Task

Context

Create an application for managing collection boxes during fundrising 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 transfered 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 transfered 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.

REST API endpoints

The application should expose all REST endpoint defined below. For simplicity, the data input and output format can be JSON.

1. Create a new fundraising event.
2. Register a new collection box.
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).
4. Unregister (remove) a collection box (e.g. in case it was damaged or stolen).
5. Assign the collection box to an existing fundraising event.
6. Put (add) some money inside the collection box.
7. Empty the collection box i.e. "transfer" money from the box to the fundraising event's account.
8. Display a financial report with all fundraising events and the sum of their accounts.

Financial report example:

Fundraising event name	Amount	Currency
Charity One	2048.00	EUR
All for hope	512.64	GBP

Non-functional requirements

1. Crete 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.

Hints

1. Remember that correct operation of the application has a higher priority than completing all the functionality of the system. It is better to write less but well than more and with errors.
2. Think about unusual use cases and test your application.
3. Include a short instruction how to build and run the application with URL's to REST services along with sample queries.

4. Keep the database schema as simple as possible.
5. Remember about decimal points in amounts (e.g. `0.99 EUR`).
6. Try to commit regularly, so that you can trace the development of the application.

Submission form

Please submit the result of your work as a GIT repository.

Good luck!