

Stocks Specification

Jan Veen

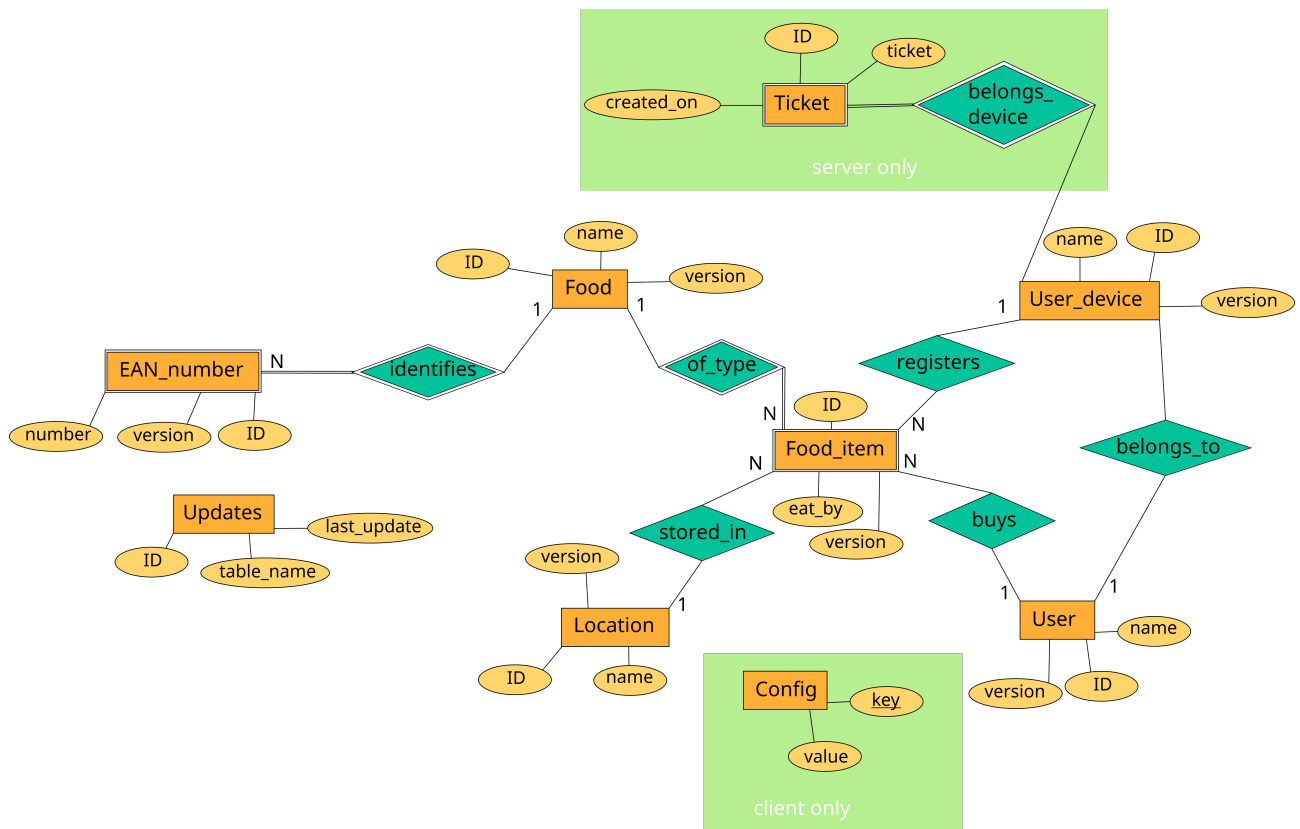
October 25, 2018

Contents

1	Data Model	2
1.1	Date formats	2
2	Architecture	3
2.1	Used Software	3
2.2	Server	3
2.3	Client	3
2.3.1	Register New Data	3
2.3.2	Refresh client database	4
2.3.3	New user registration	4
2.3.4	Device removal	6
2.4	Security	6
3	REST API	7
3.1	Common Data Types	7
3.1.1	Versions	7
3.1.2	Status codes	7
3.1.3	Timestamps	8
3.1.4	Response	8
3.2	Sentry	8
3.3	Server	8
3.3.1	Updates	8
3.3.2	Users	9
3.3.3	Devices	9
3.3.4	Food	10
3.3.5	Locations	11
3.3.6	EAN Numbers	12
3.3.7	Food items	13

Chapter 1

Data Model



1.1 Date formats

What date format is used in which part of the system:

Location	Format
Client code	Localtime
Client database	UTC
Network	UTC
Server code	UTC
Server database	UTC

Chapter 2

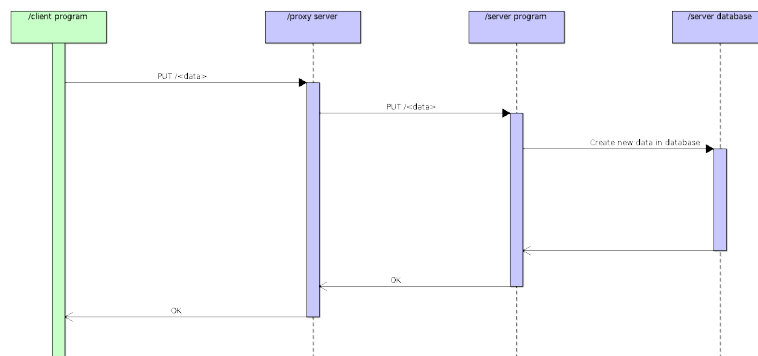
Architecture

2.1 Used Software

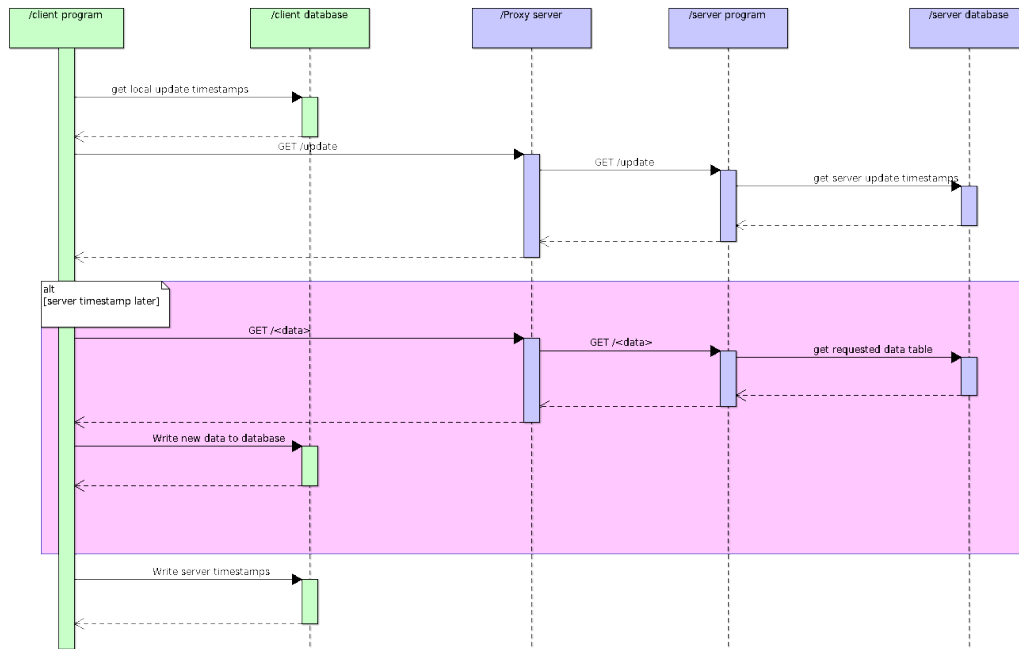
2.2 Server

2.3 Client

2.3.1 Register New Data

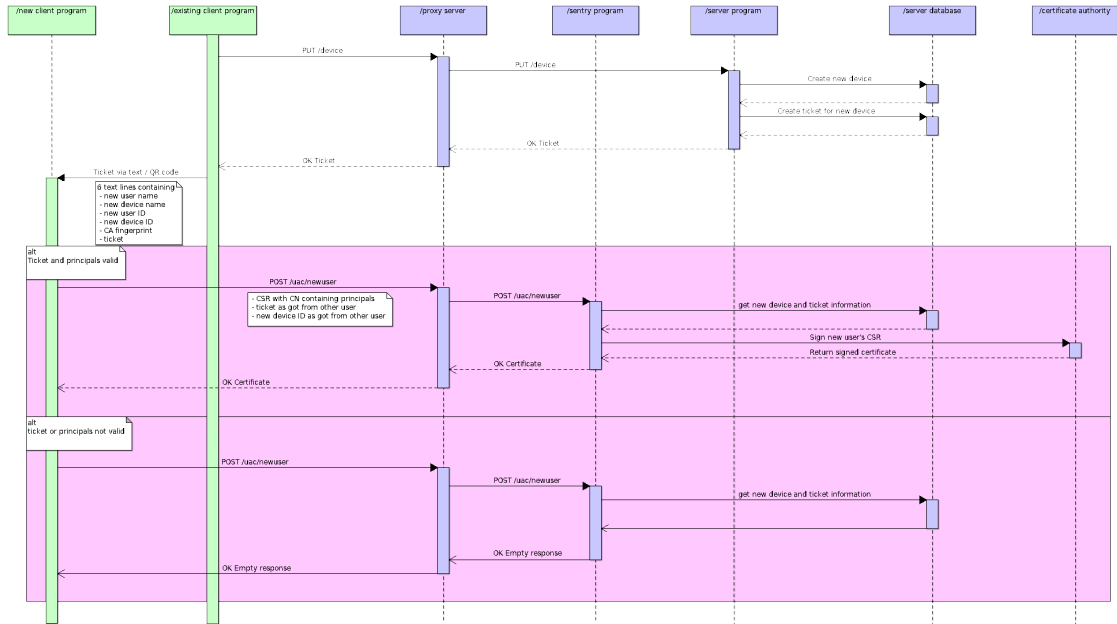


2.3.2 Refresh client database



2.3.3 New user registration

New users are always added by giving a ticket from an existing user. The details are outlined in the diagram.



Principal Names In the CSR the user stores the principals of his device. The values are formatted inside the Common Name attribute of the CSR. The pattern is `username$user_id$devicename$device_id`. So for the default test user this resolves to `John1Device$1`. The principals are checked in the sentry part of the server before the certificate is signed.

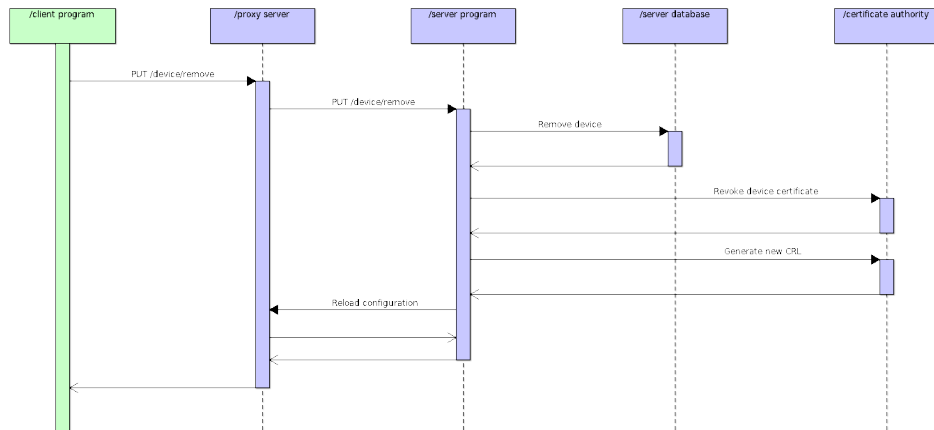
Client Verification Upon receiving a new device registration request, the sentry performs the following checks in order:

- Check if the ticket value presented by the client is found in the database
- Check the device id associated with the ticket from the database with the device id from the CSR
- Check if the remaining principals of the device match the CSR
- Check if the ticket has expired

If all the checks succeed the sentry has the CSR signed by the CA and returns it to the client.

QR Code Tickets For mobile clients it is more convenient to pass the ticket as QR code. To generate this QR code the content of the ticket has to be entered text into the QR code. The order of the values is the same as in the diagram description.

2.3.4 Device removal



2.4 Security

Chapter 3

REST API

List of all available endpoints, methods, their parameters and result types. Only v2 endpoints are listed as their usage is strongly encouraged.

Response type schemas are indicated in a pseudo JSON notation. The anonymous object atop of each description is the root object. Any further types used in that root object are described in 3.1 or directly under the root object.

3.1 Common Data Types

3.1.1 Versions

Each entity in the system has a version which is incremented each time the entity is edited. This means all modifying operations have to pass the correct version of the entity to edit. If the version does not match an error is returned using status codes, see 3.1.2 for a list of status codes to support.

3.1.2 Status codes

Numbers reporting the result status of a request. The API refers to this via the **StatusCode** type.

- 0: Success
- 1: General error
- 2: Not found
- 3: Invalid data version
- 4: Foreign key constraint violation
- 5: Database unreachable
- 6: Access denied
- 7: Invalid argument
- 8: Certificate authority unreachable

3.1.3 Timestamps

The timestamp format used is `yyyy.MM.dd-HH:mm:ss.SSS-Z`. Refer to <https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html> for semantics. The timestamps are passed as strings.

3.1.4 Response

Most calls only return a basic response with a status code. They have the following shape:

```
1 Response {
2     status: StatusCode
3 }
```

3.2 Sentry

Endpoints listed here are only reachable via the sentry port.

POST `/v2/auth/newuser`: Register new device

Form parameters:

- `device`: `int` The id of the device which wants to register
- `token`: `String` The token generated when creating the new device
- `csr`: `String` The PEM X.509 Certificate Signing Request generated by the device

Result: `application/json`

```
1 {
2     status: StatusCode
3     data: String // PEM X.509 certificate signed by the stocks CA
4 }
```

3.3 Server

3.3.1 Updates

GET `/v2/update`: Get change timestamps of entities

No parameters

Result: `application/json`

```
1 {
2     status: StatusCode
3     data: List<Update>
4 }
5
6 Update {
7     table: String
8     lastUpdate: Timestamp // yyyy.MM.dd-HH:mm:ss.SSS-Z
9 }
```

3.3.2 Users

GET /v2/user: Get all users in the system

No parameters

Result: application/json

```
1 {
2     status: StatusCode
3     data: List<User>
4 }
5
6 User {
7     id: int
8     version: int
9     name: String
10 }
```

PUT /v2/user: Add a new user into the system

Query parameters:

- name: String The name of the new user

Result: application/json, Response

DELETE /v2/user: Delete a user and all their devices from the system. The calling user is set on all food items registered by the deleted devices. Access to the system is revoked for all devices of the deleted user.

Query parameters:

- id: int The ID of the user to delete
- version: int The current version of the user to be deleted

Result: application/json, Response

3.3.3 Devices

PUT /v2/device: Create a new device in the system. This query yields both the ID of the newly created device as well as the ticket token needed to register at the sentry.

Query parameters:

- name: String The name of the new device
- belongsTo: int The user ID of the user to whom the device belongs

Result: application/json

```
1 {
2     status: StatusCode
3     data: ClientTicket
4 }
5
6 ClientTicket {
```

```

7 |     deviceId: int      // Device ID
8 |     ticket: String
9 | }

```

GET /v2/device: Get all devices in the system

No parameters

Result: application/json

```

1 | {
2 |     status: StatusCode
3 |     data: List<Device>
4 | }
5 |
6 | Device {
7 |     id: int
8 |     version: int
9 |     name: String
10 |     userId: int    // User ID
11 | }

```

DELETE /v2/device: Delete a device from the system. Access to the system is revoked for that device. The caller is set on all food items that have been registered by the deleted device.

Query parameters:

- id: int: The ID of the device to be deleted
- version: int: The version of the device to be deleted

Result: application/json, Response

3.3.4 Food

PUT /v2/food: Add a food type to the system.

Query parameters:

- name: String: The name of the new food type

Result: application/json, Response

GET /v2/food: Get all food types in the system. No parameters.

Result: application/json

```

1 | {
2 |     status: StatusCode
3 |     data: List<Food>
4 | }
5 |
6 | Food {
7 |     id: int
8 |     version: int
9 |     name: String
10 | }

```

PUT /v2/food/rename: Rename an existing food type.

Query parameters:

- **id:** `int`: The ID of the food type to rename
- **version:** `int`: The version of the food type
- **new:** `String`: The name to which the food type shall be renamed

Result: `application/json`, `Response`

DELETE /v2/food: Delete a food type. This also deletes all food items of this type.

Query parameters:

- **id:** `int`: The ID of the food type to delete
- **version:** `int`: The version of the food type to delete

Result: `application/json`, `Response`

3.3.5 Locations

PUT /v2/location: Add a new location into the system.

Query parameters:

- **name:** `String`: The name of the new location

Result: `application/json`, `Response`

GET /v2/location: Get the locations of the system.

No parameters.

Result: `application/json`

```
1 {  
2     status: StatusCode  
3     data: List<Location>  
4 }  
5  
6 Location {  
7     id: int  
8     version: int  
9     name: String  
10 }
```

PUT /v2/location/rename: Rename an existing location.

Query parameters:

- **id:** `int`: The ID of the location to rename
- **version:** `int`: The version to rename
- **newName:** `String`: The new name of the location

Result: application/json, Response

DELETE /v2/location: Delete a location.

Query parameters:

- id: int: The ID of the location to rename
- version: int: The version to rename

Result: application/json, Response

3.3.6 EAN Numbers

PUT /v2/ean: Add a new EAN code to the system.

Query parameters:

- code: String: The EAN code to register
- identifies: int: The Food type ID the code shall be associated with.

Result: application/json, Response

GET /v2/ean: Get the EAN codes of the system.

No parameters Result: application/json

```
1 {  
2     status: StatusCode  
3     data: List<EanNumber>  
4 }  
5  
6 EanNumber {  
7     id: int  
8     version: int  
9     eanCode: String  
10    identifiesFood: int    // Food ID  
11 }
```

DELETE /v2/ean: Delete an EAN Code from the system.

Query parameters:

- id: int: The ID of the EAN code to delete
- version: int: The version of the EAN code to delete

Result: application/json, Response

3.3.7 Food items

PUT /v2/fooditem: Put a new food item into the system.

Query parameters:

- eatByDate: Timestamp: The date by which the item should be consumed.
- storedIn: int: The ID of the location to store the item in
- ofType: int: The ID of the food type to which the item belongs

Result: application/json, Response

GET /v2/fooditem: Get all the food items in the system

No parameters

Result: application/json

```
1 {
2     status: StatusCode
3     data: List<FoodItem>
4 }
5
6 FoodItem {
7     id: int
8     version: int
9     eatByDate: Timestamp
10    storedIn: int           // Location ID
11    ofType: int            // Food ID
12    registers: int         // Device ID
13    buys: int              // User ID
14 }
```

PUT /v2/fooditem/edit: Edit an existing food item.

Query parameters:

- id: int: The ID of the food item to edit
- version: int: The version of the food item to edit
- eatByDate: Timestamp: The new date by which the item should be consumed
- storedIn: int: The ID of the new location to store the item in

Result: application/json, Response

DELETE /v2/fooditem: Delete a food item.

Query parameters:

- id: int: The ID of the food item to delete
- version: int: The version of the food item to delete

Result: application/json, Response