Отчет по лабораторной работе 2 - Разработка API

Введение
В рамках лабораторной работы было разработано RESTful API для сервиса, связанного с управлением походами. Основной целью являлась реализация CRUD-операций для сущностей, таких как типы походов (Hike Types), области (Areas), файлы (Files), маршруты (Tracks), лайки (Likes) и пользователи (Users). В отчете будет рассмотрен фрагмент кода, отвечающий за работу с типами походов (Hike Types), который демонстрирует общий подход к проектированию API в рамках всего проекта.

Структура API

API построено по принципу REST и использует Spring Boot в качестве основного фреймворка. Каждый контроллер отвечает за свою сущность и предоставляет стандартные методы для работы с ней:

- `GET` — получение данных;
- `POST` — создание;
- `PUT` — обновление;
- `DELETE` — удаление.

Контроллеры защищены ролевой моделью доступа, где некоторые операции доступны только администраторам. Для обработки ошибок используется глобальный обработчик исключений (`GlobalExceptionHandler`), который возвращает структурированные сообщения об ошибках в формате JSON.

Реализация HikeTypeController

Контроллер `HikeTypeController` управляет типами походов и включает следующие методы:

```
@RestController
@RequestMapping("/hikes/types")
@RequiredArgsConstructor
```

```java
public class HikeTypeController {

  private final HikeTypeService hikeTypeService;

  @PostMapping
  public ResponseEntity<HikeTypeResponse> createHikeType(
    @RequestBody HikeTypeRequest hikeTypeRequest
  ) {
    return
ResponseEntity.ok(hikeTypeService.createHikeType(hikeTypeRequest));
  }

  @GetMapping
  public ResponseEntity<List<HikeTypeResponse>> getAllHikeType() {
    return ResponseEntity.ok(hikeTypeService.getAllHikeType());
  }

  @GetMapping("/{id}")
  public ResponseEntity<HikeTypeResponse> getHikeTypeById(
    @PathVariable Long id
  ) {
    return ResponseEntity.ok(hikeTypeService.getHikeTypeById(id));
  }

  @PutMapping("/{id}")
  public ResponseEntity<HikeTypeResponse> updateHikeType(
    @PathVariable Long id,
    @RequestBody HikeTypeRequest hikeTypeRequest
  ) {
    if (!id.equals(hikeTypeRequest.id())) {
      throw new ValidationException("Path ID and body ID must match");
    }
    return ResponseEntity.ok(
      hikeTypeService.updateHikeType(id, hikeTypeRequest)
    );
  }

  @DeleteMapping("/{id}")

  public ResponseEntity<Void> deleteHikeType(@PathVariable Long id) {
    hikeTypeService.deleteHikeType(id);
    return ResponseEntity.noContent().build();
  }
}
```

- **Создание типа похода (`POST /hikes/types`)**
  Принимает запрос с названием типа, проверяет уникальность имени и сохраняет в базу данных. В случае конфликта имен выбрасывается исключение `ConflictException`.
- **Получение списка типов (`GET /hikes/types`)**
  Доступно всем пользователям. Возвращает список всех типов походов в формате `List<HikeTypeResponse>`.
- **Получение типа по ID (`GET /hikes/types/{id}`)**
  Возвращает тип похода по указанному идентификатору. Если тип не найден, выбрасывается `ResourceNotFoundException`.
- **Обновление типа (`PUT /hikes/types/{id}`)**
  Проверяет соответствие ID в пути и теле запроса, а также уникальность нового имени. При успешном обновлении возвращает обновленные данные.
- **Удаление типа (`DELETE /hikes/types/{id}`)**
  Удаляет тип похода по ID, предварительно проверяя его существование.

Примеры запросов и ответов:

- **Создание типа:**

```
POST /hikes/types
Headers: { "Authorization": "Bearer <token>" }
Body: { "name": "Горный" }
```

Ответ (успех):

```
{ "id": 1, "name": "Горный" }
```

Ответ (ошибка, если имя уже существует):

```
{ "status": 409, "message": "HikeType with name 'Горный' already exists",
"timestamp": "2023-10-01T12:00:00Z" }
```

- **Получение списка типов:**

```
GET /hikes/types
```

Ответ:

```
[ { "id": 1, "name": "Горный" }, { "id": 2, "name": "Пешеходный" } ]
```

---

3. Обработка ошибок

Глобальный обработчик `GlobalExceptionHandler` перехватывает исключения и возвращает стандартизированные ответы:

- `ResourceNotFoundException` — 404 (Not Found);
- `ConflictException` — 409 (Conflict);
- `ValidationException` — 400 (Bad Request);
- `AuthenticationException` — 401 (Unauthorized).

Пример ошибки при неверном ID:

```
GET /hikes/types/999
```

Ответ:

```
{ "status": 404, "message": "HikeType not found with id: 999", "timestamp":
"2023-10-01T12:00:00Z" }
```

Тестирование API

Для тестирования использовался Postman. Проверялись:

- Корректность работы CRUD-операций;
- Валидация входных данных;