# Usage

* Compile
* See init.sql and schema.sql
* Set up application.yml for the database and the appId and apiKey (apiKey = the actual key, appId does not matter)
* Run the application, and reach it

# Process

**As a new managerial lead,** if this code has reached me, and I have mutual trust and good relationship with the PO / scrum master / other manager:

* I would bring up the necessity of investment on code reviews, tests, continuous integration; and probably an investment on engaging people to study the latest processes / technologies.
* I would also look into the reasons why this has not been brought up by any senior members before on any of the retrospectives.
* I would also look into the development process, involving planning phase and its reviews.
* I would ask from the team how is it possible that the code has passed the code reviews, test phases, CI (and initiate those processes if not in place yet); and collaborate with them as well to figure out the possible long term solutions.

**As a lead developer** I would sit together with the junior member, compile a part of the list and ask a senior colleague to for a bit of pair programming, followed by further code reviews, to let him/her learn.

This all depends on the current targets of the company, and how much time we have for the task.

**As a senior developer** I would just compile the list for the person and probably sit together with him/her for a while.

# Actual fixes

I have registered the followings, and fixed a number of them.

The items with **+** still would need to be done, this is already over the requested 3 hours.

**breaking issues: no acceptance**

there is no configuration of database

there is no logging anywhere

**major issues: application works, for some internal guarded application it might be an acceptable temporary solution that will be fixed very quickly**

* missing sql initialisation (database? users?). —> init.sql
* 10 year old postgres reference
* constants moved to properties.
* no validation on the city parameter. Added some + a todo for further consideration
* parameters were not URL encoded
* incorrect database field (double precision for Double)
* + no documentation was available (access, parameters, etc)
* + zero tests (one was available but failing)
* + the API response is fully trusted, not validated; fields are not validated either.
* + in a microservice environment some properties would come from env variables, vault, or alike (such as api key, and password) + the DB password should be encoded, or come from vault; in case of encoding I would use jasypt

**better practices:**

* using lombok, removed the getters/setters. Also unnecessary to have jsonProperty everywhere.
* renamed application
* refactored to packages
* split the calls to several pieces
* on the restTemplate I am reading a string and converting explicitly with objectMapper. Better logging, better control.
* + restTemplate has no timeout, or proper error handling for other cases
* + the entity disclosed on the REST response should not include the internal database ID.
* + there should be a schema

**questionable:**

* + API is OK but it is not necessary to have all the annotations.
* + integration test is missing for the actual business functionality
* + separate small unit tests
* + I don’t know why use maven wrapper, and why package it in git in the first place. The version 0.4.2 is 2 years old (2nd July 2018). Since then there is also an apache maven wrapper. The developer has maven and is required to upgrade. Non-developers don’t need to run source codes, they need a pre-packaged jar.
* + there are no separate service layer under the controller. it is very lightweight application so it’s okay for now.
* + not sure why the jsonPropertyOrder is present, especially that we are only reading
* + not sure why the additionalProperties are present in the API. It seems it is our API. When we want to use some new functionality, we will add it.
* + not sure why the whole API is created instead only the subset that we want to use. Indeed if we foresee using more of the functionalities, it was the right call.
* + the database table registers all the individual requests without any metadata. I added at least the date/time.
* + The time-granularity is unclear (of the downstream). We could cache the requests/responses to a certain extent, ignore requests by time if needed (and feed from the database), etc.

**Zoltan Lusthaus**