

Описание функциональных характеристик программного обеспечения «INTEBRIX Optimization Platform»

Версия ПО: **1.0.0** (сборка от 21.08.2026, фиксация 0ec5bd7d) · Правообладатель: ООО «ИНТЕБРИКС» (ИНН 9707059284) · Редакция документа: 19.09.2026

Документ для реестра российского ПО (ПП РФ № 1236): назначение, состав четырёх расчётных ядер и сквозного контура с переменными, ограничениями, целевыми функциями и результатами, общие функции, интеграция, разграничение доступа, методы и решатели, ограничения применимости.

1. Назначение и область применения

INTEBRIX Optimization Platform — программное обеспечение для оптимизационного планирования производства и логистики. Программа принимает описание производственно-логистической системы заказчика (ресурсы, площадки, узлы сети, транспортные средства, заказы, ограничения, тарифы), формирует по нему математическую модель оптимизации, решает её встроенными решателями и возвращает исполнимый план с числовыми показателями и обоснованием.

1.1 Решаемый класс задач

Программа предназначена для поддержки принятия решений на четырёх горизонтах планирования:

Горизонт	Задача	Расчётное ядро
Год / квартал	Объёмное планирование: что и сколько производить и продавать по периодам (S&OP)	Strategic Production Planner
Квартал / месяц	Потоки на логистической сети: откуда, куда, каким видом транспорта, в каком объёме	Strategic Flow Planner
Смена / неделя	Операционное расписание производственных ресурсов (APS)	Production Planner
Сутки	Маршрутизация и диспетчеризация транспорта	Dispatcher

Горизонт	Задача	Расчётное ядро
	(VRP/VRPTW)	
Сквозной	Согласование всех четырёх горизонтов в одном прогоне	Контур Supply Chain (Pipeline)

1.2 Область применения

Промышленное производство (пищевая промышленность, химия, металлургия, фармацевтика, добыча и обогащение), транспортно-логистические операторы, распределительные и сбытовые сети, инфраструктурные и сетевые компании. Программа не привязана к отраслевой специфике на уровне кода: состав слоёв модели, типы работ, единицы измерения и ограничения задаются данными и пользовательскими правилами.

1.3 Форма поставки и режимы развёртывания

Программа поставляется как серверное приложение с веб-интерфейсом. Развёртывание — Docker Compose, пять сервисов: веб-сервер и статика (nginx), сервер приложений (FastAPI под Gunicorn/Uvicorn), фоновый исполнитель расчётов (Celery), СУБД (PostgreSQL 16), кэш и брокер очереди (Valkey, Redis-совместимый). Поддерживаются режимы `cloud`, `on_prem` и `air_gapped` (изолированный контур без доступа в интернет); режим задаётся параметром конфигурации `deploy_mode`. Один и тот же набор образов используется во всех режимах; отличия задаются переменными окружения при запуске.

Программа распространяется двумя способами:

1. **Размещение в инфраструктуре заказчика** (основной способ): заказчик получает комплект поставки и разворачивает программу на собственном сервере или в собственной облачной инфраструктуре, в том числе в закрытом контуре без доступа в интернет; порядок установки описан в документе 02 «Инструкция по установке».
2. **Предоставление доступа как к интернет-сервису (SaaS)**: программа разворачивается тем же комплектом на вычислительных мощностях правообладателя, расположенных на территории Российской Федерации, а заказчику предоставляется доступ к веб-интерфейсу и программному интерфейсу через сеть «Интернет». Функциональный состав программы в обоих способах одинаков.

Клиентская часть — одностраничное веб-приложение (React 19, TypeScript, Vite), работающее в браузерах на актуальных версиях движков; отдельная установка на рабочие места не требуется. Язык интерфейса по умолчанию — русский; в состав входит также английская локализация, переключаемая пользователем.

2. Состав программы

Программа состоит из четырёх независимых расчётных ядер и сквозного контура, объединённых общим хранилищем моделей (Data Hub), общим программным интерфейсом, единой системой заданий и общей подсистемой разграничения доступа.

2.1 Production Planner — операционное расписание производства (APS)

Программный префикс: `/api/process-scheduler`.

Решаемые задачи

Построение расписания работы производственных ресурсов на заданном горизонте, разбитом на периоды одинаковой длительности (параметры `horizonPeriods`, `periodLengthMin`): какой ресурс, в каком периоде, на каком узле производственной сети выполняет какую работу и в каком объёме. Поддерживаются три режима запуска:

- `heuristic` — приоритетная эвристика с просмотром вперёд (по умолчанию);
- `milp` — точная постановка частично-целочисленного линейного программирования;
- `replan` — перепланирование от факта: назначения до отсечки `factCutoff` фиксируются, пересчитывается только остаток горизонта.

Типы работ канонизируются к четырём категориям: `operation` (производительная работа на узле), `setup` (подготовка/переналадка), `transit` (перемещение между узлами), `idle` (простой). Пользовательские наименования типов работ на русском и английском отображаются в эти категории автоматически.

Переменные решения (режим MILP)

- $x[u, p, n] \in \{0, 1\}$ — ресурс u выполняет работу на узле n в периоде p ;
- $y[u, p] \in \{0, 1\}$ — ресурс u находится в перемещении в периоде p ;
- $z[u, p] \in \{0, 1\}$ — ресурс u простаивает в периоде p .

Поддерживаемые ограничения

- взаимная исключительность состояний ресурса: $\sum_n x[u, p, n] + y[u, p] + z[u, p] = 1$ для каждого ресурса и периода;
- вместимость узла по числу одновременно работающих ресурсов: $\sum_u x[u, p, n] \leq \text{resourceSlots}[n]$;
- объём работ на узле: $\sum_u \sum_p \text{rate}(u) \cdot x[u, p, n] \leq \text{workVolume}[n]$;
- закрепления (`pins`) — принудительное назначение ресурса на узел или в простой в конкретном периоде;
- окна допустимости работ — запрет назначения вне разрешённого интервала;
- непрерывность перемещений: время переезда между узлами рассчитывается по кратчайшему пути в графе (алгоритм Дейкстры) с учётом скорости типа ресурса;
- матрица переналадок «продукт A → продукт B» с временем переналадки (`setup_time_min`);
- предшествование операций в технологическом графе;

- рабочий календарь и плановые остановки/ремонт ресурсов (флаг `enforce_calendar_hours`, список `unavailableResources`);
- директивные сроки заказов (флаг `enforce_due_dates`);
- минимальный размер партии (флаг `enforce_min_batch`);
- список недоступных ресурсов и заморозка факта до отсечки.

Флаги `enforce_*` переключают ограничение между жёстким (нарушение запрещено) и мягким (нарушение допустимо и штрафуются).

Целевая функция

Максимизация суммарного выполненного объёма работ, взвешенного по приоритету и ценности задания. В эвристическом режиме используется составная оценка, включающая штраф за опоздание относительно директивного срока (вес `tardinessWeight`), стоимость переналадки и потери на перемещениях.

Результаты расчёта

- перечень назначений: ресурс × период × узел × продукт × объём × тип работы;
- показатели: число периодов производительной работы, перемещений, простоев и переналадок; суммарный выпущенный объём; загрузка ресурсов (по фактическому интервалу работ и по полному горизонту); длительность расписания (`makespan`); количество и тип узких мест; число незапланированных заказов от общего числа;
- детализация узких мест по ресурсам и узлам;
- перечень заказов, не вошедших в расписание;
- рекомендации по расшивке узких мест;
- служебные сведения: применённый решатель, время расчёта, число ресурсов и периодов, отсечка факта, состояние флагов ограничений.

2.2 Dispatcher — маршрутизация и диспетчеризация транспорта (VRP/VRPTW)

Программный префикс: `/api/dispatcher`.

Решаемые задачи

Построение суточного плана рейсов: назначение заявок на транспортные средства, последовательность объезда точек, расчёт времени прибытия, стоимости и загрузки. Расчёт выполняется в два шага, каждый доступен отдельным вызовом:

1. **Генерация маршрутов-кандидатов** (`/build-dispatcher`) — построение допустимых маршрутов: k кратчайших путей в дорожном графе, последовательное наращивание маршрута (одиночные заявки → пары → жадное расширение до `maxOrdersPerRoute`), отсечение по порогу `pruneThreshold` и по числу рассматриваемых соседей `maxNeighbours`, предварительная проверка выполнимости временных окон и грузоподъёмности; дополнительно формируются повторные рейсы (несколько ходок одного ТС с возвратом в депо).

2. **Оптимальный отбор маршрутов (/solve-dispatcher)** — задача покрытия множества заявок в постановке частично-целочисленного линейного программирования.

Переменные решения

- $x_i \in \{0, 1\}$ — маршрут-кандидат i включён в план;
- $y_o \in \{0, 1\}$ — заявка o обслужена.

Поддерживаемые ограничения

- каждая заявка входит не более чем в один выбранный маршрут: $\sum_{i \ni o} x_i \leq 1$;
- связь обслуживания и выбора маршрута: $y_o \leq \sum_{i \ni o} x_i$;
- мощность парка: по классам эквивалентности взаимозаменяемых ТС $\sum x \leq N_{\text{класса}}$, либо не более одного маршрута на конкретное ТС;
- грузоподъёмность по массе (capacity) и по объёму (volume_capacity, м³);
- временные окна доставки windowEarliest / windowLatest и директивный срок deadline;
- время обслуживания в точке serviceTime;
- окно смены водителя shiftStart / shiftEnd;
- предельная длительность рейса maxRouteTime и предельный пробег maxDistance;
- несовместимость грузов (incompatibility_rules — перечень пар типов груза, запрещённых к совместной перевозке);
- несколько депо отгрузки и возврат в депо (allow_multi_depot_return);
- дробление заявки между несколькими ТС (allow_split);
- разнородный парк: удельная стоимость ₽/км, почасовая ставка ₽/час, скорость, базовое депо.

Жёсткость трёх ограничений переключается флагами: enforce_hard_windows (по умолчанию выключен — окно мягкое, нарушение штрафуются), enforce_shift_end и enforce_max_route_time (по умолчанию включены).

Целевая функция

Двухэтапная лексикографическая оптимизация:

- **Этап 1** — максимизация покрытия заявок с весами по приоритету, за вычетом малого весового слагаемого по стоимости (устраняет вырожденность);
- **Этап 2** — достигнутое покрытие фиксируется ограничением, минимизируется суммарная стоимость плана;
- **Этап 2b** — при неизменной стоимости дополнительно минимизируется число порожних выездов.

Стоимость маршрута складывается из линейной перевозочной стоимости, стоимости порожнего пробега (км × ₽/км), штрафа за опоздание (₽/час × часы опоздания), штрафа за нарушение временного окна и штрафа за выход за смену.

Результаты расчёта

- выбранные маршруты: последовательность точек, назначенное ТС, состав заявок, пробег, стоимость, масса и объём, признак повторного рейса;
- перечень необслуженных заявок **с указанием причины**: `infeasible_window` (не проходит по окну), `infeasible_capacity` (не проходит по грузоподъёмности), `infeasible_shift` (не укладывается в смену), `no_route` (нет маршрута), `rejected_by_solver` (отброшено при оптимизации);
- показатели: суммарные стоимость и пробег, доля покрытия, суммарные часы опоздания и штраф за опоздание, число задействованных ТС, порожний пробег (км), суммарное время обслуживания и ожидания, доля прибытий вовремя, покрытие по классам приоритета, почасовые затраты, число рейсов и рейсов на одно ТС, загрузка по объёму, размерность решённой модели (число переменных и ограничений);
- двойственные оценки и связывающие ограничения (анализ чувствительности);
- рекомендации по изменению парка и параметров.

Расчёт матрицы расстояний

Вызов `/api/dispatcher/distance-matrix` заполняет таблицу расстояний и времён между точками по географическим координатам (не более 200 узлов за вызов). По умолчанию применяется встроенный геодезический расчёт по ортодромии с коэффициентом извилистости 1,3 и скоростью по умолчанию 50 км/ч. При явно включённой дорожной маршрутизации (п. 4.6) источником служит сервис OSRM в контуре заказчика (профили `driving`, `walking`, `cycling`), результат кэшируется, а при недоступности сервиса выполняется тот же геодезический расчёт; в ответе возвращается признак источника (`osrm` / `haversine` / `cache`) и причина переключения.

Операционный контур диспетчеризации

Отдельная группа вызовов (`/operational/*`) обслуживает посуточную работу: реестр операционных моделей, расчёт дня, ведение календаря доступности транспортных средств и водителей по датам, повторный расчёт дня после изменения доступности.

2.3 Strategic Production Planner — объёмное планирование производства и сбыта (S&OP)

Программный префикс: `/api/strategic-production`.

Решаемые задачи

Формирование объёмного плана на горизонте из нескольких периодов: какие ресурсы и площадки задействовать, сколько производить, сколько перерабатывать, через какие каналы и на какие рынки отгружать. Модель строится из динамического набора слоёв — жёсткой отраслевой схемы нет.

Слой модели:

Слой	Содержание	Ограничение по вместимости
resources	Ресурсы и оборудование (производительность, коэффициент технической готовности)	нет
sites	Производственные площадки	есть
storage	Промежуточное хранение	есть
processing	Переработка	есть
channels	Каналы отгрузки, терминалы	есть
sales	Рынки и покупатели (спрос)	есть

Связи между слоями задаются рёбрами со стоимостью, допустимостью и коэффициентом эффективности. Поддерживаются восемь типов связей, включая прямые (площадка → канал, площадка → сбыт) в обход переработки. Вместимость задаётся как единым значением, так и отдельно по каждому периоду.

Переменные решения

- $\text{choice} \in \{0,1\}$ — бинарный выбор варианта по каждой строке допустимой комбинации активных слоёв и периодов (ресурс × площадка × переработка × канал × сбыт × период);
- производные непрерывные величины, выраженные через **choice**: объём $\text{unit_volume} = \text{производительность} \times \text{часы} \times \text{КТГ\%} \times \text{эффективность\%} \times \text{choice}$, удельные и операционные затраты, выручка, маржа, логистические затраты по рёбрам.

Поддерживаемые ограничения

Ограничения формируются автоматически из данных слоёв и дополняются пользовательскими правилами.

Автоматические (переключаются флагами):

- **enforce_capacity** (по умолчанию включён) — вместимость площадок, переработки, хранения, каналов, в том числе по периодам;
- **enforce_demand** (включён) — не превышение спроса рынка;
- **enforce_balance** (выключен) — баланс потоков «вход = выход» на слоях хранения и переработки;
- **enforce_min_utilization** (выключен) — минимальная загрузка площадок и переработки за период, порог по умолчанию 0,3.

Пользовательские правила задаются структурой: тип (*capacity, allocation, budget, ratio, minimum*), слой, поле, оператор ($\leq$, $\geq$, $=$), значение, группировка (например, по площадке и периоду), фильтр по подмножеству объектов, признак жёсткости. Мягкое правило реализуется через переменную нарушения (*slack*) со штрафом, задаваемым пользователем.

Целевые функции (режимы `buildMode`)

- `maxProduction` — максимизация суммарного объёма производства;
- `minCost` — минимизация суммарных затрат;
- `maxProfit` — максимизация прибыли (выручка минус затраты либо маржа $\times$ поток);
- `cheapMaxProduction` — максимизация объёма с приоритетом над стоимостью при равном объёме.

Во всех режимах из целевой функции вычитается сумма штрафов за нарушение мягких ограничений.

Результаты расчёта

- строки плана: площадка, переработка, канал, рынок, период, объём, затраты, выручка, маржа;
- значение целевой функции и статус решения (`OPTIMAL`, `FEASIBLE`, `INFEASIBLE`, `NO_SOLUTION_FOUND`, `TIMEOUT`);
- размерность модели: число переменных и ограничений;
- анализ чувствительности: теневые цены, слэки, перечень связывающих ограничений;
- сводные показатели: взвешенная загрузка мощностей, выполнение спроса по рынкам, узкие места, движение запасов;
- рекомендации, сформированные из перечисленных показателей.

Дополнительные функции ядра

- `/forecast` — прогноз спроса методом двойного экспоненциального сглаживания Хольта (параметры `alpha`, `beta`, горизонт до 24 периодов) с расчётом внутривыборочных ошибок `MAPE` и систематического смещения; при коротких рядах применяются вырожденные методы (наивный, наивный линейный);
- `/feasibility-check` — быстрая предпроверка допустимости модели до полного расчёта: возвращает оценку 0–100 и уровень `healthy` / `tight` / `infeasible` / `unknown` (проверка не завершилась за отведённое время);
- `/constraints/templates` — библиотека шаблонов ограничений.

2.4 Strategic Flow Planner — потоки на логистической сети

Программный префикс: `/api/strategic-flow`.

Решаемые задачи

Распределение грузопотоков по мультимодальной транспортной сети: от источников (заводы, элеваторы, месторождения) через транзитные узлы, терминалы и пункты

пропуска до рынков потребления. Задача решается в постановке многопродуктового потока минимальной стоимости (multi-commodity flow problem). Расчёт разбит на этапы, каждый доступен отдельным вызовом:

1. `/generate-routes` — генерация маршрутов по мультимодальному графу;
2. `/build-task` — сборка задачи распределения (маршруты × грузы × выходы × рынки);
3. `/solve-logistics` — оптимизация распределения потоков;
4. `/analyze-bottlenecks` — анализ узких мест решённой сети;
5. `/development-candidates` и `/resolve-with-development` — подбор программы развития сети под бюджет;
6. `/generate-hooks` — генерация обходных маршрутов по остаточному графу.

Модель сети

Узлы сети расщепляются по видам транспорта: `rail` (железнодорожный), `auto` (автомобильный), `river` (речной), `air` (воздушный), `pipe` (трубопроводный). Перегрузка между видами транспорта в одном узле моделируется отдельным перегрузочным ребром с собственной пропускной способностью. Число перевалок в маршруте ограничивается параметром `maxTransfers`. Маршруты строятся как k кратчайших простых путей (алгоритм Йена) по критерию расстояния, стоимости или времени.

Переменные решения

$x_r \geq 0$ — непрерывная переменная: объём груза, направляемый по маршруту r .

Поддерживаемые ограничения

- доступный объём источника по паре (груз, источник): $\sum \{x_r \text{ из источника}\} \leq \text{доступный объём}$ — флаг `enforce_source_capacity` (включён по умолчанию);
- лимиты пунктов пропуска / выходов по паре (выход, груз) — флаг `enforce_exit_limits` (включён);
- ёмкость рынка по паре (рынок, груз) — флаг `enforce_market_demand` (включён);
- пропускная способность рёбер, отдельно по направлениям, с суммированием параллельных связей и перегрузочных рёбер — флаг `enforce_edge_capacity` (включён);
- минимальная загрузка ненулевых рёбер — мягкое ограничение $\sum x_r + \text{slack} \geq \text{порог} \times \text{пропускная способность}$ (флаг `enforce_min_throughput`, по умолчанию выключен, порог 0,20, штраф за нарушение 1000 на единицу).

Отключение любого жёсткого флага фиксируется в результатах счётчиком пропущенных ограничений — состав фактически применённых ограничений всегда виден в ответе.

Целевые функции (режимы mode)

- `maxVolume` — максимизация перевезённого объёма;
- `maxProfit` — максимизация прибыли (прибыль на тонну × объём);
- `minCost` — минимизация суммарной стоимости перевозки (стоимость на тонну × объём).

При включённом ограничении минимальной загрузки из целевой функции вычитается штраф за нарушения.

Результаты расчёта

- распределение потоков по маршрутам (откуда, куда, груз, вид транспорта, объём, стоимость);
- загрузка рёбер сети;
- сводка по источникам: отгруженный объём, лимит, коэффициент использования;
- значение целевой функции, статус решения, размерность модели;
- анализ чувствительности (теневые цены, связывающие ограничения);
- сводные показатели: взвешенная загрузка сети, выполнение спроса по рынкам, перечень узких мест, предупреждения, перечень рёбер с невыполненным минимальным порогом загрузки.

Подбор программы развития сети

Модуль формирует и оценивает мероприятия по развитию инфраструктуры трёх типов: расширение насыщенных рёбер, создание новых перегрузочных узлов, строительство новых прямых связей. Кандидаты ранжируются по теневым ценам линейной релаксации. Отбор портфеля мероприятий — задача о рюкзаке с бинарными переменными выбора и ограничением суммарной стоимости бюджетом; учитывается синергия пар мероприятий (по умолчанию +20 % к суммарному эффекту). После отбора сеть пересчитывается с применёнными мероприятиями, возвращается прирост целевой функции.

Базовые стоимостные допущения задаются параметрами и по умолчанию равны: расширение пропускной способности — 500 000 ₽ на тонну с коэффициентами по видам транспорта (железнодорожный 1,0; автомобильный 1,6; речной 0,7; морской 0,6; трубопроводный 0,5), новый перегрузочный узел — 80 000 000 ₽, новая связь — 1 000 000 ₽ на километр. Стоимости переопределяются входными данными.

2.5 Сквозной контур Supply Chain (Pipeline)

Программный префикс: /api/pipeline.

Контур выполняет последовательность шагов, каждый из которых — расчёт одного из четырёх ядер, и передаёт результат каждого шага на вход следующему. Типовая последовательность: объёмный план → потоки на сети → маршруты доставки → расписание производства; состав и порядок шагов задаётся пользователем.

Правила передачи данных между ядрами

Передача формализована набором правил, каждое из которых сопровождается текстовым описанием применённого преобразования, отображаемым в интерфейсе:

- **Strategic Production → Strategic Flow:** агрегирование планового объёма по паре (номенклатура, площадка) и подстановка в лимит груза соответствующего источника сети;
- **Strategic Flow → Dispatcher:** извлечение потоков, приходящих в узлы-депо, и формирование на их основе заявок на доставку;

- **Strategic Production → Production Planner:** передача номенклатуры и объёмов плана в состав заданий расписания.

Правила отключаются и настраиваются по шагам; шаг может брать данные из собственного источника в Data Hub.

Диагностика расхождений между ядрами

После каждого шага выполняется проверка согласованности результата с входными данными следующего шага. Реализованные правила:

- плановый объём производства превышает пропускную способность сети по данной номенклатуре — расхождение с указанием превышения в абсолютном и относительном выражении;
- объём, доставляемый сетью в депо, превышает суммарную грузоподъёмность парка;
- номенклатура из объёмного плана отсутствует в перечне продуктов расписания;
- предыдущий шаг завершился с ошибкой.

Каждое расхождение имеет уровень (info, warning, critical), заголовок, детализацию с числами и, где это возможно, предлагаемое исправление, применяемое одним действием (например, поднять лимит груза до планового объёма или ослабить соответствующий флаг ограничения). Расхождения не прерывают выполнение контура.

Повторный расчёт

Поддерживается повторный запуск с произвольного шага: результаты предшествующих шагов берутся из кэша, пересчитываются только последующие. Конфигурация контура сохраняется как сущность Data Hub и используется повторно.

3. Общие функции

3.1 Подготовка и проверка исходных данных

Единое хранилище моделей (Data Hub). Все ядра работают с общим набором сущностей четырёх типов: набор данных (dataset), модель (model), результат (result), набор ограничений (constraint_set). Поддерживаются категоризация, теги, привязка к ядру, текстовое описание, поиск и фильтрация, версионирование (номер версии инкрементируется при изменении), копирование сущности, перевод в архивный статус, удаление.

Загрузка данных. Форматы файлов: CSV, TSV, XLSX, XLS, JSON. Предельный размер одной загрузки — 25 МБ. Перед сохранением доступен предварительный просмотр разобранного файла (/api/datahub/preview) с определением состава колонок и числа записей; для книг Excel обрабатываются несколько листов.

Валидация по схеме. К сущности можно привязать описание структуры в формате JSON Schema. При сохранении данные проверяются на соответствие схеме; при несоответствии

возвращается перечень ошибок с указанием пути к проблемному полю и текстом ошибки (возвращается не более 25 первых ошибок, чтобы ответ оставался читаемым на заведомо некорректном наборе).

Контроль полноты входных данных перед расчётом. Если во входных данных отсутствуют обязательные сущности, расчёт не запускается: возвращается статус `empty_input` и перечень текстовых предупреждений на русском языке (например: «Нет заявок и продуктов для планирования», «Не задан парк ресурсов», «Нет узлов производственной сети»). Это исключает выдачу формально успешного, но пустого плана.

Предпроверка допустимости. Для объёмного планирования доступна быстрая проверка модели до полного расчёта с оценкой 0–100 и уровнем `healthy` / `tight` / `infeasible` / `unknown` (проверка не завершилась за отведённое время).

3.2 Сценарное сравнение

Вызов `/api/analysis/compare` запускает произвольное число сценариев (каждый — самостоятельный набор входных данных для указанного ядра), выполняет их и возвращает сопоставление: значения целевых функций, ключевые показатели по каждому сценарию, попарные отличия и ранжирование. Сценарии, завершившиеся с ошибкой, помечаются и не искажают сравнение.

3.3 Анализ чувствительности решения

Реализованы два независимых механизма.

Развёртка по параметру (`/api/analysis/what-if`): указывается параметр входных данных (адресация точечной нотацией, например `fleet.0.capacity_weight`) и перечень значений; программа выполняет расчёт для каждого значения и возвращает набор данных для диаграммы влияния («торнадо»).

Постоптимизационный анализ (извлекается автоматически после каждого расчёта, где это применимо): по каждому ограничению — правая часть, слэк, теневая цена (двойственная оценка), признак связывающего; по каждой переменной — значение, приведённая стоимость, коэффициент целевой функции; сводно — перечень связывающих ограничений и ранжированный перечень ресурсов-узких мест. Для задач линейного программирования при использовании решателя HiGHS доступен полный набор двойственных оценок; для частично-целочисленных задач оценки извлекаются по линейной релаксации, при работе на резервном решателе CBC набор ограничен. Отсутствие двойственных оценок не прерывает расчёт.

3.4 Диагностика несовместимых ограничений

- **Явный статус решения.** Программа различает и по-разному сообщает: `INFEASIBLE` (допустимого решения не существует), `NO_SOLUTION_FOUND` / `TIMEOUT` (решение не найдено за отведённое время — модель может быть разрешима, задача слишком велика), `UNBOUNDED`, `ERROR`. Сообщения формулируются на русском языке и содержат рекомендацию по действию.

- **Причина отказа по каждому объекту.** В маршрутизации по каждой необслуженной заявке возвращается код причины (нарушение окна, нехватка грузоподъёмности, выход за смену, отсутствие маршрута, отклонение при оптимизации).
- **Ослабление и повторный расчёт (/api/analysis/relax-resolve):** указываются ограничения к ослаблению и коэффициент от 1,0 до 2,0; программа умножает соответствующие правые части (поля вместимости, грузоподъёмности, объёма, лимитов) и повторяет расчёт. Если ограничения не указаны, ослабляются все ограничения вместимости.
- **Связывающие ограничения из анализа чувствительности** указывают, какие именно ограничения определяют решение и в каком порядке их ослабление даёт наибольший эффект.
- **Мягкие ограничения со штрафом:** там, где это допустимо постановкой (временные окна маршрутизации, минимальная загрузка сети, пользовательские правила объёмного плана), нарушение разрешается и штрафуются — программа возвращает решение с явным перечнем и величиной нарушений вместо отказа.
- **Межъядерные расхождения** сквозного контура (см. п. 2.5).
- **Диагностика неразрешимости.** При статусе INFEASIBLE выполняется поиск неустранимо-несовместимого подмножества ограничений методом эластичной релаксации с последующим фильтром удаления; перечень взаимно несовместимых ограничений возвращается в составе ответа расчёта в поле `infeasibility_report`. Поддерживается ядрами расписания производства, объёмного планирования и сетевой оптимизации. В ядре оперативной диспетчеризации причина отказа сообщается в разрезе каждой необслуженной заявки (см. выше).

3.5 История расчётов

Каждый расчёт сохраняется в журнале расчётов: ядро, вызванная операция, наименование модели, статус, значение целевой функции, длительность, пользователь, отметка времени, а также **эхо входных данных** — состав узлов, связей, парка, заказов и отраслевого профиля. Хранение входных данных вместе с результатом обеспечивает воспроизведение расчёта и восстановление показателей при повторном открытии.

Доступны: постраничный перечень расчётов с фильтрацией (/api/runs), получение отдельного расчёта, удаление отдельного расчёта и групповое удаление, аналитические выборки (/api/analytics/runs, /api/analytics/compare, /api/analytics/trends, /api/analytics/stats) — динамика показателей по времени и сопоставление расчётов между собой. Настраивается срок хранения расчётов.

3.6 Выполнение расчётов и контроль хода

Расчёты запускаются в двух режимах:

- **Асинхронный** (основной): вызов возвращает идентификатор задания и адрес канала уведомлений; ход выполнения передаётся по протоколу WebSocket (`/ws/jobs/{job_id}`) с указанием фазы (`preprocessing`, `building`, `solving`, `postprocessing`, `completed`), процента готовности и текстового сообщения. Место выполнения выбирается автоматически по оценке размера модели: короткие задания выполняются фоновой задачей процесса сервера приложений, длинные (оценка модели свыше 200 000 переменных либо запрошенный бюджет времени свыше 170 с) — через очередь заданий в отдельном процессе фонового исполнителя; принятое решение и его причина возвращаются в ответе на постановку задания (поле `execution: inline` или `queue`).
- **Синхронный** (`/sync`-варианты вызовов): расчёт выполняется в рамках запроса с ограничением по времени (180 с) и жёсткими ограничениями на размер входных данных.

Состояния задания: `pending`, `running`, `completed`, `failed`, `cancelled` (отмена — `POST /api/jobs/{job_id}/cancel`). Управление заданиями — `/api/jobs`.

3.7 Отчёты и экспорт

- **PDF** (`/api/reports/pdf`) — отчёт по результату расчёта с показателями и таблицами;
- **Excel (XLSX)** (`/api/reports/excel`) — выгрузка результата в книгу Excel;
- **Отложенные отчёты** — постановка отчёта на расписание, перечень запланированных отчётов, отмена;
- **Выгрузка исходных и результирующих данных** через программный интерфейс в формате JSON;
- **Уведомления**: рассылка по электронной почте (настройки SMTP на уровне организации, шаблоны писем на Jinja2) и в мессенджер Telegram.

3.8 Наблюдаемость и эксплуатационный контроль

- проверки состояния: `/health` (доступность приложения), `/ready` (готовность с проверкой СУБД, кэша, фонового исполнителя и загрузки расчётных модулей), `/solvers` (перечень доступных решателей);
 - метрики в формате Prometheus (`/metrics`): число и длительность HTTP-запросов, число расчётов, активные задания, потребление памяти процессом;
 - структурированные журналы в формате JSON Lines: журнал приложения, журнал аудита, журнал клиентских ошибок;
 - неизменяемый журнал аудита действий пользователей.
-

4. Интеграция с внешними системами

4.1 Программный интерфейс

Интерфейс — REST поверх HTTP/HTTPS, обмен в формате JSON. Спецификация формируется в формате OpenAPI 3.1 и доступна в двух интерактивных представлениях: Swagger UI (`/docs`) и ReDoc (`/redoc`); публикация спецификации и интерактивных

представлений по умолчанию выключена и включается параметром конфигурации `INTEBRIX_ENABLE_API_DOCS`. Базовый путь — `/api`. В сборке 1.0.0 спецификация содержит 160 путей и 178 операций в 27 тематических группах. Каждое расчётное ядро имеет как асинхронные, так и синхронные вызовы. Запросы и ответы описаны схемами Pydantic v2 с примерами заполнения.

Двусторонний канал реального времени — WebSocket `/ws/jobs/{job_id}` для передачи хода расчёта.

Ограничение частоты обращений (значения по умолчанию, задаются конфигурацией): на уровне приложения — 400 запросов в минуту с одного адреса на общие вызовы и 40 запросов в минуту на вызовы расчётных ядер; на одного пользователя — 4000 запросов в час, из них к расчётным ядрам — 240 в час; на уровне веб-сервера nginx — 600 запросов в минуту с одного адреса на `/api/` и 10 запросов в минуту на `/api/auth/`. При превышении возвращается код 429.

4.2 Форматы обмена

Направление	Форматы
Программный интерфейс	JSON
Загрузка исходных данных	CSV, TSV, XLSX, XLS, JSON
Выгрузка результатов	PDF, XLSX, JSON
Исходящие уведомления	JSON (веб-перехватчики), HTML/текст (электронная почта), текст (Telegram)

4.3 Веб-перехватчики (webhooks)

Регистрация внешних адресов для получения событий. Поддерживаемые события: `model.created`, `model.deleted`, `solver.started`, `solver.completed`, `solver.failed`, `user.invited`, `user.deactivated`. Тело уведомления подписывается алгоритмом HMAC-SHA256 общим секретом; тип события передаётся отдельным заголовком `X-Intebrix-Event`. При неуспешной доставке выполняются повторные попытки с нарастающей задержкой; история доставок сохраняется и доступна для просмотра. Подписка настраивается по составу событий и включается/отключается независимо.

4.4 Ключи программного доступа

Для машинных клиентов предусмотрены ключи API с префиксом `intbx_`. В базе данных хранится только хэш ключа; поиск при аутентификации выполняется по индексированному префиксу, сверка — с постоянным временем сравнения. Ключ можно ограничить областью действия, отозвать и просмотреть в маскированном виде. Ключ является равноправной альтернативой токenu JWT при обращении к программному интерфейсу.

4.5 Коннекторы к внешним системам

Подключаемая архитектура коннекторов; каждый коннектор реализует проверку соединения, чтение данных и запись данных. В поставку входят:

Коннектор	Тип	Параметры подключения
PostgreSQL	база данных	адрес, порт, база, пользователь, пароль, TLS
REST API	внешний сервис	базовый адрес, способ авторизации (нет / bearer / basic / ключ), дополнительные заголовки
CSV	файл	путь, разделитель, кодировка, наличие строки заголовков
Excel	файл	путь, имя листа

Параметры подключений сохраняются на уровне организации; учётные данные хранятся в зашифрованном виде. Доступны перечень типов коннекторов, проверка соединения, чтение и запись, реестр настроенных подключений.

4.6 Внешние сервисы (необязательные)

Программа сохраняет полную расчётную функциональность без доступа в интернет. Внешние сервисы используются только как дополнение и отключаются конфигурацией:

- **Сервис дорожной маршрутизации OSRM** — автозаполнение матрицы расстояний и времён в ядре маршрутизации. **По умолчанию маршрутизация выключена (`INTEBRIX_OSRM_ENABLED=false`), и обращения к сервису не выполняются; в режиме `air_gapped` она выключается принудительно.** Адрес сервиса задаётся параметром `INTEBRIX_OSRM_BASE_URL`; в конфигурации сборки 1.0.0 значение этого параметра по умолчанию указывает на публичный сервер проекта OSRM, однако оно применяется только при явном включении маршрутизации, а для промышленной эксплуатации предназначено указание собственного экземпляра OSRM, развёрнутого в контуре заказчика. При выключенной маршрутизации или недоступности сервиса применяется встроенный геодезический расчёт.
- **Картографическая подложка и сервисы карт** — фон карты в интерфейсе (поставщик выбирается параметром `INTEBRIX_MAP_PROVIDER`, по умолчанию Яндекс Карты; для закрытого контура — локальный набор тайлов) и геокодирование по ключам заказчика; на расчёт не влияют. Состав и порядок отключения — документ 08, раздел 8.

- **Сервис генеративных моделей YandexGPT** — текстовая интерпретация результатов расчёта, пояснения к узким местам, подсказки по заполнению данных. Активен только при заданных ключе и идентификаторе каталога Yandex Cloud; отключается параметром `INTEBRIX_LLM_PROVIDER=none`; при отключении соответствующие функции возвращают результат правил без обращения во внешний сервис.
- **Внешний провайдер OIDC** — единый вход (см. п. 5.2); выключен, пока не задан адрес поставщика.
- **Приёмник ошибок (Sentry-совместимый)** — включается только при заданном адресе приёмника (`INTEBRIX_SENTRY_DSN`) и только в режиме `cloud`; по умолчанию адрес не задан, и ошибки записываются в локальные файлы журналов во всех режимах.
- **Уведомления в Telegram** — выключены, пока не задан токен.

Обращений к серверам правообладателя, принудительных обновлений и удалённого управления в программе нет.

Аналитические функции, работающие без обращения к внешним сервисам: выявление отклонений в результатах по правилам (перегрузка мощности, всплеск затрат, падение загрузки, риск срыва срока, всплеск спроса, простой ресурса, смещение узкого места; четыре уровня значимости), генерация рекомендаций по отклонениям, разбор запросов на естественном языке к результатам расчёта.

5. Разграничение доступа

5.1 Организации и изоляция данных

Программа многопользовательская и многоарендная: данные принадлежат организации. Изоляция обеспечивается тремя независимыми уровнями:

1. **Контекст арендатора** определяется по записи пользователя в базе данных, а не по утверждению в токене, — подмена значения в токене не даёт доступа к чужим данным;
2. **Фильтрация на уровне запросов**: все обращения к данным проходят через типовые функции, добавляющие условие по идентификатору организации; соблюдение этого правила контролируется статическим анализатором исходного текста, входящим в штатный набор автоматических тестов;
3. **Политики защиты строк (Row-Level Security) PostgreSQL** на таблицах, содержащих данные арендаторов, — как эшелонированная защита на случай ошибки на уровне 2.

Корректность изоляции покрыта сквозным тестом, проверяющим попытки доступа из одной организации к объектам другой.

5.2 Аутентификация

- **Вход по логину и паролю** с выдачей пары токенов JWT (алгоритм подписи HS256): токен доступа сроком 60 минут и токен обновления сроком 30 суток;
- **ротация токена обновления** с хранением идентификатора выданного токена (jti) во внешнем хранилище — повторное использование отозванного токена отклоняется;
- **двухфакторная аутентификация** по одноразовым паролям TOTP (RFC 6238) с выдачей QR-кода для привязки приложения-аутентификатора; включается и отключается пользователем;
- **политика паролей** с требованиями к длине и составу символов, смена пароля;
- **блокировка учётной записи** после серии неуспешных попыток входа;
- **публичная регистрация отсутствует** — учётные записи создаются администратором либо по выданным ключам активации;
- **единый вход по OIDC** (/api/sso/config, /api/sso/login, /api/sso/callback) с внешним поставщиком идентификации;
- **ключи программного доступа** для машинных клиентов (п. 4.4).

5.3 Ролевая модель

Четыре роли с накопительным набором полномочий:

Роль	Полномочия
viewer	чтение
analyst	чтение, создание моделей, запуск расчётов
manager	то же + удаление моделей, управление составом команды
admin	то же + управление организацией, административные функции

Иерархия: viewer < analyst < manager < admin. Проверка выполняется декларативно на уровне обработчика — по конкретному полномочию либо по минимальному уровню роли. В производственном режиме сообщение об отказе не раскрывает требуемое полномочие и роль пользователя; каждый отказ фиксируется в журнале аудита. Изменение роли пользователя доступно администратору.

5.4 Лицензионное ограничение состава модулей

Доступ к каждому расчётному ядру проверяется отдельно: если ядро не включено в лицензию организации, вызов отклоняется с кодом 403 и сообщением «Модуль не включён в вашу лицензию». Лицензия представляет собой подписанный лицензионный документ, выпускаемый правообладателем: одна строка с данными лицензии (продукт и редакция сборки, организация, режим развёртывания, срок действия, срок поддержки, предел пользователей, перечень ядер и возможностей, льготный период, при необходимости — аппаратный отпечаток установки) и подписью Ed25519. Подпись

проверяется открытым ключом, встроенным в программу; закрытый ключ находится только у правообладателя, установка документ выпустить не может. Проверка выполняется целиком внутри установки, без обращений к внешним серверам: документ вносится администратором через интерфейс программы, при внесении проверяются подпись и принадлежность организации, при каждом обращении к ядрам — адресат документа (сборка и, если указан, аппаратный отпечаток). Предусмотрены отзыв лицензии администратором, льготный период по истечении срока действия (по умолчанию 7 дней — функции работают, выдаётся предупреждение о необходимости продления; после его окончания расчётные ядра недоступны до внесения нового документа), бессрочные лицензии, а также экспорт и импорт документа для изолированных контуров. Дополнительно ведётся учёт лимитов тарифного плана (число пользователей, размер модели) и потребления.

5.5 Аудит

Каждое действие, изменяющее состояние, записывается в журнал аудита: идентификатор и адрес электронной почты пользователя, тип действия, тип и идентификатор объекта, дополнительные сведения, IP-адрес, отметка времени. Журнал защищён на уровне базы данных: разрешена только вставка, попытки изменения и удаления записей отклоняются триггером. Записи журнала не удаляются.

6. Используемые методы и решатели

6.1 Математические методы

Метод	Где применяется
Частично-целочисленное линейное программирование (MILP)	отбор маршрутов (задача покрытия множества), объёмное планирование (слоистая модель с бинарным выбором), расписание производства в точном режиме, отбор программы развития сети (задача о рюкзаке 0/1)
Линейное программирование (LP)	распределение потоков на сети (многопродуктовый поток), линейные релаксации для ранжирования кандидатов
Двухэтапная лексикографическая оптимизация	маршрутизация: сначала покрытие, затем стоимость при зафиксированном покрытии
Мягкие ограничения через переменные нарушения (elastic constraints)	временные окна маршрутизации, минимальная загрузка рёбер сети, пользовательские правила объёмного плана
Постоптимизационный анализ	все ядра, где решатель

Метод	Где применяется
(двойственные оценки, приведённые стоимости, слэки)	предоставляет двойственную информацию
Эвристики: последовательное конструирование, жадное расширение, локальный поиск, просмотр вперёд	генерация маршрутов-кандидатов, расписание производства в быстром режиме
Алгоритмы на графах: кратчайший путь (Дейкстра), k кратчайших простых путей (Йен)	построение маршрутов, расчёт перемещений между узлами, генерация обходных путей по остаточному графу
Экспоненциальное сглаживание с трендом (метод Хольта)	прогноз спроса
Правила выявления отклонений	анализ результатов расчёта, диагностика узких мест, межъядерные расхождения

6.2 Решатели

Оптимизационные задачи решаются **встроенными решателями, входящими в состав поставки:**

- **HiGHS** (лицензия MIT, разработка Эдинбургского университета) — основной решатель задач линейного и частично-целочисленного линейного программирования; поставляется в составе дистрибутива в виде библиотеки;
- **CBC / COIN-OR** (лицензия EPL-2.0) — резервный решатель; поставляется в составе программы как обязательная зависимость обёртки python-mip.

Выбор решателя выполняется автоматически и одинаково для всех ядер: используется HiGHS, при его недоступности — CBC. Решатель может быть задан явно параметром запроса `_solver_override` (значения `highs`, `cbc`). Параметр конфигурации `INTEBRIX_SOLVER_NAME` на выбор решателя не влияет и сохранён для совместимости. Наличие и работоспособность решателей проверяется вызовом `/solvers` и учитывается в проверке готовности `/ready`.

Проприетарные решатели в состав поставки не входят и для работы программы не требуются. В исходном коде предусмотрена необязательная интеграция с решателем CPLEX на случай наличия у заказчика собственной лицензии: проверка `/solvers` содержит пробу наличия пакета `cpex` (признак `solver_CPLEX`). По умолчанию интеграция выключена, пакет в поставку не входит, признак `solver_CPLEX=false` является штатным и на статус готовности не влияет; расчётные ядра версии 1.0.0 обращаются только к HiGHS и CBC. Приобретение и установка сторонних коммерческих решателей (CPLEX, Gurobi, FICO Xpress и аналогов) не требуется.

6.3 Прочие расчётные библиотеки

NetworkX (алгоритмы на графах), NumPy, SciPy, pandas (числовые расчёты и обработка табличных данных) — все под лицензией BSD-3-Clause.

6.4 Лицензионная чистота

Все сторонние компоненты поставки предоставлены на условиях открытых лицензий без выплат (royalty-free); выплаты иностранным правообладателям по лицензионным договорам отсутствуют. Компоненты под лицензиями, налагающими копилефтные обязательства на собственный код программы (AGPL, SSPL, BSL, Commons Clause, Hippocratic), в составе отсутствуют; для компонентов с дизъюнктивными лицензиями зафиксирован выбор разрешительной ветки (в частности, jszip: «MIT OR GPL-3.0-or-later» — выбрана MIT). Единственные компоненты семейства GPL — нативные библиотеки времени выполнения GCC внутри колеса cbcbox (зависимость обёртки python-mip): libgfortran — GPL-3.0 с оговоркой GCC Runtime Library Exception 3.1, прямо разрешающей использование в проприетарном ПО, и libquadmath — LGPL-2.1, используется только динамической компоновкой без изменений. Решатель CBC и библиотеки COIN-OR — EPL-2.0. Ни одна из лицензий не содержит ограничений на распространение и использование программы на территории Российской Федерации. Полный перечень компонентов с указанием лицензий — файл THIRD-PARTY-NOTICES.txt, входящий в поставку; сводка по ключевым компонентам — документ 08, раздел 10.

Компоненты react-leaflet и @react-leaflet/core (лицензия Hippocratic-2.1, не одобрена OSI, содержит ограничение сферы применения и право отзыва) из поставки исключены (аудит состава сторонних компонентов от 03.08.2026). Соответствующая функциональность реализована собственной оболочкой над библиотекой Leaflet (лицензия BSD-2-Clause). Указанные компоненты отсутствуют в манифестах зависимостей и в собираемом дистрибутиве.

7. Ограничения применимости

Приведённые ниже границы получены измерением на синтетических моделях возрастающего размера (замеры от 03–04.08.2026) и зафиксированы в коде в виде защитных ограничений. Указаны размерности, при которых расчёт остаётся практичным.

7.1 Ограничения по размерности задач

Ядро	Практическая граница	Характер роста
Dispatcher	Синхронный расчёт жёстко ограничен 250 заявками и 250 узлами; свыше — только асинхронный режим. При 60 узлах и 50 заявках расчёт не	комбинаторный

Ядро	Практическая граница	Характер роста
	завершался за 5 минут. Число маршрутов-кандидатов растёт комбинаторно: 10 узлов — 704 кандидата, 20 узлов — 2 272	
Production Planner (режим MILP)	16 узлов × 4 ресурса × 72 периода даёт около 121 700 строк ограничений — допустимое решение не найдено за 94 с. Практический режим для задач такого масштаба — эвристический	комбинаторный
Strategic Production Planner	540 бинарных переменных — оптимум за 3,2 с; 2 304 бинарных переменных — расчёт не завершился за 60 с. Число бинарных переменных равно производству: ресурсы × площадки × переработка × сбыт × периоды	комбинаторный по производству слоёв
Strategic Flow Planner	Рост полиномиальный (порядка $n^2, 2 \dots n^2, 4$): 200 узлов — 18–20 с, 400 узлов — около 87 с	полиномиальный

7.2 Встроенные защитные механизмы

- **Автоматическое снижение параметров генерации маршрутов** при большом размере задачи (по большему из числа узлов и числа заявок): свыше 100 — число рассматриваемых соседей сокращается до 10, свыше 150 — число кратчайших путей до 2, свыше 200 — максимальное число заявок в маршруте до 3. Механизм отключается явным параметром запроса `disable_auto_degrade`.
- **Автоматический переход с точного режима на эвристический** в расписании производства: перед расчётом выполняется оценка размера задачи, и при превышении внутреннего порога расчёт выполняется эвристикой. Оценка носит приближённый характер; факт и причина перехода в каждом случае возвращаются

в составе результата расчёта явным признаком, поэтому применённый режим всегда известен пользователю.

- **Ограничение времени синхронных вызовов** — 180 с для всех синхронных вызовов расчётных ядер (генерация и отбор маршрутов, расписание производства, объёмное планирование, генерация маршрутов сети, сборка задачи, расчёт сети, генерация обходных путей); бюджет решателя на синхронном пути не превышает 170 с. При превышении возвращается код 504 с рекомендацией сократить модель или перейти на асинхронный вызов.
- **Ограничение времени работы решателя:** задаётся параметром запроса `timeLimitSec` (по умолчанию в зависимости от ядра: расписание производства — 30 с, маршрутизация — 60 с, объёмное планирование — 120 с, расчёт сети — 300 с). Схема запроса принимает значения до 7200 с (поле `timeLimit`), фактически применяемый бюджет ограничивается потолком очереди `INTEBRIX_QUEUE_TIME_CEILING_SEC` (по умолчанию 600 с) и внутренним пределом расчётных ядер 600 с. Дополнительно задаётся допустимый зазор оптимальности (`mirGap / mir_gap / gapTolerance`): по умолчанию 0,001; допустимые значения 0,001; 0,005; 0,01; 0,015; 0,02, иные приводятся к ближайшему допустимому.
- **Ограничение размера загружаемого файла** — 25 МБ на файл (проверяется и веб-сервером, и приложением). Для больших объёмов исходных данных предусмотрены коннекторы и предварительная агрегация.
- **Ограничение частоты обращений** — по умолчанию 40 запросов в минуту с одного адреса и 240 запросов в час на пользователя на вызовы расчётных ядер (полный перечень ограничений — п. 4.1).

7.3 Рекомендуемые аппаратные ресурсы

Масштаб установки	Конфигурация
Минимальная (одиночный сервер, все компоненты; документ 02, раздел 2.1)	2 ядра x86-64, 4 ГБ ОЗУ, 30 ГБ SSD (не менее 15 ГБ свободно в каталоге данных Docker)
Демонстрационная / малая (1–10 пользователей)	4 vCPU, 16 ГБ ОЗУ, 64 ГБ SSD
Средняя (10–50 пользователей)	8 vCPU, 32 ГБ ОЗУ, 200 ГБ SSD
Крупная (50–500 пользователей)	отказоустойчивый кластер: 3 × (8 vCPU, 32 ГБ ОЗУ), выделенные СУБД и кэш

Операционные системы серверной части, на которых работоспособность программы подтверждена испытаниями: **Astra Linux Special Edition 1.8.2 «Орёл»** и **РЕД ОС 8.0.2** (документ 05 «Протокол испытаний на российских операционных системах» от 18.08.2026: изделие развёрнуто и проверено на обеих операционных системах обоими путями поставки, результаты расчётов совпадают). Установка на иные дистрибутивы Linux x86-64 с Docker Engine 24 и новее и Docker Compose 2.20 и новее выполняется по запросу, без гарантии совместимости. Средства контейнеризации для обеих испытанных операционных систем устанавливаются из их штатных репозиториев.

7.4 Функциональные ограничения

Программа явно не реализует следующие функции, чтобы исключить неверные ожидания:

- **имитационное дискретно-событийное моделирование** — программа выполняет оптимизацию, а не имитацию поведения системы во времени;
 - **приём телеметрии в реальном времени** — интеграция с системами спутникового мониторинга транспорта выполняется через входящие веб-перехватчики; собственный приём по промышленным протоколам (MQTT, OPC UA) не реализован;
 - **финансовое планирование и управление денежными потоками** — за пределами области применения;
 - **планирование персонала (workforce planning)** — учитываются окна смен в маршрутизации, полноценное планирование графиков персонала не реализовано;
 - **мобильные приложения** — интерфейс адаптирован к экранам мобильных устройств, отдельных приложений для iOS и Android не предусмотрено;
 - **модели свыше приблизительно 100 тысяч переменных** — предсказуемая работа решателя подтверждена до этого порядка; для больших моделей требуется декомпозиция задачи.
-

8. Сведения о версии

- **Наименование программы:** INTEBRIX Optimization Platform
 - **Правообладатель:** ООО «ИНТЕБРИКС»
 - **Версия:** 1.0.0 (сборка от 21.08.2026, идентификатор фиксации исходного текста 0ec5bd7d; номер версии хранится в файле VERSION поставки и сообщается вызовом GET /health)
 - **Язык интерфейса:** русский (по умолчанию); дополнительно — английская локализация, переключаемая пользователем
 - **Способы распространения:** размещение в инфраструктуре заказчика; предоставление доступа как к интернет-сервису (SaaS) на мощностях правообладателя в Российской Федерации (п. 1.3)
 - **Операционные системы, подтверждённые испытаниями:** Astra Linux Special Edition 1.8.2, РЕД ОС 8.0.2 (п. 7.3)
 - **Класс программного обеспечения** (решение правообладателя от 19.08.2026, коды по действующему классификатору программ для ЭВМ и баз данных): основной класс — **08.19 «Программы производственного планирования (APS)»** (расписание производственных ресурсов и объёмное планирование); дополнительный класс — **09.13 «Средства управления складом и цепочками поставок (WMS, SCM)»** (оперативная диспетчеризация транспорта и оптимизация потоков в логистической сети). Сочетание классов отражает двойной профиль программы — производственное планирование и логистическая оптимизация.
-