MorphCluster Sessions

Пакет `@morphcluster/sessions` предоставляет сервис управления сессиями пользователей в экосистеме MorphCluster. Он состоит из двух основных компонентов: `Sessions` (публичный API) и `SessionsStorage` (персистентное хранение в PostgreSQL). Сервис обеспечивает создание, проверку, удаление и листинг сессионных токенов с поддержкой административных прав.

Импорт

```
import { Sessions, SessionsStorage } from '@morphcluster/sessions';
```

Sessions

Назначение: Централизованное управление сессиями пользователей. Предоставляет методы для создания, удаления, получения и валидации токенов. Реализует двухуровневое кэширование: in-memory для быстрого доступа и постоянное хранение в PostgreSQL через внутренний сервис `SessionsStorage`.

Наследует: `ServiceRequire` (из `@morphcluster/core`)

Зависимости (`requirements`):

Сервис	Тип	Описание
`SessionsStorage`	обязательный	Сервис персистентного хранения сессий в БД

Конфигурация:

Параметры передаются через объект `config`, который `Sessions` получает при создании. Основные настройки PostgreSQL извлекаются из `config.common` и передаются в `SessionsStorage`:

- `config.common.postgresUri` → `config.pg.uri` (строка подключения к PostgreSQL)
- `config.common.timeZone` → `config.pg.timezone` (часовой пояс для сессий)

Сам `Sessions` не имеет собственных параметров, кроме тех, что нужны для настройки `SessionsStorage`.

Запросы (методы)

Все запросы описаны в `service-schema.mjs` и регистрируются через `setSvcSchema`.

Метод	HTTP	Права	Параметры	Ответ	Описание

`get`	POST	`anonymous: true`	`{ token: string }`	`Session`	Получить сессию по токену. Сначала проверяется in-memory кэш, затем PostgreSQL. Если сессия не найдена, выбрасывается `ComplexError('Invalid Token', 'InvalidToken')`.
`list`	POST	`needAdmin: true`	`{ filters?: { login?, isAdmin?, permanent? } }`	`Session[]`	Получить список сессий из PostgreSQL по фильтрам. Параметр `filters` обязателен, но может быть пустым объектом.
`create`	POST	`needAdmin: true`	`{ userId: number, login: string, isAdmin: boolean }`	`{ token: string }`	Создать новую сессию. Генерирует UUID-токен, сохраняет в памяти и асинхронно записывает в БД через `SessionsStorage.insert`.
`delete`	POST	`needAdmin: true`	`{ token: string }`	`void`	Удалить сессию. Удаляет из БД и из in-memory кэша, если запись имеет тип `"storage"`.
`deleteMany`	POST	`needAdmin: true`	`{ filters: { login?, isAdmin?, permanent? } }`	`number` (rowCount)	Удалить несколько сессий по фильтрам. Фильтры обязательны. Также очищает in-memory кэш полностью.
`setPermanent`	POST	`needAdmin: true`	`{ token: string, permanent: boolean }`	`number` (rowCount)	Установить флаг постоянной сессии (не истекающей).

Вспомогательный метод `validateHttp`

Используется другими сервисами (например, `FastifyGateway`, `Eventer`) для проверки сессии из HTTP-заголовка `Authorization: Bearer <token>`. Сигнатура:

```
async validateHttp(request, reply, log = null, needAdmin = false)
```

- Извлекает токен из заголовка `Authorization`.
- Вызывает `this.get(token, "common", log)`.
- Если `needAdmin === true`, проверяет наличие `isAdmin` в сессии.
- В случае отсутствия токена или недостатка прав выбрасывает `ComplexError`.

Внутреннее устройство

- **In-memory кэш:** массив `this.sessions`, хранящий объекты сессий (тип `Session`). Используется для быстрого поиска и хранения «невалидных» записей (чтобы не обращаться в БД повторно для несуществующих токенов).
- **Генерация токенов:** метод `makeid()` возвращает UUID v4 через `crypto.randomUUID()`.
- **Поиск сессии:** метод `get()` сначала ищет в `this.sessions`. Если не найдено и токен соответствует формату UUID, выполняет запрос в `SessionsStorage.byId()`. При отрицательном результате создаёт в памяти запись `{ token, type: "invalid" }` и выбрасывает ошибку `InvalidToken`.
- **Удаление:** метод `delete()` удаляет запись из БД, затем из кэша, но только если сессия в кэше имеет тип `"storage"`. Это предотвращает повторное появление невалидной записи.

Жизненный цикл

- `start(log)`: вызывает `super.start(log)`, что в свою очередь запускает зависимый `SessionsStorage`.
- `stop()`: наследуется от `ServiceRequire` (останавливает `SessionsStorage`).

Примечания

- Токены создаются с использованием криптографически стойкого UUID.
- Асинхронная вставка в БД при создании сессии не ожидается (`this.SessionsStorage.insert(newSession).then()`), поэтому возможна кратковременная несогласованность между кэшем и базой.
- Метод `deleteMany` полностью очищает in-memory кэш (`this.sessions = []`), что может привести к лишним запросам к БД для ещё действительных сессий.

SessionsStorage

Назначение: Персистентное хранение сессий в PostgreSQL. Обеспечивает миграцию схемы БД, пул соединений и все CRUD-операции над таблицей `sessions`.

Наследует: `ServiceRequire`

Зависимости (`requirements`):

Сервис	Тип	Описание
`PostgresMigrator`	обязательный	Сервис миграции схемы БД (из `@morphcluster/postgres-migrator`)

Конфигурация: передаётся через объект `config.pg`:

- `uri` — строка подключения к PostgreSQL (формируется из `config.common.postgresUri`).
- `schema` — имя схемы БД, в которой будет создана таблица `sessions`.
- `timezone` — часовой пояс (из `config.common.timeZone`).

Методы (внутренние, вызываются сервисом `Sessions`)

Метод	Параметры	Возврат	Описание
`byId(id)`	`id: string`	`Session` или `null`	Получить сессию по идентификатору (токену). Выполняет запрос `SELECT ... WHERE id = $1`.
`insert(session)`	`session: Session`	`void`	Вставить новую сессию в таблицу.
`list(req)`	`req: { login?, isAdmin?, permanent? }`	`Session[]`	Получить список сессий с динамическими фильтрами. Поля `isAdmin` и `permanent` приводятся к `0/1`.
`delete(id)`	`id: string`	`number` (rowCount)	Удалить одну сессию по ID.
`deleteMany(req)`	`req: { login?, isAdmin?, permanent? }`	`number` (rowCount)	Удалить сессии по фильтрам.
`setPemanent(id, perm)`	`id: string`, `perm: boolean`	`number` (rowCount)	Установить флаг `permanent` для сессии.

Жизненный цикл

- **`start(log)`:**
  1. Вызывает `super.start(log)` для инициализации `PostgresMigrator`.
  2. Запускает миграцию SQL-файлов из директории `./sql` (или `./dist/sessions-sql` для Webpack-сборки) с указанием схемы и строки подключения.
  3. Создаёт пул соединений `pg.Pool` с переданной строкой подключения.
  4. Проверяет соединение запросом `SELECT 1`.
- **`stop()`:** Закрывает пул соединений, если он был создан.

Примечания

- Имя метода `setPemanent` содержит опечатку (правильно `setPermanent`), но сохранено для совместимости с существующим кодом.
- Миграции находятся в подпапке `sql` относительно файла `storage.mjs`.
- Пул соединений создаётся один раз и используется всеми запросами.

Тип Session

```
/**
 * @typedef {Object} Session
 * @property {string} token
 * @property {string} type       // "storage" или "invalid"
 * @property {number} userId
 * @property {string} login
 * @property {boolean} isAdmin
 */
```

Поле `type` используется в `Sessions` для различения записей: `"storage"` — сессия из БД, `"invalid"` — кэшированное отсутствие сессии.

Пример использования

```
import { ServiceHost, Config, Logger } from '@morphcluster/core';
import { Sessions } from '@morphcluster/sessions';

const host = new ServiceHost('main');
const config = new Config({
  common: {
    postgresUri: 'postgresql://user:pass@localhost:5432/mydb',
    timeZone: 'UTC'
  },
  pg: {
    schema: 'public'
  }
});

const sessions = new Sessions(host, config.values);
host.addService(sessions, 'Sessions');

await host.start(logger);
```

```
// Создание сессии администратора
const { token } = await sessions.sendRequest('create', {
  userId: 1,
  login: 'admin',
  isAdmin: true
}, log);

// Проверка сессии
const session = await sessions.sendRequest('get', { token }, log);
console.log(session.isAdmin); // true
```

Интеграция с другими сервисами

- **Auth** из @morphcluster/core использует Sessions для аутентификации администратора.
- **FastifyGateway** и **HttpProxy** применяют Sessions.validateHttp для проверки авторизационных заголовков.
- **Eventer** из Supervisor использует Sessions для аутентификации WebSocket-подключений.

Revision #3
Created 21 June 2026 21:02:32 by Admin
Updated 26 June 2026 13:52:59 by Admin