Соглашения по стилю и API для проекта Secret Santa (v4)

# 1. Имена REST‑ресурсов и URI

* • Ресурс ⇢ существительное: /events, /participants – глаголы прячем в HTTP‑методах
* • Всегда множественное число: /events/42/participants – консистентность > грамматика
* • Иерархия = «/»: /games/{gameId}/participants/{id} – подчёркивает вложенность
* • Без завершающего «/»: GET /events ← ok, GET /events/ ← нет
* • kebab‑case, только lower‑case: /gift-exchanges вместо /giftExchanges
* • Не использовать «\_» и расширения файлов: /participants/{id}, а не /participants.json
* • Версионирование префиксом: /api/v1/events
* • Фильтры, сортировка, пагинация – query‑string: /events?year=2025&page=2&size=20

|  |  |  |
| --- | --- | --- |
| **GET** | Чтение ресурса (без побочных эффектов). | GET /api/v1/events/42 ― получить один розыгрыш. |
| **POST** | Создание нового ресурса в коллекции **или** выполнение нестандартного действия. | POST /api/v1/events ― создать розыгрыш. |
| **PUT** | Полная замена существующего ресурса (идемпотентно). | PUT /api/v1/events/42 ― обновить все поля события. |
| **PATCH** | Частичное обновление ресурса (только изменённые поля). | PATCH /api/v1/events/42 ― поменять дату жеребьёвки. |
| **DELETE** | Удаление ресурса. | DELETE /api/v1/events/42/participants/7 ― удалить участника. |
| **OPTIONS** | Узнать, какие методы/заголовки поддерживает эндпоинт. | Браузерный CORS pre‑flight для POST /api/v1/events. |
| **HEAD** | То же, что GET, но без тела ответа (проверка наличия/метаданных). | HEAD /api/v1/events/42 ― проверить, существует ли ресурс и получить ETag. |

Примеры:  
GET /api/v1/events  
POST /api/v1/events  
GET /api/v1/events/{id}  
PUT /api/v1/events/{id}  
POST /api/v1/events/{id}/participants  
GET /api/v1/events/{id}/participants  
POST /api/v1/events/{id}/assignments  
GET /api/v1/events/{id}/assignments/{id}

# 2. Java (Spring Boot) стиль кода

* • Пакеты: com.yourorg.secretsanta.<feature>, lower‑case, по функционалу.
* • Классы / интерфейсы: PascalCase – EventService, GiftAssignmentRepository.
* • Методы и переменные: camelCase – generateAssignments(), participantEmail.
* • Константы: UPPER\_SNAKE\_CASE – DEFAULT\_DRAW\_DATE.
* • DTO / Command‑объекты: ParticipantDto, CreateEventRequest, AssignmentResponse.
* • Builder‑pattern или параметр‑объект, если >3 аргументов или есть boolean‑флаги.
* • Передача «именованных» параметров: startGame(participants, isAnonymous). Никогда startGame(array, true).
* • Булин‑геттеры с префиксом is/has – isAnonymousDraw(). isNotChotoTam == ебантизм
* • Repository ↔︎ Service ↔︎ Controller – разделение слоёв.
* • Facade‑pattern для связанных под‑сервисов: все вызовы UserProfileService, UserNotificationService идут через UserService (единую «точку входа»). Внешние слои знают только UserService.
* • Strategy‑pattern для разных алгоритмов распределения подарков (classic draw, avoidance draw).
* • Тесты: \*Test.java, под src/test/java.

# 3. React + TypeScript

* • Файлы компонентов: PascalCase = имя default‑экспорта – CreateEventForm.tsx.
* • Hooks: префикс use – useParticipants, useDrawAssignments.
* • Переменные состояния: camelCase – isLoading, currentPage.
* • Named parameters через объект‑деструктуризацию: startGame({ participants, isAnonymous }).
* • Tailwind: utility‑first, повторяемые наборы в @apply или UI‑компоненты.
* • Container / Presentational pattern: логика в containers, чистый UI в components.
* • Custom Hook pattern – всё, что использует useState/useEffect более одного раза.
* • Импорты: абсолютные алиасы @components, @hooks.

# 4. PostgreSQL

* • Схемы: public или тематические (santa).
* • Таблицы: snake\_case + plural – events, participants, assignments.
* • PK: surrogate id (bigserial). FK – event\_id, participant\_id.
* • Индексы: idx\_{table}\_{columns}.
* • Materialized View pattern для отчётных агрегаций.
* • Миграции: Liquibase/Flyway с timestamp‑префиксом – 20250719‑01‑init.sql.

# 5. Комментарии и документация

* • Javadoc / TSDoc описывают поведение, не реализацию.
* • REST‑ручки описываются через OpenAPI (SpringDoc).
* • README.md: как запустить, переменные окружения, пример запроса.
* • ADR‑pattern (Architecture Decision Record) для ключевых решений.

# 6. Проверка стиля и CI

* • Java: Checkstyle + SpotBugs + Spring Boot DevTools.
* • JS/TS: ESLint (airbnb‑config) + Prettier.
* • Git hooks (Husky): npm run lint && mvn test.
* • CI‑pipeline: GitHub Actions – lint → test → build → docker‑image → deploy.

# 7. Git Workflow

* • 🔒 main – защищённая ветка, всегда деплой‑готовая. CI: 'build → test → deploy-prod'. Мёрдж только через pull‑request из dev; fast‑forward или squash.
* • 🛠 dev – интеграционная ветка текущей разработки. CI: 'build → test → deploy‑staging'. После стабилизации релиза делаем pull‑request в main.
* • 🌱 feature/<short-ticket-or-desc> – из dev; одна фича = одна ветка. Пример: feature/SS-42-random-draw.
* • 🚑 hotfix/<issue-id> – из main для urgent‑фикс в production, затем мёрдж в dev.
* • 🏷 release/<version> – (опц.) фриз фич, только баг‑фиксы, подготовка changelog.
* • Теги версий: v1.3.0 на коммите main после релиза.

\*\*Зачем такое деление?\*\*  
• main остаётся «зоной безопасности» для быстрых rollback.  
• dev собирает все фичи централизованно, уходят конфликты при прямом мёрдже множества feature‑веток.  
• feature‑ветка упрощает code‑review и позволяет параллельную работу разных разработчиков.  
• hotfix отделяет критические правки от длинного цикла релиза.

## Правила именования коммитов (Conventional Commits + scope)

* • Шаблон: <type>(<scope>): <subject>  
  subject ≤ 50 симв., в повелит. наклонении (Add, Fix, Refactor …).
* • types: feat, fix, docs, style, refactor, test, chore, build, ci, perf, revert.
* • scope — модуль/директория: (event), (participant), (ci), (\*).
* • Body (необ.) — что и почему, ссылки на issue (#42).
* • Footer для BREAKING CHANGE или Co‑Authored‑By.
* • Пример: feat(r assignment): add avoidance draw algorithm  
   fix(ci): correct Dockerfile path #108  
   docs(j readme): update run instructions

# 8. Главное

* •Кайфануть жестко пацаны не забываем

Проект делаем в портфолио, и ради того, чтобы прокачаться. Но нужно не забывать, что в первую очередь, это должно нам удовольствие приносить. Этот проект не должен стать второй женой. Он должен быть пиздатой любовницей… Ауф

FEIN.