории Российской Федерации с использованием инфраструктуры, находящейся под полным контролем организации-разработчика.

---

2. Технические средства хранения исходного кода

2.1 Система управления версиями

Параметр	Значение
Система	GitLab (self-hosted)
Адрес	gitlab.pdd.dev
Версия	GitLab Community Edition
Размещение	Выделенный виртуальный сервер, Российская Федерация
Протокол доступа	HTTPS, SSH
Аутентификация	SSH-ключи, персональные токены доступа

2.2 Структура репозиториев

Исходный код МБорд организован в 4 отдельных репозитория:

№	Репозиторий	Назначение	Основные технологии
1	`mboard-audit/backend`	Серверная часть (API, бизнес-логика, работа с БД)	Python 3.11+, FastAPI, SQLAlchemy, Alembic
2	`mboard-audit/frontend`	Клиентская часть (веб-интерфейс)	React 19, TypeScript 5.9, Vite 7
3	`mboard-audit/deploy`	Конфигурации развёртывания, Docker Compose, скрипты	Docker, Nginx, shell scripts
4	`mboard-audit/arch`	Архитектурная документация, регистрационные документы	Markdown

2.3 Стратегия ветвления (Branching Strategy)

Проект использует упрощённую модель ветвления:

Ветка	Назначение	Политика
`main`	Основная ветка, содержит стабильный код	Защищённая, мерж только через Merge Request
`feature/*`	Ветки разработки новых функций	Создаются от main, мержатся обратно в main
`fix/*`	Ветки исправления ошибок	Создаются от main, мержатся обратно в main
`release/*`	Ветки подготовки релизов	Создаются от main для финальной стабилизации

2.4 Версионирование

- Семантическое версионирование (SemVer): MAJOR.MINOR.PATCH
- Версия MINOR синхронизируется между всеми 4 репозиториями (backend, frontend, deploy, arch)
- Теги Git создаются при каждом релизе (формат: `vX.Y.Z`)
- История изменений ведётся в файле `releases.md`

2.5 Резервное копирование исходного кода

Параметр	Значение
Метод	Полный бэкап GitLab (репозитории + метаданные)
Частота	Ежедневно
Хранение бэкапов	Объектное хранилище (S3-совместимое), Yandex Cloud
Глубина хранения	30 дней
Проверка восстановления	Ежемесячно

---

3. Технические средства компиляции

3.1 Backend (Python)

Python является интерпретируемым языком программирования. Исходный код серверной части не проходит классическую компиляцию в машинный код, а исполняется интерпретатором CPython.

Параметр	Значение
Интерпретатор	CPython 3.11+
Пакетный менеджер	UV (unified Python package manager)
Файл зависимостей	`pyproject.toml`, `uv.lock`
Линтер/форматтер	Ruff
Тестирование	pytest
Сборка Docker-образа	Dockerfile (multi-stage build)

Процесс сборки Backend:

1. Установка базового образа Python 3.11 (slim)
2. Копирование файлов зависимостей (`pyproject.toml` , `uv.lock`)
3. Установка зависимостей через UV (`uv sync --frozen`)
4. Копирование исходного кода приложения
5. Конфигурация entrypoint (запуск через uvicorn)
6. Сборка финального Docker-образа

3.2 Frontend (TypeScript/JavaScript)

Клиентская часть написана на TypeScript и проходит этапы транспиляции и бандлинга.

Параметр	Значение
Язык исходного кода	TypeScript 5.9
Язык объектного кода	JavaScript (ES2020+)
Транспилятор	tsc (TypeScript Compiler) через Vite
Бандлер	Vite 7 (использует Rollup для production build)
Пакетный менеджер	npm
Файл зависимостей	`package.json`, `package-lock.json`
Конфигурация сборки	`vite.config.ts`, `tsconfig.json`

Процесс сборки Frontend:

1. Установка зависимостей (`npm ci`)
2. Проверка типов TypeScript (`tsc --noEmit`)
3. Линтинг кода
4. Продакшен-сборка (`vite build`):
   - Транспиляция TypeScript → JavaScript
   - Tree-shaking (удаление неиспользуемого кода)
   - Минификация JavaScript и CSS
   - Генерация хешированных имён файлов (cache busting)
   - Генерация source maps (для отладки)
5. Результат: директория `dist/` со статическими файлами (HTML, JS, CSS, assets)
6. Сборка Docker-образа (Nginx + статика)

3.3 CI/CD Pipeline

Автоматизированная сборка выполняется средствами GitLab CI/CD.

Параметр	Значение
Система CI/CD	GitLab CI/CD
Runner	GitLab Runner (self-hosted, тег: vasa3)
Executor	Docker
Конфигурация	`.gitlab-ci.yml` в корне каждого репозитория

Этапы пайплайна (stages):

Этап	Описание	Условие запуска
lint	Статический анализ кода (Ruff для Python, ESLint для TypeScript)	Каждый push
test	Запуск автотестов (pytest, Playwright)	Каждый push
build	Сборка Docker-образов	Push в main, теги
deploy	Развёртывание на целевом сервере	Push тега vX.Y.Z

3.4 Сборка Docker-образов

Параметр	Значение
Инструмент	Docker BuildKit
Стратегия	Multi-stage builds (разделение build и runtime)
Базовые образы	python:3.11-slim (backend), node:20-alpine (frontend build), nginx:alpine (frontend runtime)
Кэширование	Docker layer cache, BuildKit cache mounts

---

4. Технические средства хранения объектного кода

4.1 GitLab Container Registry

Собранные Docker-образы хранятся в GitLab Container Registry.

Параметр	Значение
Система	GitLab Container Registry
Адрес	registry.gitlab.pdd.dev
Хранилище	S3-совместимое объектное хранилище (Yandex Cloud)
Протокол доступа	HTTPS
Аутентификация	GitLab-токены, CI/CD job tokens

4.2 Структура образов

Образ	Описание	Формат тега
`mboard/backend`	Серверная часть (Python + FastAPI)	`vX.Y.Z`, `latest`
`mboard/frontend`	Клиентская часть (Nginx + статика)	`vX.Y.Z`, `latest`

4.3 Версионирование образов

- Каждый релиз получает тег в формате `vX.Y.Z` (соответствует Git-тегу)
- Тег `latest` указывает на последний стабильный образ из ветки `main`
- Образы из feature-веток получают теги вида `feature-<name>-<short-sha>`
- Хранение: минимум 5 последних релизных версий

4.4 Целостность и безопасность

- Образы подписываются SHA256-дайджестами (Docker Content Trust)
- Доступ к Registry ограничен авторизованными пользователями и CI/CD job tokens
- Сканирование образов на уязвимости (периодическое)
- Базовые образы обновляются при выходе security-патчей

---

5. Дополнительные технические средства

5.1 Docker и Docker Compose

Параметр	Значение
Назначение	Контейнеризация и оркестрация сервисов
Docker Engine	24+
Docker Compose	v2
Количество контейнеров	4 (timescaledb, redis, backend, frontend)
Конфигурация	`docker-compose.yml`, `docker-compose.dev.yml`

5.2 Nginx

Параметр	Значение
Назначение	Реверс-прокси, TLS-терминация, раздача статики фронтенда
Версия	Последняя stable (alpine)
Конфигурация	Встроена в Docker-образ frontend

5.3 PostgreSQL / TimescaleDB

Параметр	Значение
Назначение	Основная СУБД для хранения данных приложения и телеметрии
PostgreSQL	15
TimescaleDB	Последняя совместимая версия
Особенности	Гипертаблицы для временных рядов, непрерывные агрегаты, компрессия
Бэкап	pg_basebackup + WAL archiving

5.4 Redis

Параметр	Значение
Назначение	Кэширование, хранение сессий, Pub/Sub для real-time событий
Версия	7
Persistence	AOF (Append-Only File)
Лимит памяти	Настраиваемый (200 МБ для dev)

---

6. Список сокращений

Сокращение	Расшифровка
API	Application Programming Interface — программный интерфейс приложения
AOF	Append-Only File — метод персистентности Redis
CI/CD	Continuous Integration / Continuous Delivery — непрерывная интеграция и доставка
CSS	Cascading Style Sheets — каскадные таблицы стилей
Docker	Платформа контейнеризации приложений
ES2020	ECMAScript 2020 — стандарт языка JavaScript
Git	Распределённая система управления версиями
HMR	Hot Module Replacement — горячая замена модулей
HTML	HyperText Markup Language — язык разметки
HTTPS	HyperText Transfer Protocol Secure — защищённый протокол передачи данных
JS	JavaScript — язык программирования
JSON	JavaScript Object Notation — формат обмена данными
JWT	JSON Web Token — токен аутентификации
ORM	Object-Relational Mapping — объектно-реляционное отображение
RBAC	Role-Based Access Control — управление доступом на основе ролей
REST	Representational State Transfer — архитектурный стиль
S3	Simple Storage Service — объектное хранилище
SemVer	Semantic Versioning — семантическое версионирование
SHA256	Secure Hash Algorithm 256-bit — алгоритм хеширования
SQL	Structured Query Language — язык структурированных запросов
SSH	Secure Shell — защищённый сетевой протокол
СУБД	Система управления базами данных
TLS	Transport Layer Security — протокол защиты транспортного уровня
TS	TypeScript — типизированный язык программирования
WAL	Write-Ahead Log — журнал упреждающей записи