Charity Box Manager – Backend Documentation

Author: Tomasz Mol

System Requirements

To run this application, you need:

- **Java 21** (JDK 21) — required by the project (`<java.version>21</java.version>` in `pom.xml`)
- **Maven** — for building and running the project
- **Internet access** — the application fetches currency rates from the NBP API
- **Operating system:** Windows, Linux, or macOS (the project is platform-independent)
- **Port 8080** — the application listens on this port by default

No external database is required — the built-in H2 database is used.

How to Run the Application

To run the backend application, follow these steps:

1. Clone or extract the project repository.
2. Navigate to the backend directory:
   ```
   cd ./charity-box-manager/backend/
   ```
3. Build the project using Maven:
   ```
   mvn clean install
   ```
4. Start the application:
   ```
   mvn spring-boot:run
   ```

Once the application is running, it should be accessible at:
http://localhost:8080

General Information

General description:
An application for managing collection boxes during fundraising events for charity organizations.

Functional requirements

1. Every collection box should have a unique identifier.

2. A collection box can be assigned to only one fundraising event at a time.
3. You can only assign a collection box to a fundraising event if the collection box is empty.
4. When a collection box is unregistered, it is automatically emptied. No money is transferred to any fundraising event's account.
5. A collection box can contain money in different currencies. For simplicity, you can limit possible currencies to a subset of your choice (e.g. three different currencies).
6. Fundraising events have their own accounts where they store collected money. The account has only one currency.
7. When money is transferred from a collection box to a fundraising event's account, it is converted to the currency used by the fundraising event. The exchange rates can be hardcoded.

Optional: Fetch currency exchange rates from an online API.

Non-functional requirements

1. Create only the backend part (no UI is required).
2. The application should expose a REST API.
3. Use Java programming language and Spring Framework.
4. Use Maven or Gradle.
5. Use relational in-memory database (e.g. H2).
6. No security features (authorization, authentication, encryption etc.) are required.

System architecture

Technologies Used

- **Java 21** – main programming language
- **Spring Boot 3.1.3** – application framework for building RESTful backend services
- **Spring Data JPA** – data persistence and ORM
- **H2 Database** – in-memory relational database for development and testing
- **Maven** – build and dependency management tool
- **JUnit 5** – unit testing framework
- **Mockito** – mocking framework for tests
- **Springdoc OpenAPI** – automatic generation of OpenAPI documentation for REST API
- **IntelliJ IDEA** – recommended IDE for development

Project Structure Overview

The project consists of several key components organized as follows:

```
/
├── backend/
│   ├── .gitignore
│   ├── pom.xml
│   ├── src/
│   │   ├── main/
│   │   │   ├── java/
│   │   │   │   └── com/
│   │   │   │       └── charitybox/
│   │   │   │           ├── Main.java
│   │   │   │           ├── config/
│   │   │   │           ├── controller/
│   │   │   │           ├── dto/
│   │   │   │           ├── exception/
│   │   │   │           ├── model/
│   │   │   │           ├── repository/
│   │   │   │           └── service/
│   │   │   └── resources/
│   │   │       └── application.properties
│   │   └── test/
│   │       └── java/
│   │           └── com/
│   │               └── charitybox/
│   │                   ├── controller/
│   │                   └── service/
├── LAT25-Java-opis stażu ogólnego w Digital CC.docx
├── README.md
├── TASK.pdf
└── README.pdf
```

Description of Key Components

- `backend/.gitignore` – Defines files and directories to be excluded from version control.
- `backend/pom.xml` – Maven configuration file managing dependencies and build lifecycle.
- `backend/src/main/java/com/charitybox/`
  - `Main.java` – The entry point of the Spring Boot application.
  - `config/` – Contains configuration classes
  - `controller/` – REST API controllers handling HTTP requests.
  - `dto/` – Data Transfer Objects used for request/response mapping.
  - `exception/` – Custom exception classes and global exception handling.
  - `model/` – JPA entities mapping the application's database structure.
  - `repository/` – Interfaces extending Spring Data JPA for database access.
  - `service/` – Business logic and service layer implementations.
- `backend/src/main/resources/application.properties` – Application configuration file (e.g., DB connection, server port).
- `backend/src/test/java/com/charitybox/`
  - `controller/` – Unit tests for REST controllers.
  - `service/` – Unit tests for service layer logic.
- `LAT25-Java-opis stażu ogólnego w Digital CC.docx` – General internship description (in Polish).

- `README.md` – Project introduction and setup instructions.
- `TASK.pdf` – Task description or project assignment.
-
- `README.pdf` – Project introduction and setup instructions (in .pdf).

Database Schema

The application uses a relational in-memory H2 database with the following main entities:

- **FundraisingEvent**
  - `id` (Long, primary key)
  - `name` (String)
  - `accountBalance` (Decimal, not null)
  - `accountCurrency` (Enum: PLN, GBP, EUR, USD; not null)
  - *Relationships*: One-to-many with `CollectionBox` (a fundraising event can have multiple collection boxes assigned)
- **CollectionBox**
  - `id` (Long, primary key)
  - `fundraisingEvent_id` (Long, foreign key, nullable — assigned fundraising event)
  - `collectedAmounts` (Map<Currency, Decimal> — stores amounts per currency)
  - *Relationships*: Many-to-one with `FundraisingEvent` (a collection box can be assigned to one fundraising event at a time)
- **collection_box_amounts** (auxiliary table for mapping collected amounts per currency)
  - `collection_box_id` (Long, foreign key to CollectionBox)
  - `currency` (Enum: PLN, GBP, EUR, USD)
  - `amount` (Decimal, non-negative)

Constraints and Notes:

- Each collection box can only be assigned to one fundraising event at a time and only if it is empty.
- Collected amounts in a collection box are stored per currency.
- Fundraising event accounts operate in a single currency.
- The `amount` column in the `collection_box_amounts` table is enforced to be non-negative by a database check constraint, which is added at application startup by the `StartupSqlRunner` component.

The schema is managed automatically by JPA/Hibernate based on the entity classes, with additional constraints applied at startup.

Testing Database Operations

When the application starts, the H2 in-memory database is automatically created based on the JPA entities. You can test database operations in several ways:

- **H2 Console:**
  A web console for executing SQL queries directly is available at:
  http://localhost:8080/h2-console
  Default connection details:
    - Driver Class: org.h2.Driver
    - JDBC URL: `jdbc:h2:mem:charityboxdb`
    - User Name: `sa`
    - Password: *(empty)*
- **Unit Tests:**
  The project includes unit tests that automatically verify database operations using the in-memory H2 database.
- **REST API:**
  You can perform database operations by calling the REST API endpoints.

API Testing

You can easily test and explore all available REST API endpoints using Swagger UI, which is available at:

http://localhost:8080/swagger-ui/index.html

Swagger UI provides interactive documentation, allowing you to send requests and view responses directly from your browser.

HTTP Method	Endpoint	Description	Request Body	Response Body
POST	`/api/boxes`	Create a new collection box	–	`CollectionBox`
GET	`/api/boxes`	List all collection boxes	–	List of `CollectionBoxDto`
DELETE	`/api/boxes/{id}`	Delete (unregister) a collection box	–	–
PUT	`/api/boxes/{id}/add-money`	Add money to a collection box	`AddMoneyRequest`	–
POST	`/api/boxes/{id}/empty`	Empty a collection box	–	–

HTTP Method	Endpoint	Description	Request Body	Response Body
POST	`/api/events`	Create a new fundraising event	`FundraisingEventDto`	`FundraisingEvent`
PUT	`/api/events/{eventId}/assign-box/{boxId}`	Assign a collection box to a fundraising event	–	–
GET	`/api/events/financial-report`	Get a financial report for all fundraising events	–	List of `FundraisingEventReportDto`

Sample queries to test REST API endpoints

(queries are prepared to be pasted into a Linux terminal using `\` line continuation symbols)

1. Create a new fundraising event. - `POST /api/events`

Valid event creation

```
curl -X 'POST' \
  'http://localhost:8080/api/events' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Charity Event 1",
  "accountBalance": 0,
  "accountCurrency": "PLN"
}'
```

Creating an event without specifying a currency

If the currency is not provided, the default currency (`PLN`) is applied.
This default can be configured in `FundraisingDefaultsProperties` .

```
curl -X 'POST' \
  'http://localhost:8080/api/events' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Without Currency",
  "accountBalance": 0
}'
```

Creating an event without specifying an account balance

If the account balance is not provided, it defaults to `0`.
This default value can also be adjusted via `FundraisingDefaultsProperties`.

```
curl -X 'POST' \
  'http://localhost:8080/api/events' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Without Amount",
  "accountCurrency": "GBP"
}'
```

Creating an event without specifying both currency and account balance

If both fields are missing, default values for currency and balance are applied (`PLN` and `0` respectively).
Both defaults can be customized in `FundraisingDefaultsProperties`.

```
curl -X 'POST' \
  'http://localhost:8080/api/events' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Without Currency and Amount"
}'
```

Attempting to create an event with an invalid currency

An unsupported currency code will result in an error response.

```
curl -X 'POST' \
  'http://localhost:8080/api/events' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Charity Event",
  "accountBalance": 0,
  "accountCurrency": "WrongCurrency"
}'
```

Creating an event with a negative initial balance

There is **intentionally no validation** preventing the creation of events with a negative account balance. Allowing an initial negative (debit) balance is a **conscious design decision** for this application.

```
curl -X 'POST' \
  'http://localhost:8080/api/events' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Initial Negative Balance Charity",
  "accountBalance": -50,
  "accountCurrency": "PLN"
}'
```

---

2. Register a new collection box. - `POST /api/boxes`

Valid box registration

```
curl -X 'POST' \
  'http://localhost:8080/api/boxes' \
  -H 'accept: */*' \
  -d ''
```

A new collection box is registered with a unique identifier and an empty state by default.
No request body is required. The box is initially unassigned and ready for use.

---

3. List all collection boxes. Include information if the box is assigned (but don’t expose to what fundraising event) and if it is empty or not (but don’t expose the actual value in the box). - `GET /api/boxes`

```
curl -X 'GET' \
  'http://localhost:8080/api/boxes' \
  -H 'accept: */*'
```

This request retrieves a list of all registered collection boxes.
Each entry includes:

- Whether the box is currently **assigned** to any fundraising event (without revealing which one),
- Whether the box is **empty** or contains money (without showing the actual amounts or currencies).

This endpoint provides a high-level overview of the collection box statuses without exposing sensitive financial details.

---

4. Unregister (remove) a collection box (e.g. in case it was damaged or stolen). - `DELETE /api/boxes/{id}`

Make sure you have previously registered a collection box before attempting to delete it.

```
curl -X 'DELETE' \
  'http://localhost:8080/api/boxes/1' \
  -H 'accept: */*'
```

This request unregisters (removes) the collection box with ID `1`.
When a box is unregistered, its contents are automatically emptied, and it becomes permanently unavailable for further use.

Trying to unregister the same collection box again

```
curl -X 'DELETE' \
  'http://localhost:8080/api/boxes/1' \
  -H 'accept: */*'
```

If the same request is made a second time, the server will respond with an error indicating that the collection box with the given ID no longer exists and therefore cannot be unregistered again.

Attempting to unregister a non-existent collection box

```
curl -X 'DELETE' \
  'http://localhost:8080/api/boxes/1000' \
  -H 'accept: */*'
```

If a collection box with the specified ID (e.g. `1000`) does not exist, the API will return an appropriate error message indicating that the box could not be found.

---

5. Assign the collection box to an existing fundraising event. - `PUT /api/events/{eventId}/assign-box/{boxId}`

First, register a new collection box

Assuming previous steps have been followed and box with ID `1` was deleted, this request will create a new box, likely with ID `2`.

```
curl -X 'POST' \
  'http://localhost:8080/api/boxes' \
  -H 'accept: */*' \
  -d ''
```

Assigning an existing collection box to an existing fundraising event

```
curl -X 'PUT' \
  'http://localhost:8080/api/events/1/assign-box/2' \
  -H 'accept: */*'
```

This assigns collection box with ID `2` to fundraising event with ID `1`.

Attempt to assign an existing box to a non-existent fundraising event

```
curl -X 'PUT' \
  'http://localhost:8080/api/events/1000/assign-box/2' \
  -H 'accept: */*'
```

The server will return an error indicating that the fundraising event with ID `1000` does not exist. Assignment cannot proceed.

Attempt to assign a non-existent box to an existing fundraising event

```
curl -X 'PUT' \
  'http://localhost:8080/api/events/1/assign-box/2000' \
  -H 'accept: */*'
```

The API will respond with an error message stating that the box with ID `2000` could not be found.

Attempt to assign a non-existent box to a non-existent fundraising event

```
curl -X 'PUT' \
  'http://localhost:8080/api/events/1000/assign-box/2000' \
  -H 'accept: */*'
```

In this case, both the event and the box do not exist, so the server will indicate that neither resource is available for the operation.

---

6. Put (add) some money inside the collection box. - `PUT /api/boxes/{id}/add-money`

Successfully adding money

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/2/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "currency": "PLN",
  "amount": 10
}'
```

This request adds 10 units of currency `PLN` to the collection box with ID `2`.
The box must already be **assigned** to a fundraising event to accept funds.

Attempt to add money with an invalid currency

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/2/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "currency": "WrongCurrency",
  "amount": 10
}'
```

Returns an error indicating that the specified currency is not supported.
Only predefined currencies are accepted.

Attempt to add money without specifying currency

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/2/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "amount": 10
}'
```

The server will reject the request due to missing currency information.

Attempt to add a negative amount with a valid currency

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/2/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "currency": "PLN",
  "amount": -10
}'
```

Negative amounts are not allowed. This will result in a validation error.

Attempt to add money without specifying the amount

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/2/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "currency": "PLN",
}'
```

Malformed JSON or missing required fields (like `amount`) will result in a bad request error.

Attempt to add money with neither amount nor currency

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/2/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
}'
```

Request is invalid due to missing mandatory fields. Both `amount` and `currency` must be provided.

Attempt to add a negative amount with an invalid currency

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/2/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "currency": "WrongCurrency",
  "amount": -10
}'
```

This will trigger validation errors for both the invalid currency and the negative amount.

Attempt to add money to a non-existent box

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/1000/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "currency": "PLN",
  "amount": 10
}'
```

The server will respond that box with ID `1000` does not exist.

Attempt to add money to a box that hasn't been assigned to any fundraising event

Creating a new box that hasn’t been assigned yet:

```
curl -X 'POST' \
  'http://localhost:8080/api/boxes' \
  -H 'accept: */*' \
  -d ''
```

Assuming it gets ID `3`, trying to add money to it:

```
curl -X 'PUT' \
  'http://localhost:8080/api/boxes/3/add-money' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "currency": "PLN",
  "amount": 10
}'
```

This request will be rejected. Adding money to an **unassigned** collection box is **not permitted**, as it would violate the rule that boxes must be **empty** at the moment of assignment.

---

7. Empty the collection box i.e. “transfer” money from the box to the fundraising event’s account. - `POST /api/boxes/{id}/empty`

Successfully emptying a collection box

```
curl -X 'POST' \
  'http://localhost:8080/api/boxes/2/empty' \
  -H 'accept: */*' \
  -d ''
```

This request transfers the entire content of collection box `2` to its assigned fundraising event.
All funds are converted to the event’s account currency using fetched from NBP API exchange rates.
After the operation, the box is emptied but remains registered and assigned.

Attempt to empty a non-existent box

```
curl -X 'POST' \
  'http://localhost:8080/api/boxes/2000/empty' \
  -H 'accept: */*' \
  -d ''
```

Returns an error indicating that the collection box with ID `2000` does not exist.
Only existing and assigned boxes can be emptied.

---

8. Display a financial report with all fundraising events and the sum of their accounts. - `GET /api/events/financial-report`

Generate financial report

```
curl -X 'GET' \
  'http://localhost:8080/api/events/financial-report' \
  -H 'accept: */*'
```

This request retrieves a financial summary of all fundraising events.
Each entry in the report includes:

- The name of the fundraising event
- The total amount collected
- The currency in which the fundraising account operates