```
useEffect( effect: () => {
    fetchMyBooks()
}, deps: []);
```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4

27 to do

- [react-native] [layout] options screen
  Added by **milowedo**
- [react-native] put books icons from urls that are received
  Added by **milowedo**
- [react-native] on navigate to offers send a request with books to /calculate
  Added by **milowedo**
- [react-native] on added new book send books to database
  Added by **milowedo**
- [react-native] when accepted the BookForm, use a passed through LibraryListComponent callback to add a book and send it to the database
  ?? send whole collection
  Added by **milowedo**

1 in progress

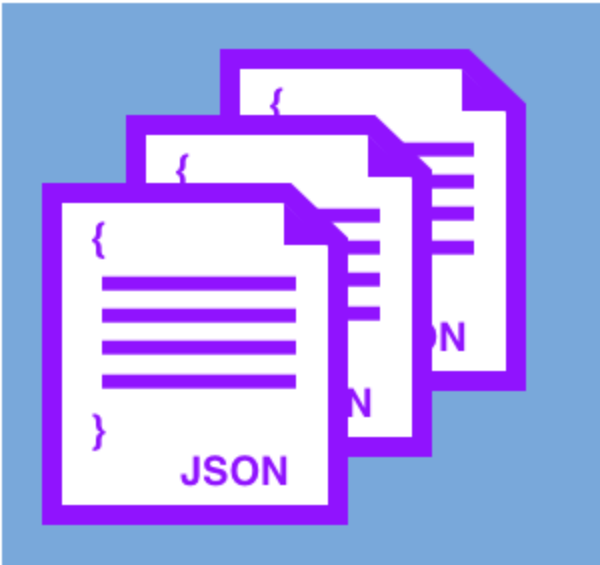
- [react-native] icons to corelate with actions
  settings return
  settings link
  lib, wanted, offers
  Added by **milowedo**

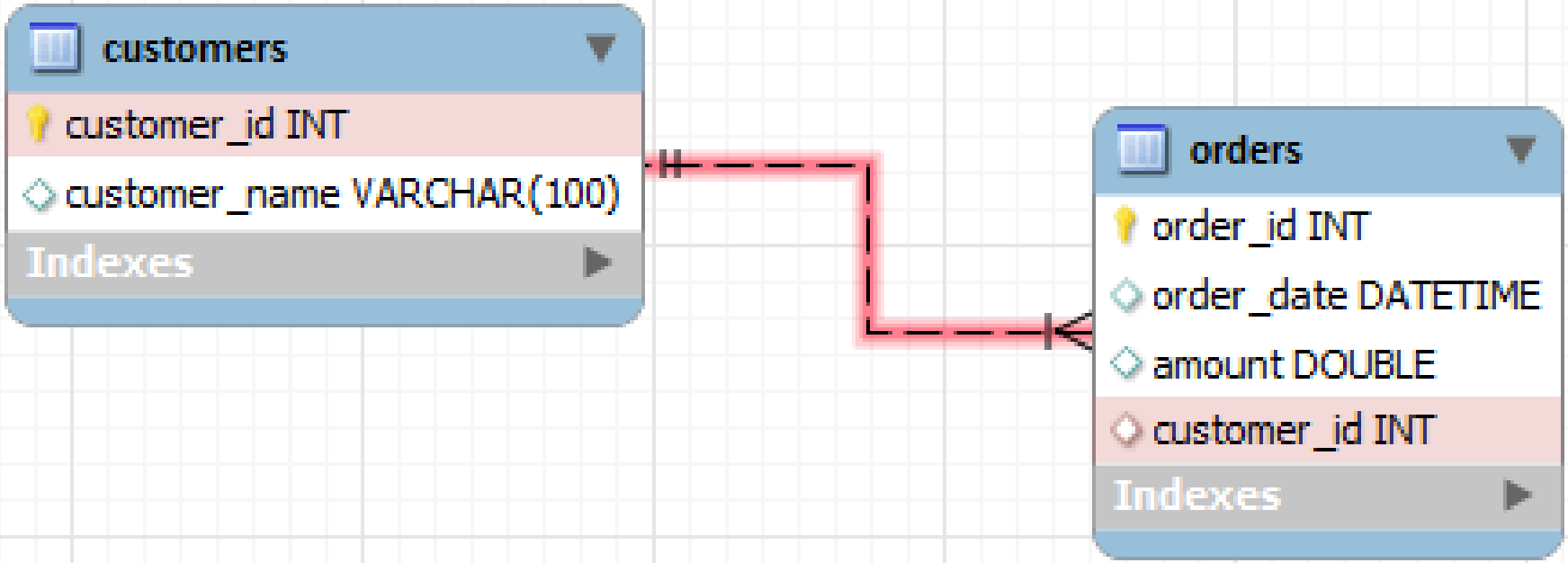
33 done

- [react-native] adding new book to libraries
  ?? some thin element at the top with a button?
  ?? on click to have a sample data in placeholder
  ?? have defeault max price to 8
  Added by **milowedo**
- [gateway] add an endpoint that takes a list of books and returns auctions
  Should call the offers fetcher service with those books and authorization token ready to use
  Added by **milowedo**
- [offers] calculate result and send back in a response
  Added by **milowedo**
- [java-dev] OffersFetcher implementation
  Handle endpoint that takes list of

```
mongoose.connect(connString, options: {
    useNewUrlParser: true,
    useCreateIndex: true,
    useUnifiedTopology: true
}).then();
mongoose.connection.on( event: 'connected', listener: () => {
    console.info( message: 'Connected to mongo instance')
});
```

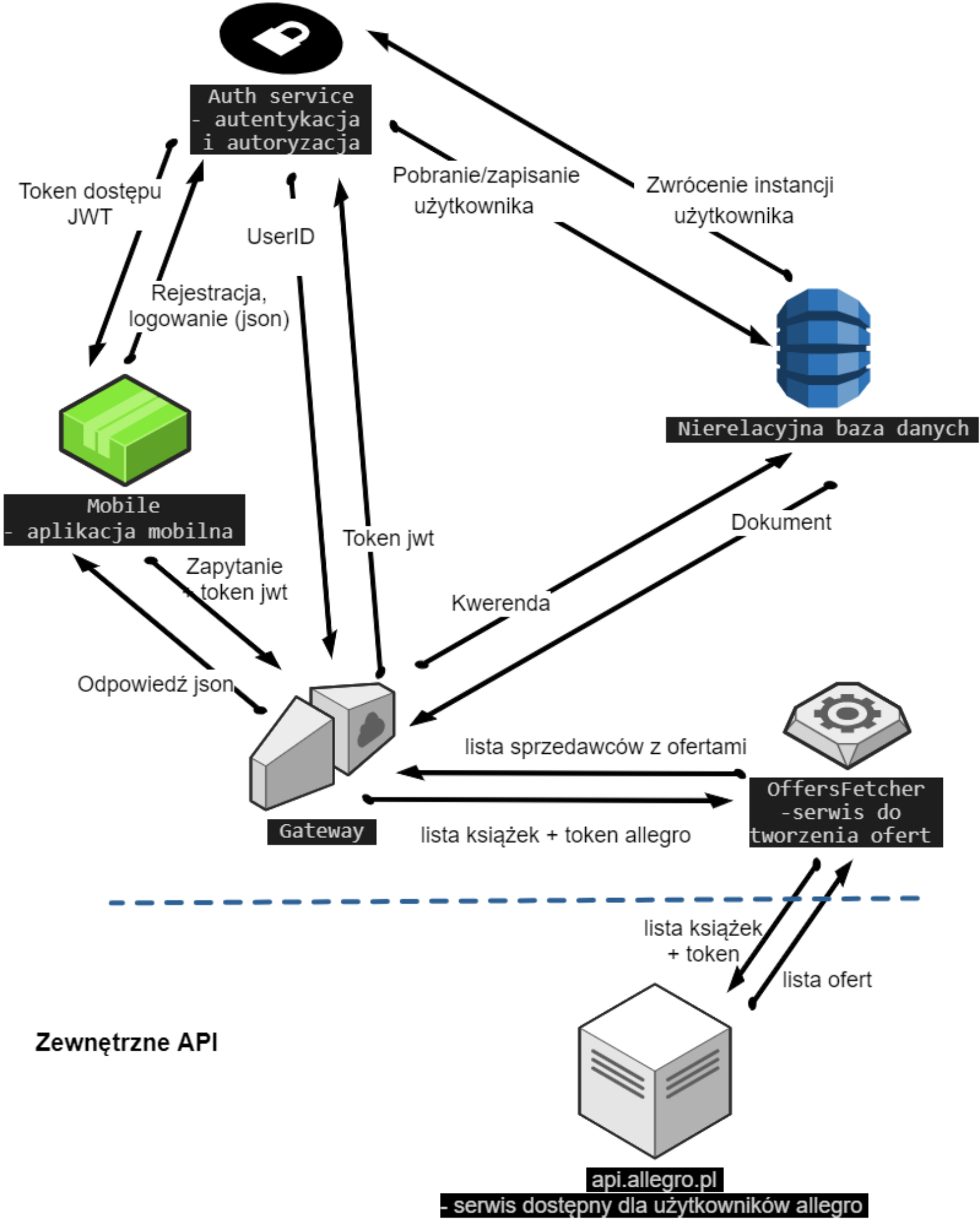
```
_id: ObjectId("5e0475c53c89d7ceca573698")
userID: "5de6be0eb33c1c0024070c49"
__v: 0
> wanted: Array
v library: Array
  v 0: Object
      _id: "0"
      writer: "Kurt Vonnegut"
      title: "Slaughterhouse no 5"
  > 1: Object
  > 2: Object
  v 3: Object
      _id: "3"
      writer: "Jerome K. Jerome"
      title: "Trzech panów w łódce (nie licząc psa)"
  > 4: Object
  > 5: Object
  > 6: Object
  v 7: Object
      _id: "7"
      writer: "Franz Kafka"
      title: "Proces"
  v 8: Object
      _id: "8"
      writer: "Niccolò Machiavelli"
      title: "Książe"
  > 9: Object
  > 10: Object
  > 11: Object
  > 12: Object
  > 13: Object
```



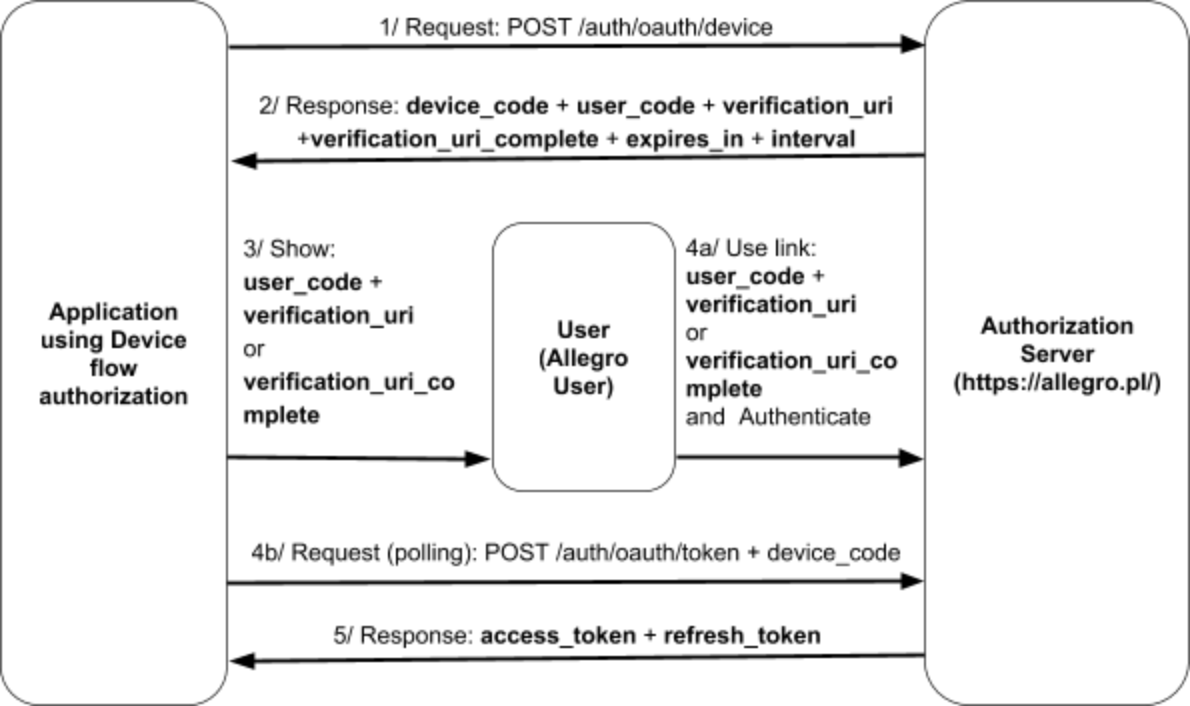

customers
customer_id INT
customer_name VARCHAR(100)
Indexes
orders
order_id INT
order_date DATETIME
amount DOUBLE
customer_id INT
Indexes

```java
public static CompletableFuture<Void> addOffersForBook(BookEntityReceived bookEntityReceived,
                                                       ConcurrentHashMap<Seller, HashSet<BookResult>> calculatedResult) {
    return callApi(bookEntityReceived).thenCompose(AllegroRequestHandler::parseResponseToOffers)
            .thenApplyAsync(books ->
                    IntStream.range(0, books.size() - 1)
                            .mapToObj(books::get)
                            .collect(Collectors.toList())
            ).thenAcceptAsync(list ->
                    list.parallelStream().map(JsonElement::getAsJsonObject).forEach(singleBook -> {
                        var seller = extractSellerFromJson(singleBook);
                        var newBook = extractBookResultFromJson(singleBook);
                        if (seller == null || newBook == null) {
                            AllegroRequestHandler.logger.info("Could not create book or seller from json offer object");
                            return;
                        }
                        newBook.setBookTitle(bookEntityReceived.title);
                        newBook.setWriter(bookEntityReceived.writer);
                        calculatedResult.putIfAbsent(seller, new HashSet<>());
                        calculatedResult.get(seller).add(newBook);
                    }));
}
```

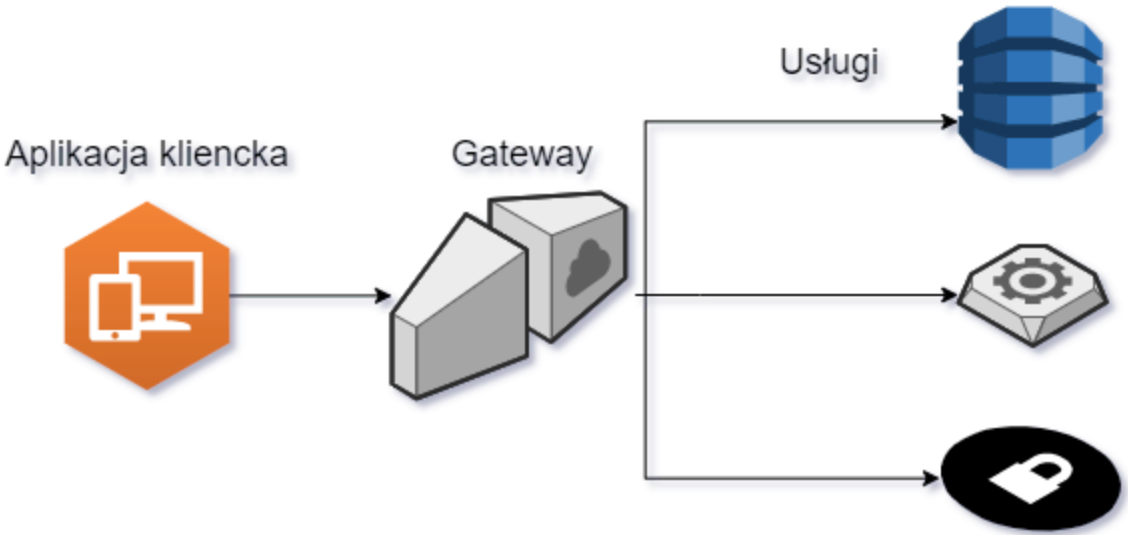
```java
public static CompletableFuture<Void> addOffersForBook(BookEntityReceived bookEntityReceived,
                                                       ConcurrentHashMap<Seller, HashSet<BookResult>> calculatedResult) {
    return callApi(bookEntityReceived).thenCompose(AllegroRequestHandler::parseResponseToOffers)
            .thenApplyAsync(books ->
                    IntStream.range(0, books.size() - 1)
                            .mapToObj(books::get)
                            .collect(Collectors.toList())
            ).thenAcceptAsync(list ->
                    list.parallelStream().map(JsonElement::getAsJsonObject).forEach(singleBook -> {
                        var seller = extractSellerFromJson(singleBook);
                        var newBook = extractBookResultFromJson(singleBook);
                        if (seller == null || newBook == null) {
                            AllegroRequestHandler.logger.info("Could not create book or seller from json offer object");
                            return;
                        }
                        newBook.setBookTitle(bookEntityReceived.title);
                        newBook.setWriter(bookEntityReceived.writer);
                        calculatedResult.putIfAbsent(seller, new HashSet<>());
                        calculatedResult.get(seller).add(newBook);
                    }));
}
```


Auth service
- autentykacja
i autoryzacja
Token dostępu
JWT
Rejestracja,
logowanie (json)
UserID
Pobranie/zapisanie
użytkownika
Zwrócenie instancji
użytkownika
Nierelacyjna baza danych
Mobile
- aplikacja mobilna
Token jwt
Zapytanie
+ token jwt
Kwerenda
Dokument
Odpowiedź json
Gateway
lista sprzedawców z ofertami
lista książek + token allegro
OffersFetcher
-serwis do
tworzenia ofert
lista książek
+ token
lista ofert
Zewnętrzne API
api.allegro.pl
- serwis dostępny dla użytkowników allegro

```
function authorizationBeat(url :string  = 'https://api.allegro.pl/sale/categories/') {
    requests({
        url: url,
        headers: {'authorization': `Bearer ${properties.get('access_token')}`}
    }, (error, response, body) => {
        if (response.statusCode === 200) {
            console.log("User is authorized.");
            return body;
        }
        console.log("Access token is not valid, user not authorized.");
        if (response.statusCode === 401) {
            let refresh_token_acquired_time = properties.get('refresh_token_time');
            let refresh_token = properties.get('refresh_token');
            console.log(`Refresh token: ${refresh_token}`);
            if (refresh_token === "undefined") {
                console.log("Refresh token has never been acquired.");
                acquireLinkForAuthorization();
            } else if ((Date.now() - refresh_token_acquired_time) > (30 * 24 * 60 * 60 * 1000)) {
                console.log(`Refresh token is outdated: ${refresh_token_acquired_time}`);
                acquireLinkForAuthorization()
            } else {
                console.log("Refresh token is valid, sending refreshing request.");
                refreshTheToken();
            }
        }
    })
}
```


1/ Request: POST /auth/oauth/device
2/ Response: device_code + user_code + verification_uri +verification_uri_complete + expires_in + interval
Application using Device flow authorization
3/ Show: user_code + verification_uri or verification_uri_complete
User (Allegro User)
4a/ Use link: user_code + verification_uri or verification_uri_complete and Authenticate
Authorization Server (https://allegro.pl/)
4b/ Request (polling): POST /auth/oauth/token + device_code
5/ Response: access_token + refresh_token

```
const express = require('express');
const app = express();
app.get('/beat', (req, res) => res.status(200).send( body: 'auth service is up'));
app.listen(
    port: process.env.PORT || port,
    callback: () => console.info( message: `App is listening on ` + (process.env.PORT || port) + '.')
);
```


Usługi
Aplikacja kliencka
Gateway