



# Sviluppo Applicazione

## **Skupply**

Versione 1.5  
6 Febbraio 2023

**Di Zepp Dorjan**  
[dorijan.dizepp@studenti.unitn.it](mailto:dorijan.dizepp@studenti.unitn.it)

**Piccin Andrea**  
[andrea.piccin@studenti.unitn.it](mailto:andrea.piccin@studenti.unitn.it)

**Rossi Simone**  
[simone.rossi-2@studenti.unitn.it](mailto:simone.rossi-2@studenti.unitn.it)





## Indice

1. Scopo del documento
2. User flows
3. Project Structure
4. Project Dependencies
5. Project data & Database
6. Project APIs
  - 6.1. Resources Extraction from Class Diagram
  - 6.2. Resource Models
7. Sviluppo API
8. Documentazione API
9. Implementazione frontend
10. Github Repository
11. Deployment
12. Testing

## 1. Scopo del documento

Il presente documento riporta tutte le informazioni riguardanti la fase produttiva dell'applicazione Skupply.

Partendo dalla descrizione degli user flow e continuando nelle varie fasi realizzative come l'implementazione delle funzioni di backend e frontend e la realizzazione delle API, il documento offre una vista su tutte le funzionalità principali della piattaforma.

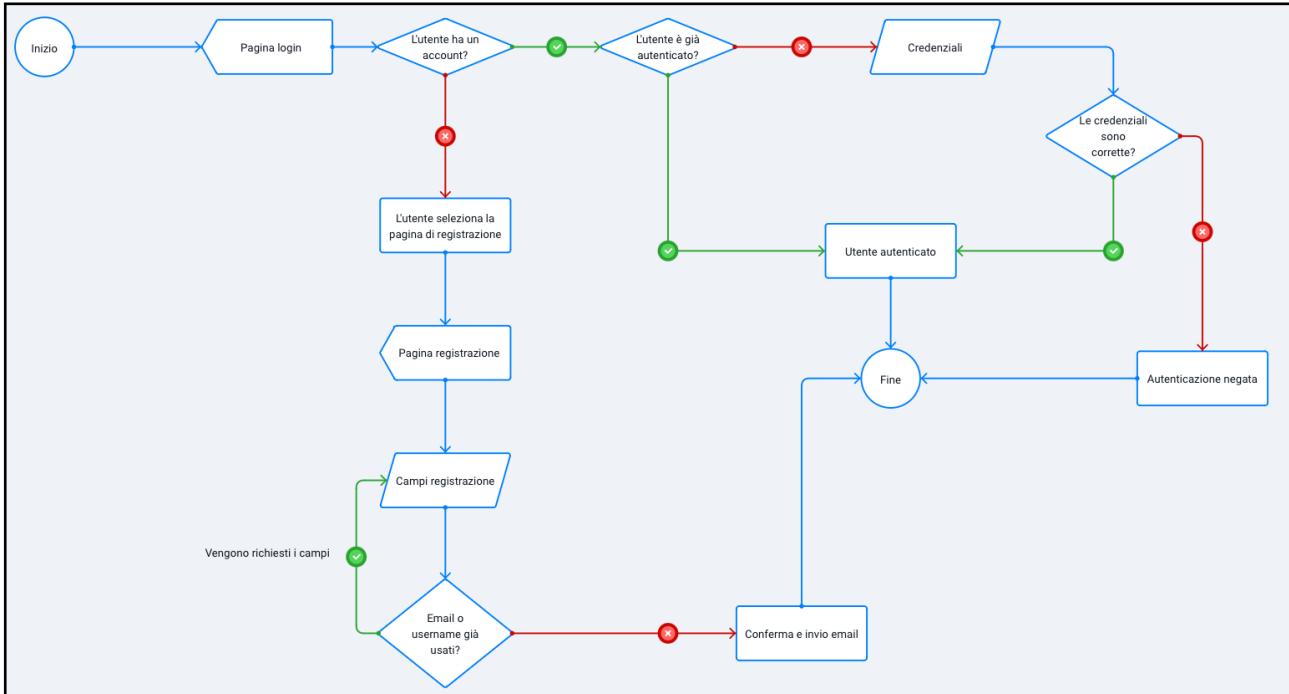
Per ogni API realizzata, oltre ad una descrizione delle funzionalità fornite, si presenta la sua documentazione e i test effettuati.

Infine, una sezione è dedicata alle informazioni delle Git Repository e al deployment dell'applicazione.

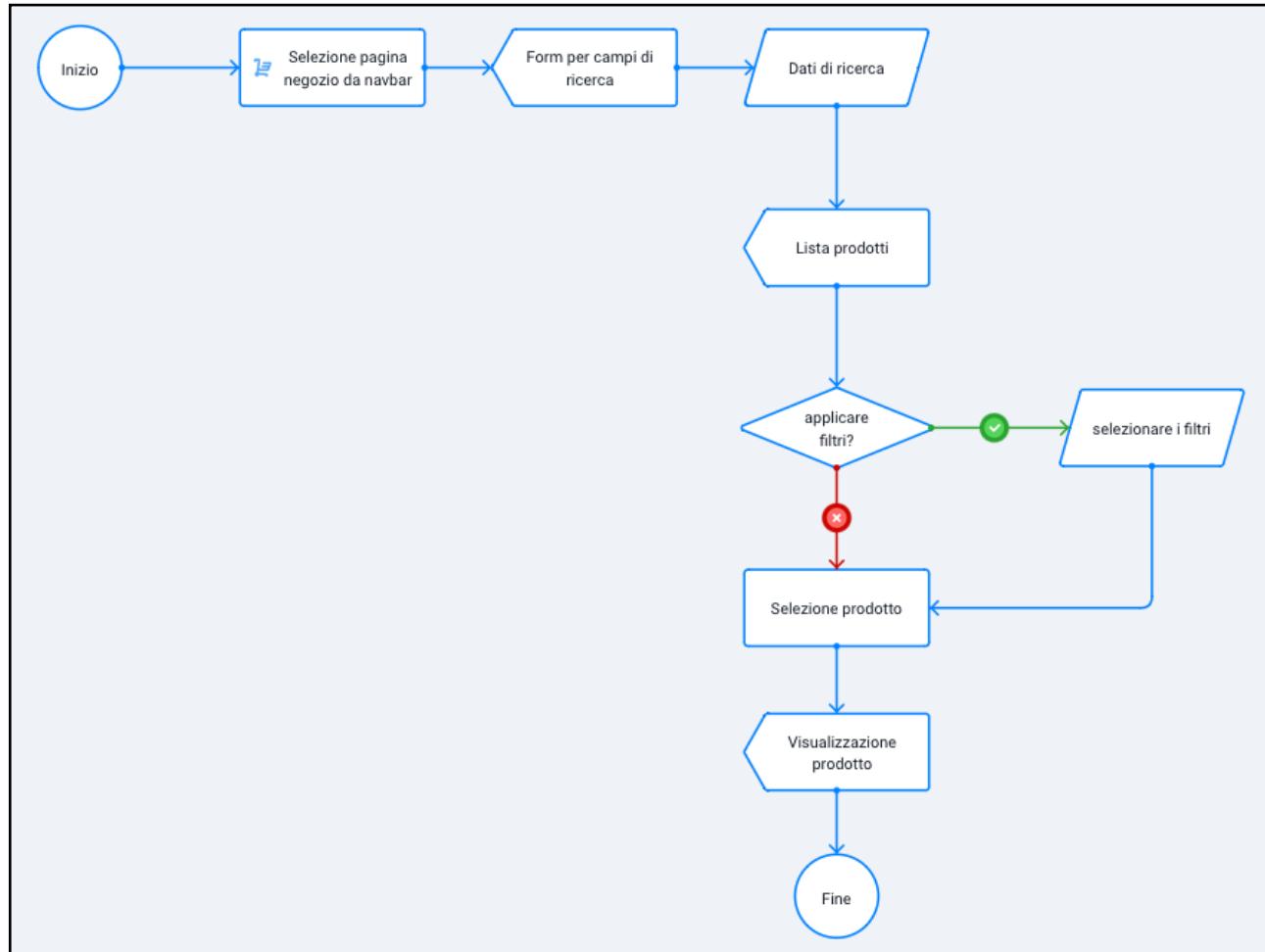
## 2. User Flows

In questa sezione del documento di sviluppo riportiamo gli “user flow” per le fasi di:

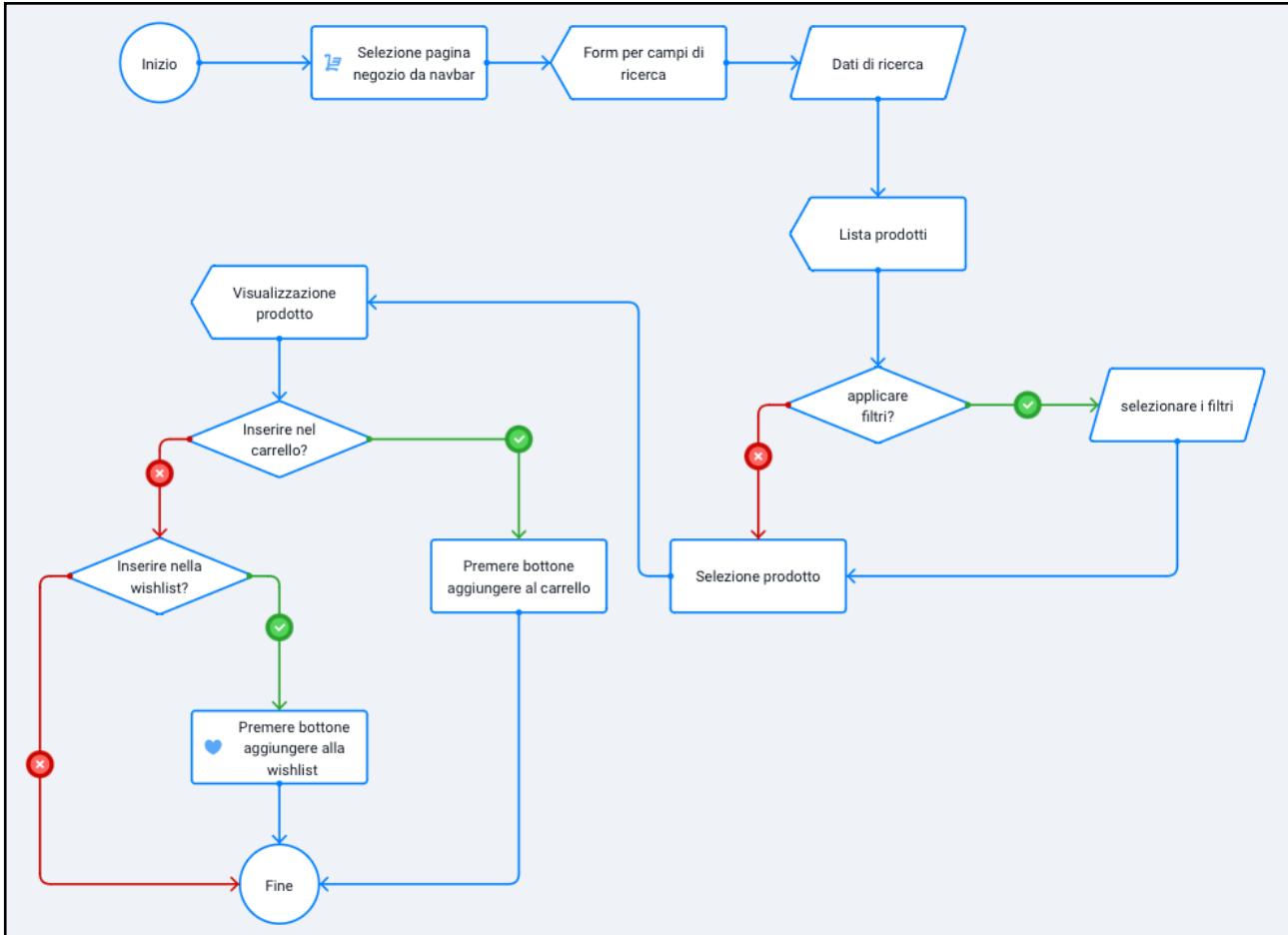
### 1. Autenticazione



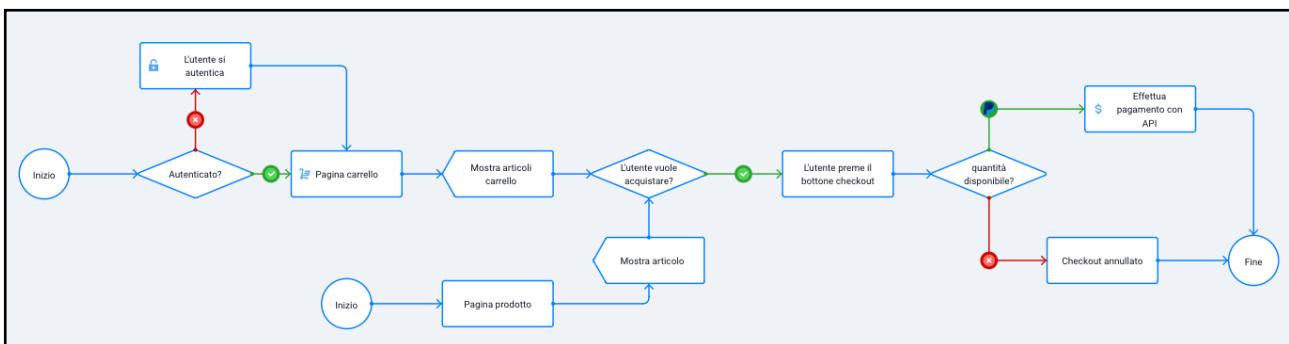
## 2. Ricerca di un prodotto



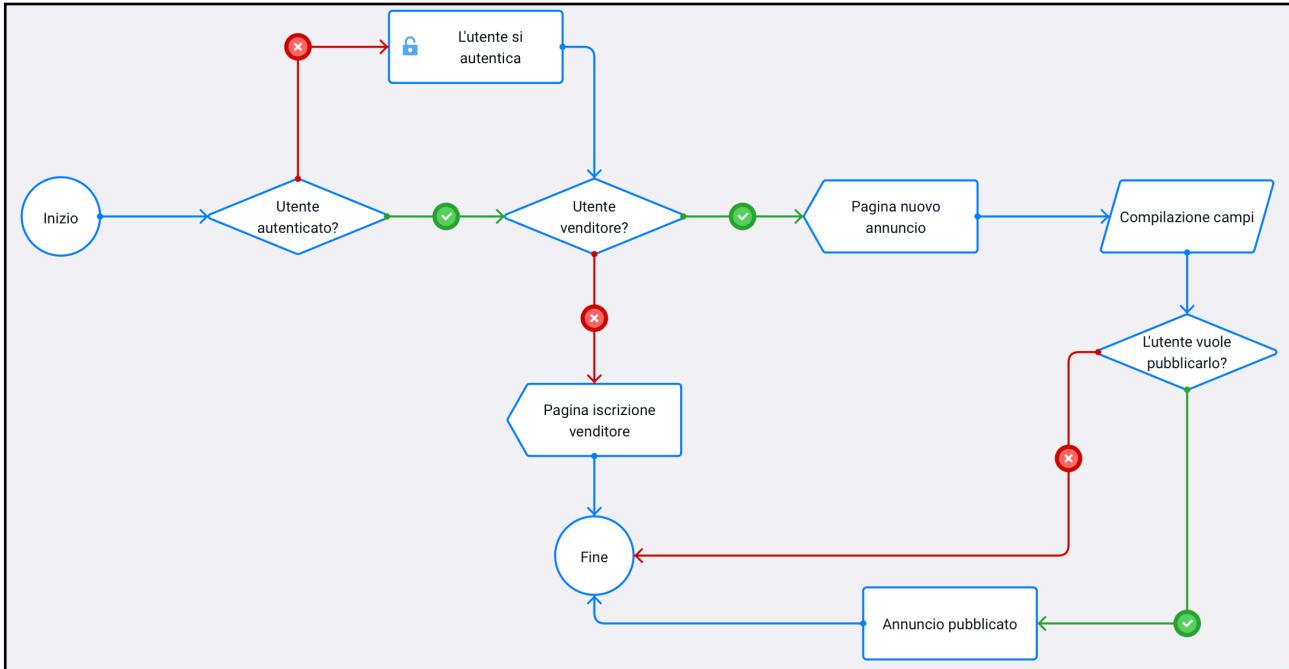
### 3. Aggiunta di un prodotto al carrello e/o alla wishlist



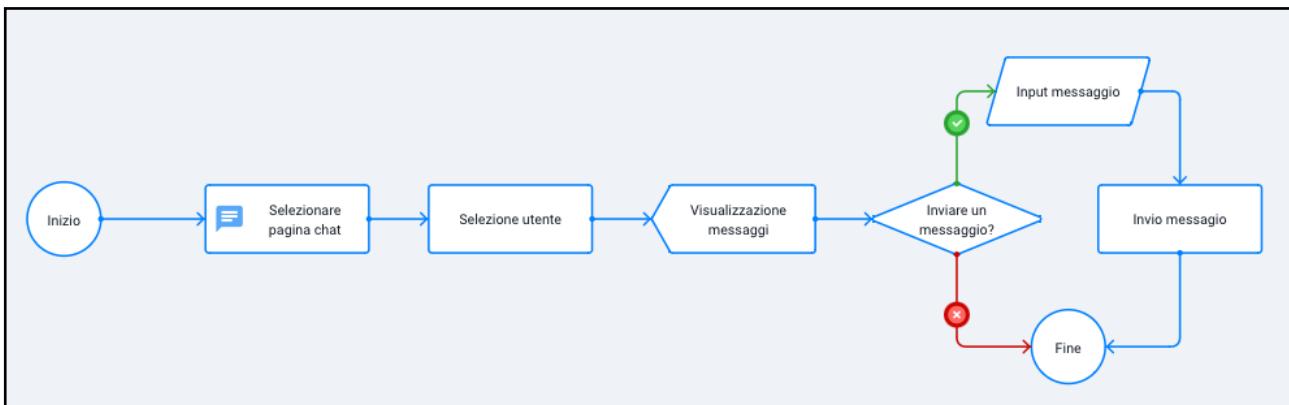
### 4. Pagamento di uno o più articoli



## 5. Creazione di un annuncio



## 6. Gestione della chat



## 3. Project Structure

L'applicazione è stata sviluppata separando completamente la parte di backend da quella di frontend utilizzando NodeJS in combinazione con VueJS.

Per lo storage ci siamo appoggiati al servizio di MongoDB.

Il backend si sviluppa così:

```
src
  └── index.js           Eseguibile per l'inizializzazione del server
  └── controllers
      ├── buyer.js
      ├── cart.js
      ├── chat.js
      ├── email.js
      ├── item.js
      ├── login.js
      ├── order.js
      ├── proposal.js
      ├── review.js
      ├── search.js
      ├── seller.js
      └── wishlist.js
  └── models             Manipolazione dei dati e gestione database
      ├── Buyer.js
      ├── Category.js
      ├── Chat.js
      ├── Item.js
      ├── Message.js
      ├── Order.js
      ├── Proposal.js
      ├── Review.js
      ├── Seller.js
      └── Shipment.js
  └── routes              Pubblicazione degli endpoint e specifica delle API
      ├── buyer.js
      ├── cart.js
      ├── chat.js
      ├── email.js
      ├── item.js
      ├── login.js
      ├── order.js
      ├── proposal.js
      ├── review.js
      ├── search.js
      ├── seller.js
      └── wishlist.js
  └── swagger            Documentazione delle API
      ├── components
      ├── index.yaml
      ├── paths
      └── swagger.json
  └── test                Test delle funzionalità del backend
      ├── cart.test.js
      ├── chat.test.js
      ├── email.test.js
      ├── item.test.js
      ├── login.test.js
      ├── order.test.js
      ├── review.test.js
      └── search.test.js
  └── utils               Funzionalità di appoggio
      ├── address.js
      ├── auth.js
      └── email.js
```

Il frontend invece presenta la seguente struttura:

```

src
├── App.vue
├── assets
│   ├── images
│   ├── main.css
│   └── theme.js
└── components
    ├── Address.vue
    ├── ContactCard.vue
    ├── Contacts.vue
    ├── DraftCard.vue
    ├── EditModal.vue
    ├── Footer.vue
    ├── FooterLink.vue
    ├── HomeCard.vue
    ├── LoginForm.vue
    ├── MarketCard.vue
    ├── Messages.vue
    ├── NavBar.vue
    ├── NavLink.vue
    ├── OfferCard.vue
    ├── OrderCard.vue
    ├── Payments.vue
    ├── PaymentsPopup.vue
    ├── ProductCard.vue
    ├── Proposal.vue
    ├── Review.vue
    ├── ReviewCard.vue
    ├── ShipModal.vue
    ├── SignupForm0.vue
    ├── SignupForm1.vue
    ├── SignupForm2.vue
    ├── icons
    └── images
├── json
│   └── faq.json
├── main.js
├── router
│   └── index.js
└── stores
    ├── server.js
    └── user.js
└── views
    ├── Cart.vue
    ├── Chat.vue
    ├── HelpFaq.vue
    ├── Home.vue
    ├── Login.vue
    ├── Market.vue
    ├── NotFound.vue
    ├── NotVerified.vue
    ├── Product.vue
    ├── Profile.vue
    ├── Search.vue
    ├── Sell.vue
    ├── Signup.vue
    ├── Terms.vue
    ├── VendorPrivate.vue
    ├── VendorPubblic.vue
    ├── Verify.vue
    └── Wishlist.vue

```

Gestione dei font

Contiene i componenti grafici utilizzati nelle View

File di inizializzazione

Gestione degli endpoint delle API

Gestione della sessione

Interfacce grafiche dell'applicazione

## 4. Project Dependencies

Nella realizzazione del frontend sono stati utilizzati i seguenti moduli di Node:

- **Geoapify** (`geocoder-autocomplete@1.5.1`): validazione e autocompletamento indirizzi
- **Paypal** (`paypal-js@5.1.4`): emulazione pagamenti di checkout
- **Express** (`express@4.18.2`): gestione del routing tra le pagine
- **Moment** (`moment@2.29.4`): formattazione e gestione date
- **Naive UI** (`naive-ui@2.33.5`): component library per lo sviluppo delle interfacce
- **Pinia** (`pinia@2.0.24`): gestione dei dati di sessione
- **Vite** (`vite@3.2.4`): JS development server
- **Vue** (`vue@3.2.45`): framework JS per il frontend

Mentre per il backend:

- **Deep email validator** (`deep-email-validator@0.1.21`): validazione email
- **Dotenv** (`dotenv@16.0.3`): gestione variabili d'ambiente
- **Express** (`express@4.18.2`): gestione API
- **Jest** (`jest@29.3.1`): testing
- **JWT - Json Web Token** (`jsonwebtoken@9.0.0`): token per l'autenticazione
- **Mongoose** (`mongoose@6.6.5`): interfaccia verso MongoDB
- **Multer** (`multer@1.4.5-lts.1`): middleware per l'upload di file (foto nel nostro caso)
- **Node mailer** (`nodemailer@6.8.0`): per l'invio di email
- **Supertest** (`supertest@6.3.3`): testing
- **Swagger UI** (`swagger-ui-express@4.6.0`): per la documentazione delle API



## 5. Project Data & Database

Per la rappresentazione dei dati dell'applicazione abbiamo definito i seguenti modelli (e conseguenti collections):

- Buyer

1	_id: ObjectId('639f33fdff7dbe641b8fd26b')	ObjectId
2	firstname: "Andrea"	String
3	lastname: "Piccin"	String
4	username: "ap39"	String
5	email: "andrea@skupply.shop"	String
6	passwordHash: "571c792ce2d3c108d3b01316093ee28d4628d364e69d6766261039cd32ed295f"	String
7	> addresses: Array	Array
8	> phone: Object	Object
9	isTerms: true	Boolean
10	isVerified: true	Boolean
11	verificationCode: "01fd525389e06ebe98302a6a8bbalef448a16972d15ddeb71560e8ac1a4d9f02"	String
12	> proposals: Array	Array
13	isSeller: true	Boolean
14	sellerId: 63a30cba145a77e810f1277b	ObjectId
15	creationDate: 2022-12-18T15:38:37.927+00:00	Date
16	> cart: Array	Array
17	> wishlist: Array	Array
18	__v: 16	Int32
19	> chats: Array	Array

- Category

1	_id: ObjectId('638e1d66f5259e7e1a031e45')	ObjectId
2	title: "università"	String
3	description: "Articoli scolastici universitari"	String

- Chat

1	_id: ObjectId('63930ab3e5261e6911adaf99')	ObjectId
2	> user1: Object	Object
3	> user2: Object	Object
4	> messages: Array	Array
5	__v: 10	Int32

- Message

1	_id: ObjectId('63930ccb195fe86812afe5e1')	ObjectId
2	> sender: Object	Object
3	text: "testo libero"	String
4	date: 2022-12-09T10:23:22.398+00:00	Date
5	__v: 0	Int32



- Item

```

1  _id: ObjectId('63a032fef94e87dabd0bc3f8') ObjectId
2  title: "Zaino eastpak" String
3  description: "Zaino marca eastpak colore nero" String
4  ownerId: 63a30cba145a77e810f1277b ObjectId
5  quantity: 1 Int32
6  > categories: Array Array
7  > photos: Array Array
8  conditions: "LIKE_NEW" String
9  price: 20 Decimal128
10 city: "Pergine" String
11 state: "PUBLISHED" String
12 pickUpAvail: false Boolean
13 shipmentAvail: true Boolean
14 shipmentCost: 0 Decimal128
15 __v: 0 Int32

```

- Order

```

1  _id: ObjectId('63dd052ec562739977311245') ObjectId
2  buyer: 63a1bec10aa2f57867912bd9 ObjectId
3  seller: 639f6b399b38c1bfc9633360 ObjectId
4  < article: Object Object
5    id: 63a034adb93b4039ab376a6c ObjectId
6    quantity: 1 Int32
7    price: 10 Decimal128
8    shipment: 0 Decimal128
9    state: "COMPLETED" String
10   reviewed: true Boolean
11   payment: "SENT" String
12   date: 2023-02-03T12:59:26.897+00:00 Date
13   __v: 0 Int32
14   courier: "GLS" String
15   trackingCode: "codiceProva" String

```

- Proposal

```

1  _id: ObjectId('63d5402eff2e096eb4299557') ObjectId
2  itemId: 63a034fd599f3e730eb65755 ObjectId
3  authorId: 63b46b70e445484129c2656d ObjectId
4  state: "ACCEPTED" String
5  price: 1 Decimal128
6  __v: 0 Int32

```

- Review

```
1 _id: ObjectId('63da4feaf3d95d6f44dc9347') ObjectId
2 authorId: 63a1bec10aa2f57867912bd9 ObjectId
3 sellerId: 639f6b399b38c1bfc9633360 ObjectId
4 title: "Ottimo venditore"
5 description: null String
6 rating: 3 Null
7 __v: 0 Int32
```

- Seller

```
1 _id: ObjectId('639f6b399b38c1bfc9633360') ObjectId
2 userId: 63a96e8a0859356907ff72045 ObjectId
3 > items: Array Array
4 > reviews: Array Array
5 > proposals: Array Array
6 __v: 49 Int32
```

## 6. Project APIs

All'interno di questa sezione si individuano due sotto sezioni: l'estrazione delle risorse dal modello delle classi e i resource model.

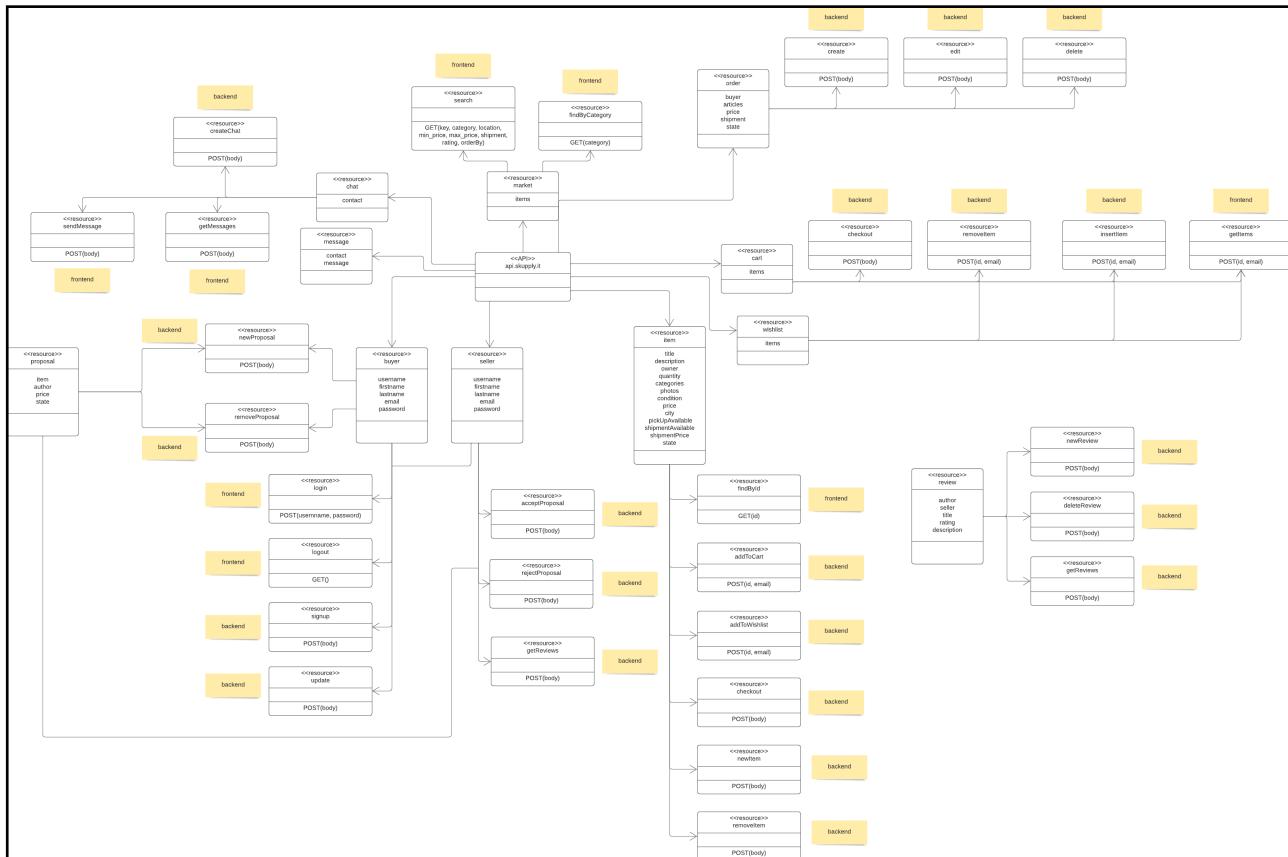
### 6.1. Resource Extraction

Nel seguente diagramma vengono visualizzate le risorse estratte dal diagramma delle classi descritto nel D3 e se le funzioni hanno un applicazione nel frontend o nel backend.

Qui vengono elencate le api che l'utente può utilizzare all'interno della piattaforma.

Premesso che:

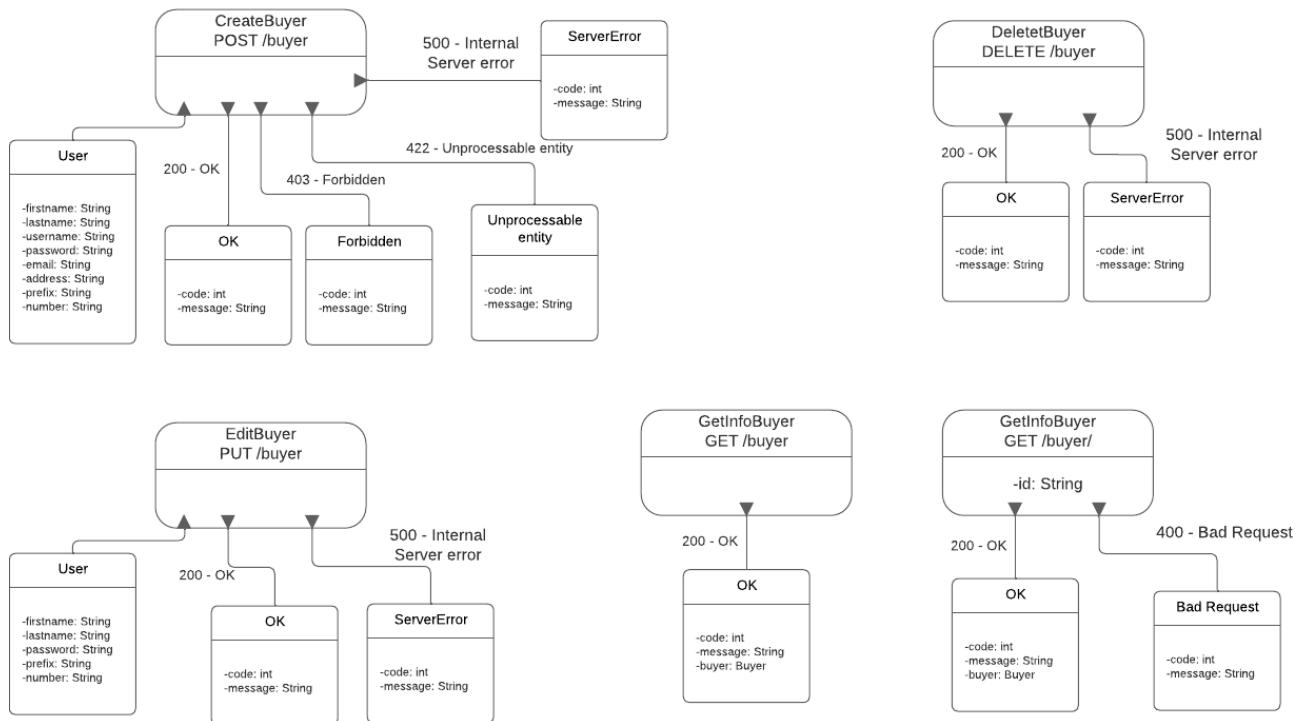
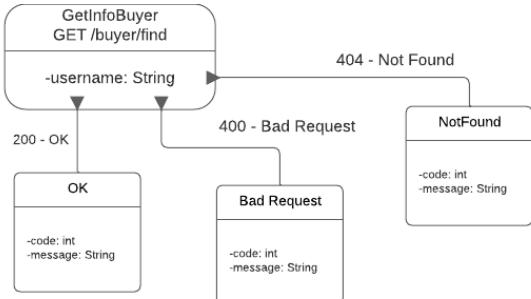
- Per le entità cart e wishlist vengono utilizzate per entrambe le stesse funzioni come definito nel diagramma delle classi.
- Per l'entità market, che opera su una lista di elementi (gli annunci), vengono definite le funzioni di ricerca disponibili.
- Come specificato nel D3, l'utente viene diviso in due possibili entità, buyer e seller e anche per loro sono state definite delle api.



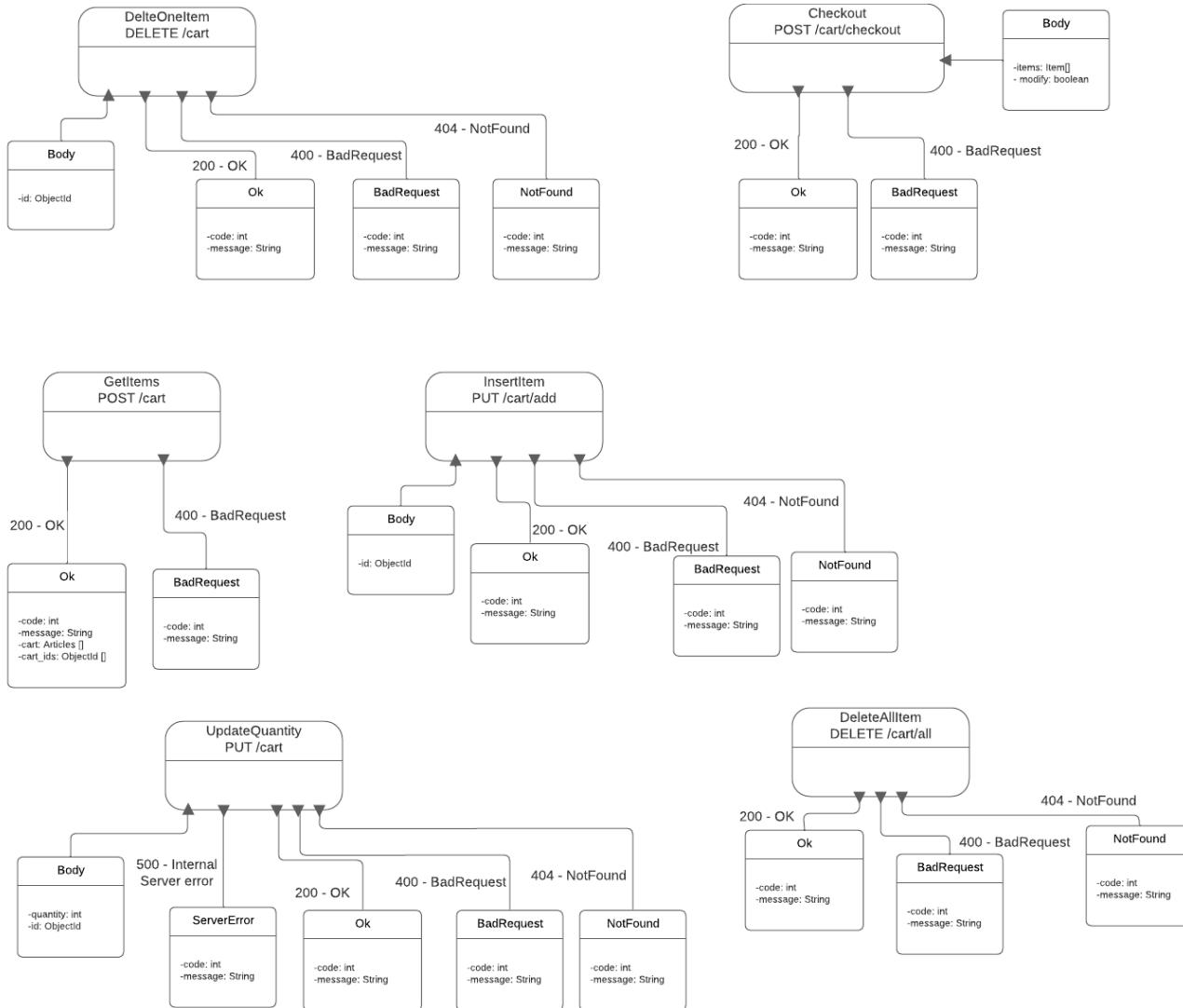
## 6.2. Resource Model

Il resource model presenta lo sviluppo delle API tramite diagrammi costruiti sulla base della resource extraction, di seguito saranno illustrate le API suddivise per risorsa.

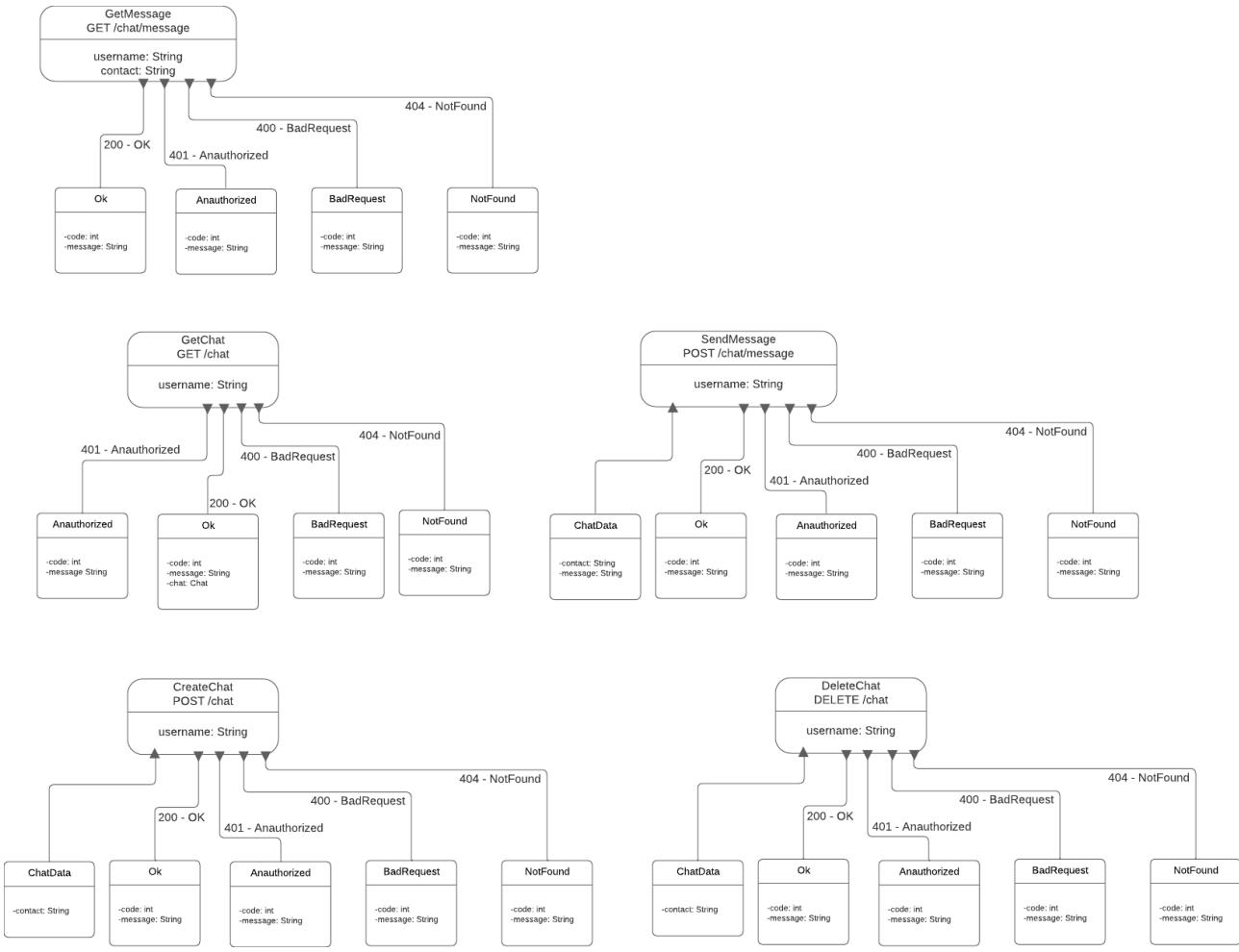
- Buyer



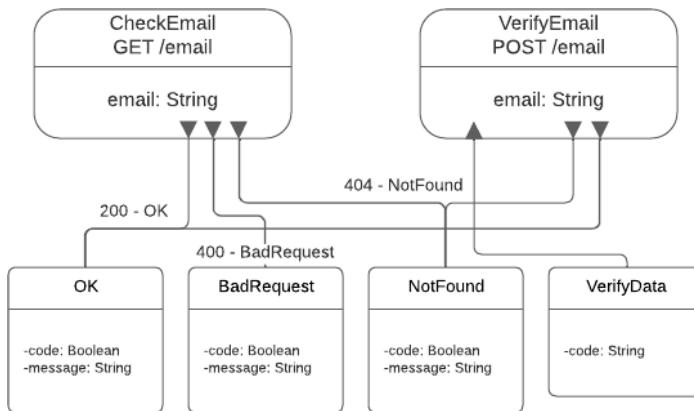
- Cart



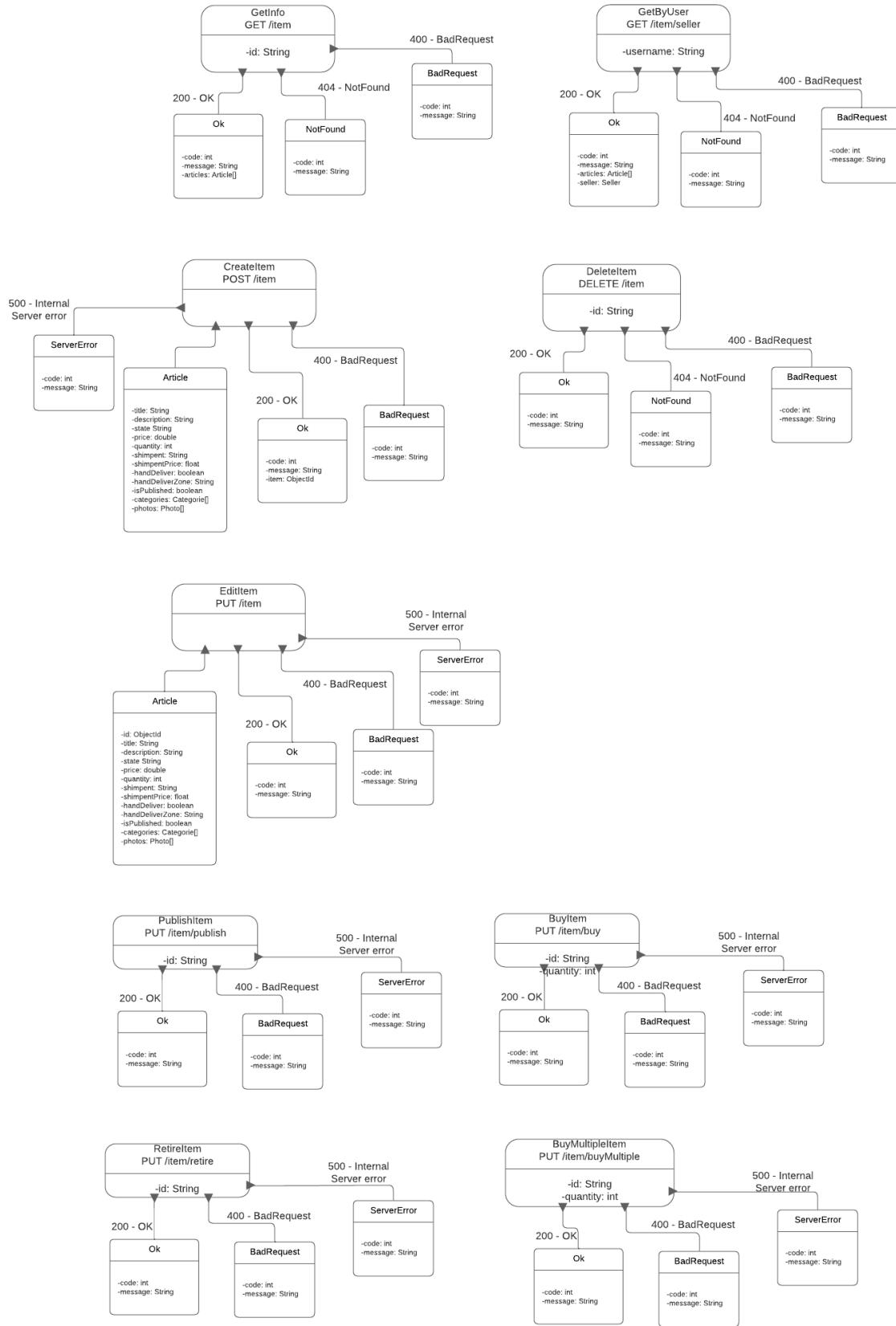
- Chat



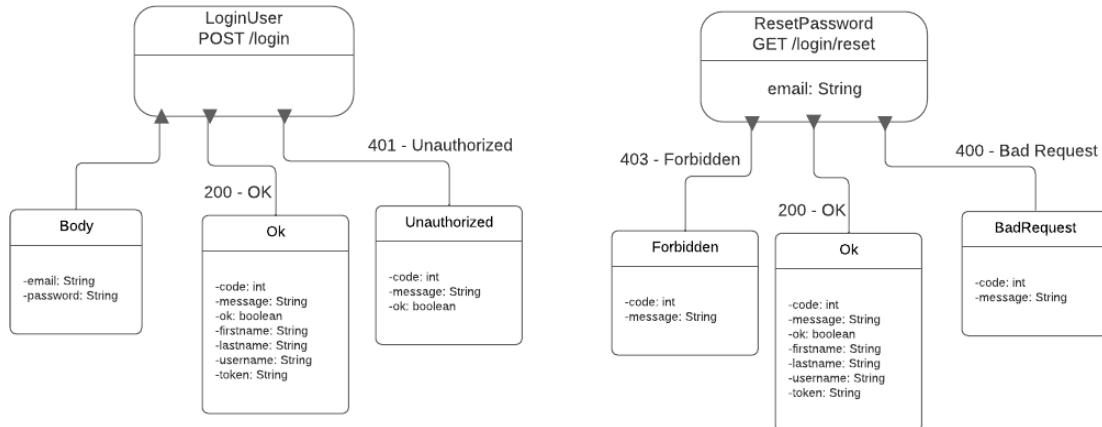
- Email



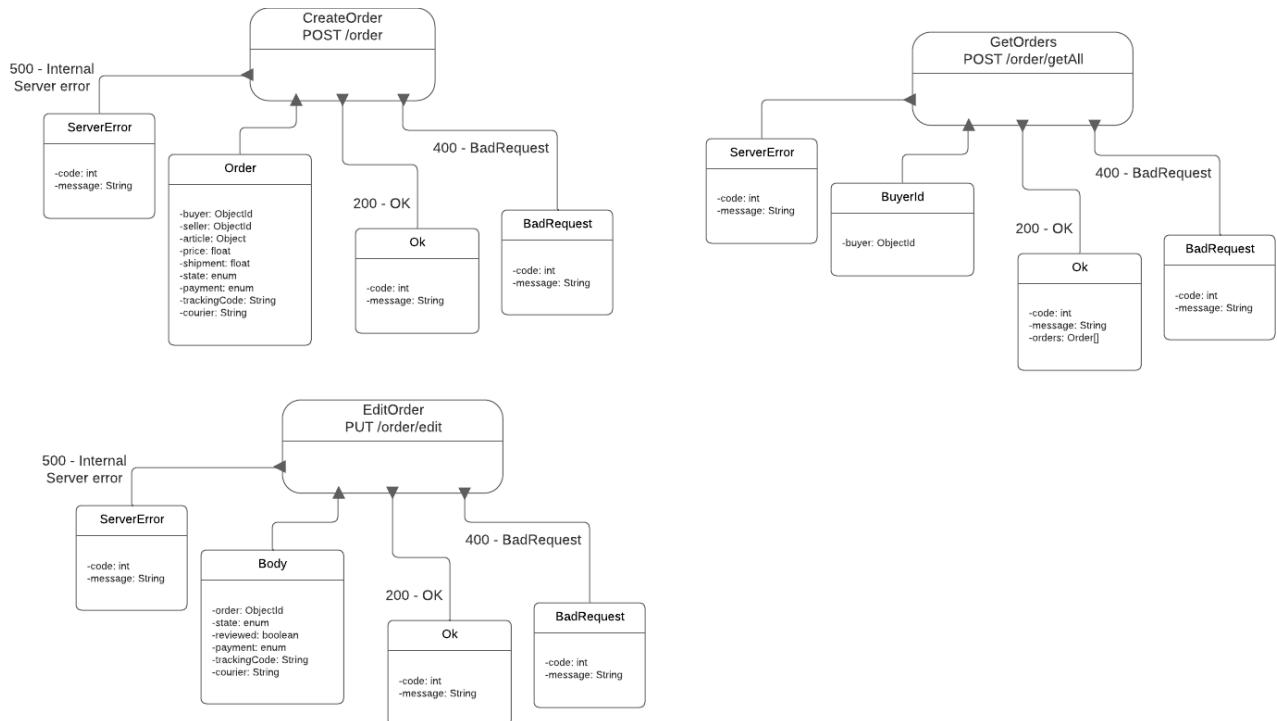
- Item



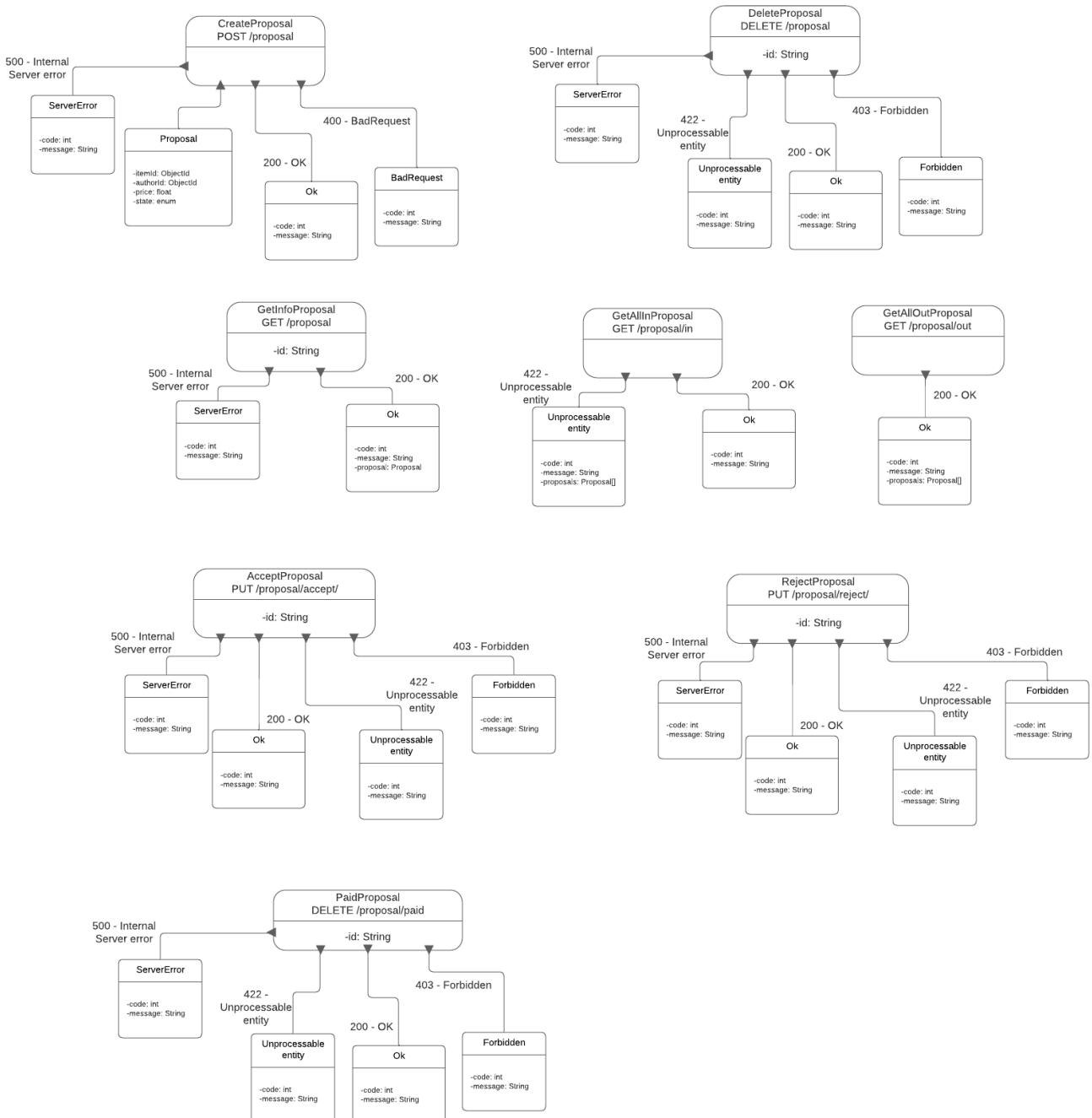
- Login



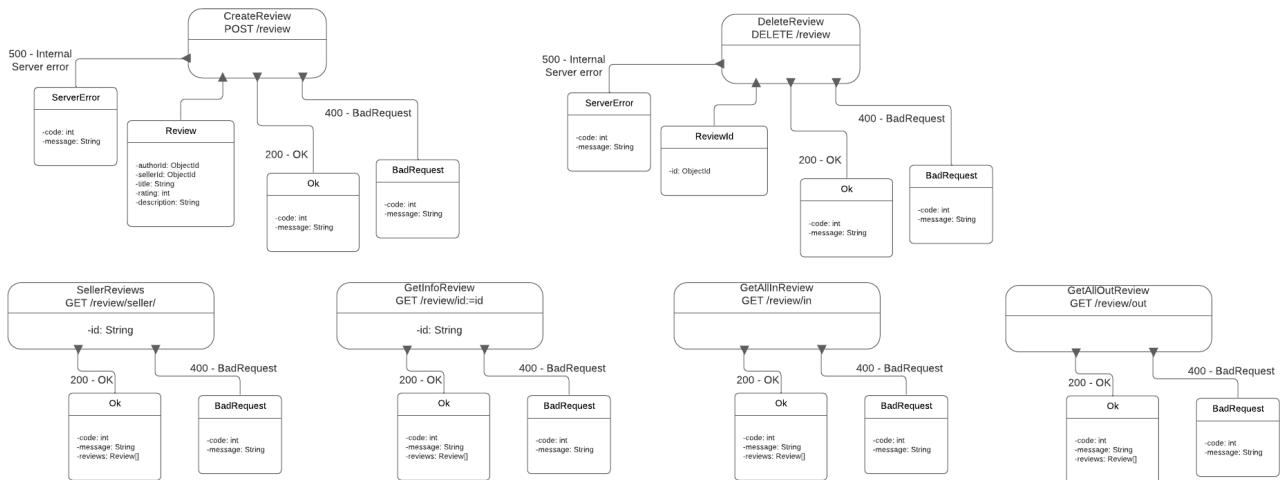
- Order



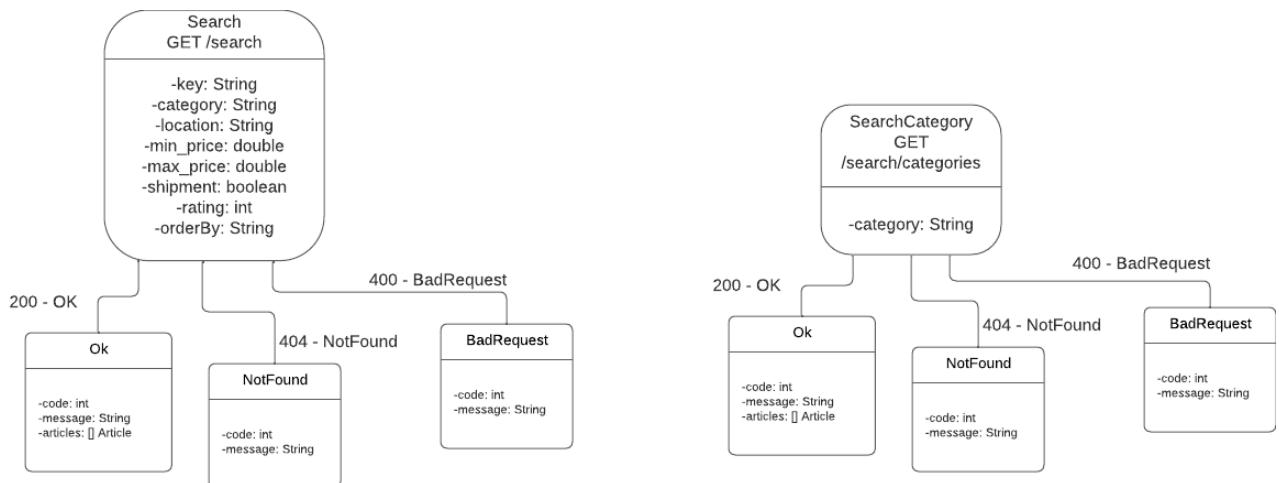
- Proposal



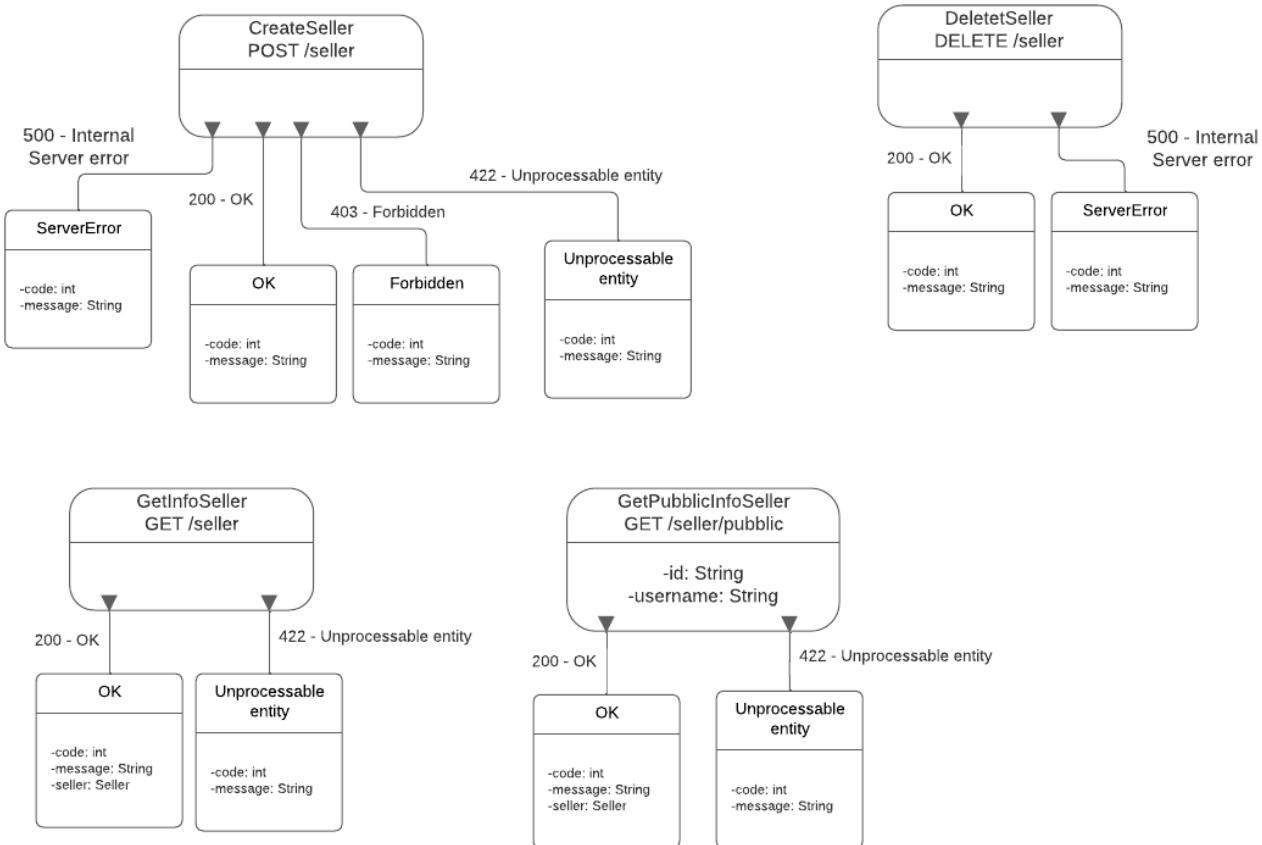
- Review



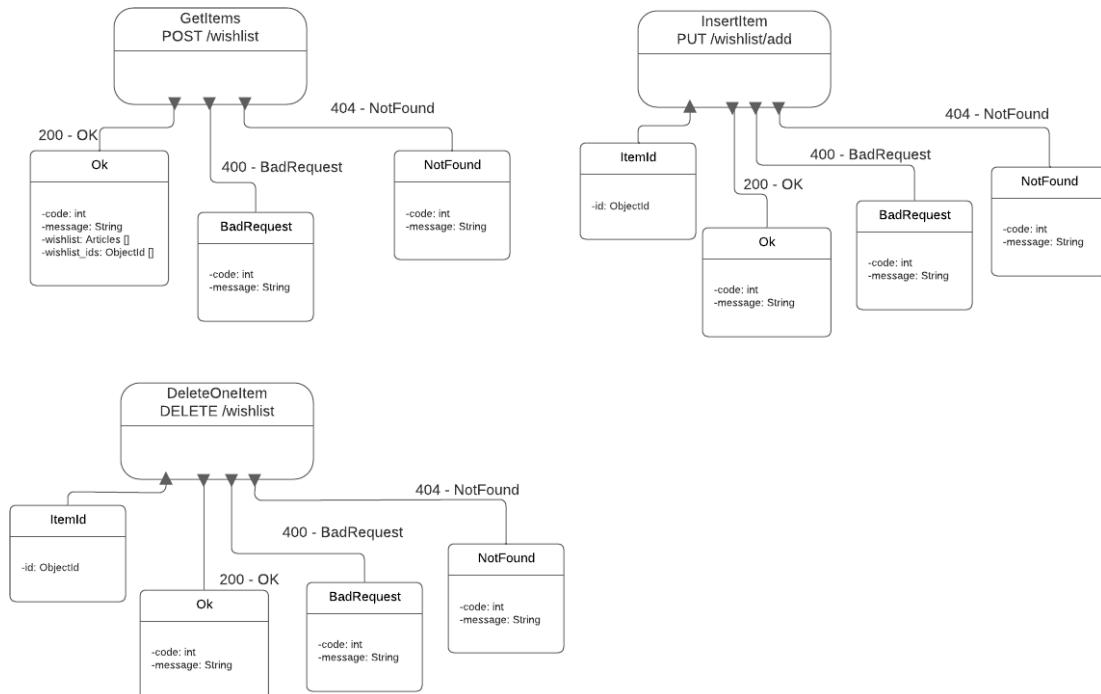
- Search



- Seller



- Wishlist



## 7. Sviluppo API

Nella sezione seguente si procede a riportare viste sul codice delle API più significative per ogni model riportando le funzioni necessarie ad implementare le operazioni CRUD. In caso di procedure particolari ci distaccheremo da questo schema per riportarle.

- Buyer

### Create

```
const create = async(req, res) => {
  const url = require('../utils/address');
  const data = req.body;

  if (!(data.firstname && data.lastname && data.username && data.email && data.password && data.terms))
    return res.status(403).json({ code: 102, message: 'Missing arguments' });

  const hash = crypto.createHash('sha256');
  const password = hash.update(data.password, 'utf-8').digest('hex');

  let code, result;
  do {
    code = crypto.randomBytes(32).toString('hex');
    result = await Buyer.exists({ verificationCode: code });
  } while (result);

  if (await Buyer.exists({ username: data.username }))
    return res.status(422).json({ code: 103, message: "Username not available" });

  const buyer = new Buyer({
    firstname: data.firstname,
    lastname: data.lastname,
    username: data.username,
    email: data.email,
    passwordHash: password,
    addresses: { address: data.address, isDefault: true },
    phone: { prefix: data.prefix, number: data.number },
    isTerms: true,
    isVerified: false,
    verificationCode: code,
  });

  let seller;
  if (data.isSeller && data.address && data.prefix && data.number)
    buyer.isSeller = true;
  seller = new Seller({
    userId: buyer
  });
  buyer.sellerId = seller.id;

  try {
    await buyer.save();
    if (seller)
      await seller.save();

    await Mail.send(data.email, 'Creazione Account Skupply', `Grazie per aver scelto skupply.\nPer verificare la tua registrazione, clicca sul link: ${url}/verify?code=${code}`);
  } catch (error) {
    return res.status(500).json({ code: "101", message: "unable to create" });
  }
}
```

### Update

```
const edit = async(req, res) => {
  let buyer = await getAuthenticatedBuyer(req, res);

  if (req.body.firstname)
    buyer.firstname = req.body.firstname;

  if (req.body.lastname)
    buyer.lastname = req.body.lastname;

  if (req.body.password) {
    const hash = crypto.createHash('sha256');
    const password = hash.update(req.body.password, 'utf-8').digest('hex');
    buyer.passwordHash = password;
  }

  if (req.body.prefix)
    buyer.phone.prefix = req.body.prefix;

  if (req.body.number)
    buyer.phone.number = req.body.number;

  if (req.body.addresses)
    buyer.addresses = req.body.addresses;

  buyer.save()
  .then(ok => {
    return res.status(200).json({ code: "", message: "success" })
  })
  .catch(err => {
    return res.status(500).json({ code: "", message: "unable to save changes" });
  });
}
```

### Delete

```
const remove = async(req, res) => {
  let buyer = await getAuthenticatedBuyer(req, res);

  //remove seller
  if (buyer.isSeller) {
    let seller = await Seller.findById(buyer.sellerId);

    if (seller.items)
      seller.items.forEach(async itemId => {
        let item = await Item.findById(itemId);
        item.state = 'DELETED';
        item.save().then(ok => {}).catch(err => {
          return res.status(500).json({ code: "", message: "unable to save changes" });
        });
      });

    if (seller.reviews) {
      await Review.deleteMany({ _id: { $in: seller.reviews } }).catch(err => { return res.status(500) });
    }

    if (seller.proposals)
      seller.proposals.forEach(async proposalId => {
        let proposal = await Proposal.findById({ _id: proposalId }, { state: 'PENDING' });
        proposal.state = 'DELETED';
        await proposal.save().catch(err => { return res.status(500).json({ code: "", message: "unat" });
      });

    Seller.deleteOne({ id: seller.id }, err => {
      if (err)
        return res.status(500).json({ code: "", message: "unable to remove" });
    });
  }

  if (buyer.proposals)
    buyer.proposals.forEach(async proposalId => {
      let proposal = await Proposal.findById({ _id: proposalId }, { state: 'PENDING' });
      proposal.state = 'DELETED';
      await proposal.save().catch(err => { return res.status(500).json({ code: "", message: "unable t" });
    });

  if (buyer.chats) {
    await Chat.deleteMany({ $or: [{ user1: buyer._id }, { user2: buyer._id }] }).catch(err => { return
  }

  Buyer.deleteOne({ id: buyer.id }, err => {
    if (err)
      return res.status(500).json({ code: "", message: "unable to remove" });
  });

  return res.status(200).json({ code: "", message: "success" });
}
```

### Read

```
const getInfo = async(req, res) => {
  const buyer = await getAuthenticatedBuyer(req, res);
  const ret = {
    id: buyer.id,
    firstname: buyer.firstname,
    lastname: buyer.lastname,
    username: buyer.username,
    email: buyer.email,
    addresses: buyer.addresses,
    phone: buyer.phone,
    cart: buyer.cart,
    wishlist: buyer.wishlist,
    proposals: buyer.proposals,
    isVerified: buyer.isVerified,
    isSeller: buyer.isSeller,
    sellerId: buyer.sellerId
  }
  return res.status(200).json({ buyer: ret, code: "", message: "success" });
}
```



- Cart

## Inserimento di un item

```
const insertItem = async(req, res) => {
  const data = req.body;
  let id_item = data.id;
  let user = await getAuthenticatedBuyer(req, res);

  if (!id_item)
    return res.status(400).json({ code: "402", message: "missing arguments" });

  if (!mongoose.Types.ObjectId.isValid(id_item)) || !(await Item.findById(id_item))
    return res.status(404).json({ code: "401", message: "item not found" });

  //se l'elemento è un duplicato, questo non viene inserito e non va a modificare la
  //quantità di quello già presente
  const result = await Buyer.find({ "$and": [{ _id: user._id }, { cart: { $elemMatch: { id: id_item } } }] });
  if (!result) return res.status(404).json({ code: "401", message: "user or item not found" });
  else {
    if (Object.keys(result).length === 0) {
      //item non già presente nel carrello, inserimento id
      const result = await Buyer.updateOne({ _id: user._id }, { $push: { cart: { id: id_item } } });
      return res.status(200).json({ code: "400", message: "product added in cart" });
    } else
      return res.status(200).json({ code: "400", message: "product not added in cart" });
  }
}
```

## Aggiornamento quantità di un item

```
const updateQuantity = async(req, res) => {
  let id = req.body.id; //id item
  let quantity = req.body.quantity;
  let user = await getAuthenticatedBuyer(req, res);

  if (!id || quantity < 0)
    return res.status(400).json({ code: "402", message: "missing arguments" });
  //campi non presenti o non validi, sessione probabilmente non valida

  if (!mongoose.Types.ObjectId.isValid(id))
    return res.status(404).json({ code: "401", message: "item not found" });

  let result = await Buyer.findByIdAndUpdate(user._id);
  if (!result) return res.status(404).json({ code: "403", message: "user not found" });

  //modifica quantità articolo carrello
  const items = result.cart;
  let id_item; //questo è l'objectid dell'item
  let notFound = true;

  for (i = 0; i < items.length && notFound; i++) {
    if (items[i].id == id) {
      //item trovato
      notFound = false;
      id_item = items[i]._id;
    }
  }

  if (notFound) return res.status(404).json({ code: "404", message: "product not found" });

  result = await Buyer.updateOne({ $and: [{ id: user._id }, { 'cart._id': id_item }] }, {
    $set: { 'cart.$.quantity': quantity }
  });

  if (!result) return res.status(500).json({ code: "401", message: "database error" });
  return res.status(200).json({ code: "400", message: "product's quantity updated" });
}
```

## Rimozione di un item

```
const deleteOneItem = async(req, res) => {
  //remove an item with a defined id
  let id = req.body.id;
  let user = await getAuthenticatedBuyer(req, res);

  if (!id)
    return res.status(400).json({ code: "402", message: "missing arguments" });
  //campi non presenti, sessione probabilmente non valida

  //modifica carrello del risultato ottenuto
  const items = user.cart;
  let id_item;
  let notFound = true;

  for (i = 0; i < items.length && notFound; i++) {
    if (items[i].id == id) {
      //item trovato
      notFound = false;
      id_item = items[i]._id;
    }
  }

  if (notFound) return res.status(404).json({ code: "404", message: "product not found" });

  result = await Buyer.updateOne({ _id: user._id }, {
    $pull: {
      cart: {
        _id: { $in: id_item }
      }
    }
  });

  if (!result) return res.status(404).json({ code: "401", message: "database error" });
  return res.status(200).json({ code: "400", message: "product removed" });
};
```

## Read

```
const getItems = async(req, res) => {
  //get all items inserted in the cart
  let user = await getAuthenticatedBuyer(req, res);

  const result = await Buyer.findById(user._id);
  if (!result) return res.status(404).json({ code: "403", message: "user not found" });

  //una volta trovati gli id, devo trovare i prodotti all'interno della collection articoli
  let cart = result.cart;
  let articoli = [];

  for (let i = 0; i < cart.length; i++) {
    const result = await Item.findOne({ _id: cart[i].id });
    if (result) articoli.push(result);
  }

  //inserire nella risposta gli articoli
  return res.status(200).json({ code: "400", message: "success", cart: articoli, cart_ids: cart });
};
```

- Chat

## Create

```

const createChat = async(req, res) => {
  const username = req.query.username;
  if (!username) { res.status(400).json({ code: 802, message: 'Username argument is missing' }); return }

  const contactUsername = req.body.contact;
  if (!contactUsername) { res.status(400).json({ code: 802, message: 'Contact property is missing' }); re
  if (username == contactUsername) { res.status(400).json({ code: 806, message: 'Contact can not coincide' })

  const contact = await Buyer.findOne({ username: contactUsername });
  if (!contact) { res.status(404).json({ code: 804, message: 'The provided contact does not exist' }); re

  const user = await Buyer.findOne({ username: username });
  if (!user) { res.status(404).json({ code: 805, message: 'The provided user does not exist' }); return }

  const Id = mongoose.Types.ObjectId;
  const checkChat = await Chat.findOne({
    $or: [
      { $and: [{ user1: { id: new Id(user.id) } }, { user2: { id: new Id(contact.id) } } ] },
      { $and: [{ user1: { id: new Id(contact.id) } }, { user2: { id: new Id(user.id) } } ] }
    ]
  });

  if (!checkChat) {
    const chat = new Chat({
      user1: { id: user.id },
      user2: { id: contact.id },
    });

    await chat.save().catch(err => console.log(err))
  }

  res.status(200).json({ code: 800, message: 'Chat created successfully' });
}

```

## Update

```

const sendMessage = async(req, res) => {
  const username = req.query.username;
  if (!username) { res.status(400).json({ code: 802, message: 'Username argument is missing' }); return }

  const contactUsername = req.body.contact;
  if (!contactUsername) { res.status(400).json({ code: 802, message: 'Contact property is missing' }); re
  const messageText = req.body.message;

  if (!messageText) { res.status(400).json({ code: 802, message: 'Message property is missing' }); return }
  if (username == contactUsername) { res.status(400).json({ code: 806, message: 'Contact can not coincide' })

  const contact = await Buyer.findOne({ username: contactUsername });
  if (!contact) { res.status(404).json({ code: 804, message: 'The provided contact does not exist' }); re

  const user = await Buyer.findOne({ username: username });
  if (!user) { res.status(404).json({ code: 805, message: 'The provided user does not exist' }); return }

  const Id = mongoose.Types.ObjectId;

  const chat = await Chat.findOne({
    $or: [
      { $and: [{ user1: { id: new Id(user.id) } }, { user2: { id: new Id(contact.id) } } ] },
      { $and: [{ user1: { id: new Id(contact.id) } }, { user2: { id: new Id(user.id) } } ] }
    ]
  });

  if (!chat) { res.status(404).json({ code: 807, message: 'Chat not found' }); return }
  const message = new Message({
    sender: { id: new Id(user.id) },
    text: messageText
  });

  await message.save().catch(err => console.log(err))

  //update chat con inserimento nuovo messaggio
  const result = await Chat.updateOne({
    $or: [
      { $and: [{ user1: { id: new Id(user.id) } }, { user2: { id: new Id(contact.id) } } ] },
      { $and: [{ user1: { id: new Id(contact.id) } }, { user2: { id: new Id(user.id) } } ] }
    ],
    { $push: { "messages": { id: new Id(message._id) } } }
  });

  res.status(200).json({ code: 800, message: 'Message sent successfully' });
}

```

## Delete

```

const deleteChat = async(req, res) => {
  const username = req.query.username;
  if (!username) { res.status(400).json({ code: 802, message: 'Username argument is missing' }); return }

  const contactUsername = req.body.contact;
  if (!contactUsername) { res.status(400).json({ code: 802, message: 'Contact property is missing' }); re
  if (username == contactUsername) { res.status(400).json({ code: 806, message: 'Contact can not coincide' })

  const contact = await Buyer.findOne({ username: contactUsername });
  if (!contact) { res.status(404).json({ code: 804, message: 'The provided contact does not exist' }); re

  const user = await Buyer.findOne({ username: username });
  if (!user) { res.status(404).json({ code: 805, message: 'The provided user does not exist' }); return }

  const Id = mongoose.Types.ObjectId;
  const chat = await Chat.findOne({
    $or: [
      { $and: [{ user1: { id: new Id(user.id) } }, { user2: { id: new Id(contact.id) } } ] },
      { $and: [{ user1: { id: new Id(contact.id) } }, { user2: { id: new Id(user.id) } } ] }
    ]
  });

  if (!chat) { res.status(404).json({ code: 807, message: 'Chat not found' }); else await Message.deleteMany({ _id: { $in: chat.messages.map(message => new Id(message.id) ) } });

  Chat.deleteOne({
    $or: [
      { $and: [{ user1: { id: new Id(user.id) } }, { user2: { id: new Id(contact.id) } } ] },
      { $and: [{ user1: { id: new Id(contact.id) } }, { user2: { id: new Id(user.id) } } ] }
    ]
  }, (err, data) => {
    if (err) res.status(500).json({ code: 801, message: 'Database error' });
    else res.status(200).json({ code: 800, message: 'Chat deleted successfully' });
  });
}

```

## Read

```

const getChat = async(req, res) => {
  const username = req.query.username;
  if (!username) { res.status(400).json({ code: 802, message: 'Username argument is missing' }); return }

  const result = await Buyer.findOne({ username: username });
  if (!result) res.status(404).json({ code: 803, message: 'User not found' });

  let idUser = result._id;
  //ricerca all'interno della collection Chat
  const result2 = await Chat.find({
    "$or": [
      { "user1.id": idUser, "user2.id": idUser },
      { "user2.id": idUser, "user1.id": idUser }
    ]
  });

  res.status(200).json({ code: 800, message: 'Success', chats: result2 });
}

```

- Email

Verifica esistenza

```
const checkEmail = async(req, res) => {
  const email = req.query.email;
  if (!email) { res.status(400).json({ code: 202, message: 'Email argument is missing' }); return }

  const check = await Buyer.findOne({ email: email });
  if (!check) { res.status(403).json({ code: 205, message: 'Email already used' }); return }

  const result = await validator.validate(email);
  res.status(200).json(result.valid ? { code: 203, message: 'Email reachable' } : { code: 204, message: 'Email not reachable' });
};
```

Verifica validità

```
const verifyEmail = async(req, res) => {
  const code = req.body.code;
  const email = req.query.email;
  if (!code) { res.status(400).json({ code: 202, message: 'Code argument is missing' }); return }
  if (!email) { res.status(400).json({ code: 203, message: 'Email argument is missing' }); return }

  const check = await Buyer.findOne({ email: email });
  if (!check) { res.status(403).json({ code: 205, message: 'Email not associated to any account' }); return }
  if (!check.isVerified) { res.status(200).json({ code: 207, message: 'Email already verified' }); return;

  if (check.verificationCode == code) {
    check.isVerified = true;
    await check.save();
    res.status(200).json({ code: 208, message: "Email verified successfully" });
  } else {res.status(200).json({ code: 206, message: "Invalid verification code" })};
}
```

- Login

Login

```
const loginUser = async(req, res) => {
  const data = req.body;
  const hash = crypto.createHash('sha256');
  const password = hash.update(data.password, 'utf-8').digest('hex');

  const result = await Buyer.findOne({ email: data.email });
  const token = jwt.sign(data.email, process.env.ACCESS_TOKEN);

  if (result && result.passwordHash == password)
    res.status(200).json({
      code: "200",
      message: "logged in",
      ok: true,
      user: {
        id: result.id,
        firstname: result.firstname,
        lastname: result.lastname,
        username: result.username,
        email: result.email,
        addresses: result.addresses,
        phone: result.phone,
        cart: result.cart,
        wishlist: result.wishlist,
        proposals: result.proposals,
        isVerified: result.isVerified,
        isSeller: result.isSeller,
        sellerId: result.sellerId,
        token: token
      }
    });
  else
    res.status(401).json({ code: "303", message: "wrong credentials", ok: false });
};
```

Recupero password

```
const resetPassword = async(req, res) => {
  const email = req.query.email;
  if (!email) return res.status(400).json({ code: '302', message: 'Missing arguments' });

  const result = await Buyer.findOne({ email });
  if (!result) return res.status(403).json({ code: '306', message: 'Invalid email' });

  const hash = crypto.createHash('sha256');
  const random = crypto.randomBytes(16).toString('hex');
  const password = hash.update(random, 'utf-8').digest('hex');

  result.passwordHash = password;

  await result.save().catch(err => {
    console.log(err);
    return res.status(500).json({ code: '301', message: 'Database error' });
  })

  const url = require('../utils/address');
  await Mail.send(result.email, 'Reset Password Skupply', `La tua nuova password è: ${random}\nPuoi cambiare la password da questa pagina`);

  res.status(200).json({ code: '300', message: 'Password resettata correttamente' });
};
```

- Item

### Create

```
const create = async(req, res) => {
  const token = req.headers['x-access-token'];
  if (!token) return res.status(400).json({ code: "902", message: "missing arguments" });

  const email = jwt.verify(token, process.env.ACCESS_TOKEN, {err, data} => data);
  const buyer = await Buyer.findOne({ email });

  let seller = await Seller.findById(buyer.sellerId);

  const itemCategories = await Category.find({ title: { $in: req.body.categories } });
  const categories = itemCategories.map(category => category.id)

  let item = new Item({
    title: req.body.title,
    description: req.body.description,
    ownerId: seller.id,
    quantity: req.body.quantity,
    categories: categories,
    photos: [ ${buyer.sellerId}.${seller.items.length}.${req.body.ext}` ],
    conditions: req.body.conditions,
    price: req.body.price,
    city: req.body.city,
    state: 'DRAFT',
    pickupAvail: req.body.pickupAvail,
    shipmentAvail: req.body.shipmentAvail,
    shipmentCost: req.body.shipmentCost,
  });

  let error = false;
  await item.save().catch(err => {
    console.log(err);
    error = true
  })

  if (error) return res.status(500).json({ code: "901", message: "unable to create" });

  if (!seller.items) seller.items = [];
  seller.items.push(item.id);

  await seller.save().catch(err => {
    console.log(err);
    error = true;
  });

  if (error) return res.status(500).json({ code: "901", message: "unable to save changes" });

  return res.status(201).json({ code: "900", message: "success", item: item.id });
}
```

### Update

```
const edit = async(req, res) => {
  // required params
  if (!req.body.itemId)
    return res.status(400).json({ code: "902", message: "missing arguments" });

  if (!mongoose.Types.ObjectId.isValid(req.body.itemId) || !(await Item.exists({ id: req.body.itemId })))
    return res.status(400).json({ code: "903", message: "invalid arguments" });

  let buyer = await getAuthenticatedBuyer(req, res);
  if (!buyer.isSeller)
    return res.status(400).json({ code: "904", message: "invalid user type" });

  let seller = await Seller.findById(buyer.sellerId);
  if (!seller.items.includes(req.body.itemId))
    return res.status(400).json({ code: "905", message: "operation not permitted" });

  let item = await Item.findById(req.body.itemId);

  const itemCategories = await Category.find({ title: { $in: req.body.categories } });
  const categories = itemCategories.map(category => category.id)

  item.title = req.body.title ? req.body.title : item.title;
  item.description = req.body.description ? req.body.description : item.description;
  item.quantity = req.body.quantity ? Number(req.body.quantity) : item.quantity;
  item.categories = req.body.categories ? categories : item.categories;
  item.conditions = req.body.conditions ? req.body.conditions : item.conditions;
  item.price = req.body.price ? parseFloat(req.body.price) : item.price;
  item.city = req.body.city ? req.body.city : item.city;
  item.pickupAvail = req.body.pickupAvail ? req.body.pickupAvail : item.pickupAvail;
  item.shipmentAvail = req.body.shipmentAvail ? req.body.shipmentAvail : item.shipmentAvail;
  item.shipmentCost = req.body.shipmentCost ? parseFloat(req.body.shipmentCost) : item.shipmentCost;
  item.photos[0] = req.body.ext ? String(buyer.sellerId).concat('_').concat(seller.items.length).concat(`.${req.body.ext}`) : item.photos[0];

  item.save()
    .then(ok => {
      if (ok)
        return res.status(200).json({ code: "900", message: "success" });
    })
    .catch(err => {
      console.log(err);
      return res.status(500).json({ code: "901", message: "unable to save changes" });
  })
}
```

### Delete

```
const remove = async(req, res) => {
  // required params
  if (!req.query.id)
    return res.status(400).json({ code: "902", message: "missing arguments" });

  if (!mongoose.Types.ObjectId.isValid(req.query.id) || !(await Item.exists({ id: req.query.id })))
    return res.status(400).json({ code: "903", message: "invalid arguments" });

  let buyer = await getAuthenticatedBuyer(req, res);
  if (!buyer.isSeller)
    return res.status(400).json({ code: "904", message: "invalid user type" });

  let seller = await Seller.findById(buyer.sellerId);
  if (!seller.items.includes(req.query.id))
    return res.status(400).json({ code: "905", message: "operation not permitted" });

  let item = await Item.findById(req.query.id);

  // delete all proposals
  var proposals = await Proposal.find({ $and: [{ itemId: item._id }, { state: "PENDING" }] });
  proposals.forEach(async proposal => {
    proposal.state = "DELETED";
    await proposal.save().catch(err => res.status(500).json({ code: "901", message: "unable to save changes" }));
  })

  // remove from carts and wishlists
  var buyers = await Buyer.find();
  buyers.forEach(async buyer => {
    if (buyer.wishlist.findIndex(x => x.id == item._id)) {
      buyer.wishlist = buyer.wishlist.filter(x => x.id != item._id)
    } else if (buyer.cart.findIndex(x => x.id == item._id)) {
      buyer.cart = buyer.cart.filter(x => x.id != item._id)
    }
  })

  await buyer.save().catch(err => res.status(500).json({ code: "901", message: "unable to save change" }));

  item.state = "DELETED";
  await item.save().catch(err => {
    return res.status(500).json({ code: "901", message: "unable to save changes" });
  });

  return res.status(200).json({ code: "900", message: "success" });
}
```

### Buy

```
const buy = async(req, res) => {
  // required params
  if (!req.query.id || !req.query.quantity)
    return res.status(400).json({ code: "902", message: "missing arguments" });

  if (!mongoose.Types.ObjectId.isValid(req.query.id) || !(await Item.exists({ _id: req.query.id })))
    return res.status(400).json({ code: "903", message: "invalid arguments" });

  if (!Number.isInteger(req.query.quantity) || !req.query.quantity > 0)
    return res.status(400).json({ code: "903", message: "invalid arguments" });

  let item = await Item.findById(req.query.id);

  if (item.state != 'PUBLISHED')
    return res.status(400).json({ code: "907", message: "invalid item state" });

  if (item.quantity < req.query.quantity)
    return res.status(400).json({ code: "906", message: "max quantity exceeded" });

  item.quantity -= req.query.quantity;
  if (item.quantity <= 0) {
    item.state = 'SOLD';
    item.quantity = 0;
  }

  // delete all proposals
  var proposals = await Proposal.find({ $and: [{ itemId: item._id }, { state: "PENDING" }] });
  proposals.forEach(async proposal => {
    proposal.state = "DELETED";
    await proposal.save().catch(err => res.status(500).json({ code: "901", message: "unable to save changes" }));
  })

  // remove from carts and wishlists
  var buyers = await Buyer.find();
  buyers.forEach(async buyer => {
    if (buyer.wishlist.findIndex(x => x.id == item._id)) {
      buyer.wishlist = buyer.wishlist.filter(x => x.id != item._id)
    } else if (buyer.cart.findIndex(x => x.id == item._id)) {
      buyer.cart = buyer.cart.filter(x => x.id != item._id)
    }
  })

  await buyer.save().catch(err => res.status(500).json({ code: "901", message: "unable to save change" }));

  await item.save().catch(err => {
    return res.status(500).json({ code: "901", message: "unable to save changes" });
  });

  return res.status(200).json({ code: "900", message: "success" });
}
```

- Order

### Create

```
const create = async(req, res) => {
  let user = await getAuthenticatedBuyer(req, res);

  const buyer = user._id; //id del compratore
  const seller = req.body.seller; //id del venditore
  const article = req.body.article;
  const price = req.body.price;
  const shipment = req.body.shipment;
  const state = "PAID"; //l'ordine creato ha come stato pagato
  const payment = "LOCKED"; //il codice create ha come stato pagamento locked
  const trackingCode = req.body.anyTrackingCode; //codice di tracking del pacco
  const courier = req.body.courier; //il corriere che gestisce la spedizione

  //verifica presenza parametri di richiesta
  //non viene fatto il controller per shipment perché potrebbe avere valore zero
  //che un valore valido a differenza di price
  if (!buyer || !seller || !article || !price) {
    return res.status(400).json({ code: 1002, message: "Missing arguments" });
  }

  //verifica esistenza buyer e articolo
  if (!mongoose.Types.ObjectId.isValid(buyer)) || !(await Buyer.findById(buyer)) {
    return res.status(404).json({ code: 1005, message: "Buyer not found" });
  }

  if (!mongoose.Types.ObjectId.isValid(article.id)) || !(await Item.findById(article.id)) {
    return res.status(404).json({ code: 1005, message: "Item not found" });
  }

  //creazione ordine e salvataggio su db
  const order = new Order({
    buyer: buyer,
    seller: seller,
    article: article,
    price: price,
    shipment: shipment,
    state: state,
    payment: payment,
    trackingCode: trackingCode,
    courier: courier
  });

  try {
    await order.save();
    return res.status(200).json({ code: 1000, message: "success" });
  } catch (error) {
    return res.status(500).json({ code: 1001, message: "database error" });
  }
};
```

### Update

```
const edit = async(req, res) => {
  const order = req.body.orderId; //id ordine
  const newState = req.body.state; //nuovo stato dell'ordine
  const newReviewed = req.body.reviewed; //valore nuovo flag reviewed
  const newPayment = req.body.payment; //valore nuovo stato pagamento
  const trackingCode = req.body.trackingCode; //codice di tracking del pacco
  const courier = req.body.courier; //il corriere che gestisce la spedizione

  //verifica presenza parametri di richiesta
  if (!order) {
    return res.status(400).json({ code: 1002, message: "Missing arguments" });
  }

  //verifica esistenza ordine
  if (!mongoose.Types.ObjectId.isValid(order)) || !(await Order.findById(order)) {
    return res.status(404).json({ code: 1007, message: "Order not found" });
  }

  //verifica valore enumerativo newState [ PAID, SHIPPED, COMPLETED, DELETED]
  if (newState && newState != "PAID" && newState != "SHIPPED" && newState != "COMPLETED" && newState != "DELETED")
    return res.status(403).json({ code: 1003, message: "Invalid arguments" });

  //verifica valore enumerativo newPayment [ PAID, SHIPPED, COMPLETED, DELETED]
  if (newPayment && newPayment != "LOCKED" && newPayment != "SENT" && newPayment != "REJECTED")
    return res.status(403).json({ code: 1003, message: "Invalid arguments" });

  //recupero ordine e modifica dei campi
  try {
    let result = await Order.findByIdAndUpdate(order);
    if (newState) result.state = newState;
    if (newReviewed) result.reviewed = newReviewed;
    if (newPayment) result.payment = newPayment;
    //per la modifica dei parametri riguardanti l'ordine, i valori trackingCode e courier
    //devono essere entrambi presenti
    if (trackingCode && courier) {
      result.trackingCode = trackingCode;
      result.courier = courier;
    }

    await result.save();
    return res.status(200).json({ code: 1000, message: "success" });
  } catch (error) {
    return res.status(500).json({ code: 1001, message: "database error" });
  }
};
```

### Recupero degli ordini di un acquirente (buyer)

```
const getAll = async(req, res) => {
  let user = await getAuthenticatedBuyer(req, res);

  const buyer = user._id; //id del compratore
  //verifica presenza parametri di richiesta
  if (!buyer)
    return res.status(400).json({ code: 1002, message: "Missing arguments" });

  //verifica esistenza buyer e articoli
  if (!mongoose.Types.ObjectId.isValid(buyer)) || !(await Buyer.findById(buyer)) {
    return res.status(404).json({ code: 1005, message: "Buyer not found" });
  }

  //recupero ordini fatti dal buyer
  try {
    const result = await Order.find({ buyer: buyer });

    return res.status(200).json({ code: 1000, message: "success", orders: result });
  } catch (error) {
    return res.status(500).json({ code: 1001, message: "database error" });
  }
};
```

### Recupero degli ordini di un venditore (seller)

```
const buy = async(req, res) => {
  // required params
  if (!req.query.id || !req.query.quantity)
    return res.status(400).json({ code: 902, message: "missing arguments" });

  if (!mongoose.Types.ObjectId.isValid(req.query.id)) || !(await Item.exists({ _id: req.query.id }))
    return res.status(400).json({ code: 903, message: "invalid arguments" });

  if (!Number.isInteger(req.query.quantity) || req.query.quantity < 0)
    return res.status(400).json({ code: 903, message: "invalid arguments" });

  let item = await Item.findById(req.query.id);

  if (item.state != 'PUBLISHED')
    return res.status(400).json({ code: 907, message: "invalid item state" });

  if (item.quantity < req.query.quantity)
    return res.status(400).json({ code: 906, message: "max quantity exceeded" });

  item.quantity -= req.query.quantity;
  if (item.quantity <= 0) {
    item.state = 'SOLD';
    item.quantity = 0;

    // delete all proposals
    var proposals = await Proposal.find({ $and: [{ itemId: item._id }, { state: "PENDING" }] });
    proposals.forEach(async proposal => {
      proposal.state = "DELETED";
      await proposal.save().catch(err => res.status(500).json({ code: "901", message: "unable to save changes" }));
    });

    // remove from carts and wishlists
    var buyers = await Buyer.find();
    buyers.forEach(async buyer => {
      if (buyer.wishList.find(x => x.id == item._id)) {
        buyer.wishList = buyer.wishList.filter(x => x.id != item._id);
      } else if (buyer.cart.find(x => x.id == item._id)) {
        buyer.cart = buyer.cart.filter(x => x.id != item._id);
      }

      await buyer.save().catch(err => res.status(500).json({ code: "901", message: "unable to save changes" }));
    });

    await item.save().catch(err => {
      return res.status(500).json({ code: "901", message: "unable to save changes" });
    });
  }

  return res.status(200).json({ code: "900", message: "success" });
};
```

- **Proposal**

### Create

```
const create = async(req, res) => {
    // required params
    if (!req.body.itemId && !req.body.price)
        return res.status(400).json({ code: "1102", message: "missing arguments" });

    let valid = true;
    // article ID must exist
    if (valid)
        valid = mongoose.Types.ObjectId.isValid(req.body.itemId) && (await Item.exists({_id: req.body.itemId}));

    let author = await getAuthenticatedBuyer(req, res);

    // only one proposal can exists for one author and one item
    if (valid)
        valid = !(await Proposal.findOne({ itemId: req.body.itemId, authorId: author._id, state: 'PENDING' }));

    // the author of the proposal must be different from the item owner
    if (valid)
        valid = !author.isSeller || !(await Seller.findById(author.sellerId)).items.includes(req.body.itemId);

    if (valid)
        return res.status(422).json({ code: "1103", message: "invalid arguments" });

    let proposal = new Proposal({
        itemId: req.body.itemId,
        authorId: author._id,
        state: 'PENDING',
        price: req.body.price
    });

    await proposal.save()
        .catch(err => {
            return res.status(500).json({ code: "1101", message: "unable to create" });
        });

    author.proposals.push(proposal.id);
    await author.save()
        .catch(err => {
            return res.status(500).json({ code: "1101", message: "unable to save changes" });
        });

    const item = await Item.findById(req.body.itemId);
    let seller = await Seller.findById(item.ownerId);
    seller.proposals.push(proposal.id);
    seller.save()
        .then(ok => {
            return res.status(201).json({ code: "1100", message: "success" });
        })
        .catch(err => {
            return res.status(500).json({ code: "1101", message: "unable to save changes" });
        });
}
```

### Accept / Reject

```
const accept = async(req, res) => {
    //id must exist
    if (!mongoose.Types.ObjectId.isValid(req.params.id) || !(await Proposal.exists({ id: req.params.id })))
        return res.status(500).json({ code: "1103", message: "invalid arguments" });
    let proposal = await Proposal.findById(req.params.id);

    // verify authorization
    let buyer = await getAuthenticatedBuyer(req, res);
    if (!buyer.isSeller || !(await Seller.findById(buyer.sellerId)).proposals.includes(proposal.id))
        return res.status(403).json({ code: "1105", message: "proposal on not owned item" });

    //proposal must be in 'PENDING' state
    if (proposal.state != 'PENDING')
        return res.status(422).json({ code: "1104", message: "invalid proposal state" });

    proposal.state = 'ACCEPTED';
    proposal.save()
        .then(ok => {
            return res.status(200).json({ code: "1100", message: "success" });
        })
        .catch(err => {
            return res.status(500).json({ code: "1101", message: "unable to accept" });
        });
}

const reject = async(req, res) => {
    //id must exist
    if (!mongoose.Types.ObjectId.isValid(req.params.id) || !(await Proposal.exists({ id: req.params.id })))
        return res.status(500).json({ code: "", message: "invalid arguments" });
    let proposal = await Proposal.findById(req.params.id);

    // verify authorization
    let buyer = await getAuthenticatedBuyer(req, res);
    if (!buyer.isSeller || !(await Seller.findById(buyer.sellerId)).proposals.includes(proposal._id))
        return res.status(403).json({ code: "", message: "proposal on not owned item" });

    //proposal must be in 'PENDING' state
    if (proposal.state != 'PENDING')
        return res.status(422).json({ code: "", message: "invalid proposal state" });

    proposal.state = 'REJECTED';
    proposal.save()
        .then(ok => {
            return res.status(200).json({ code: "", message: "success" });
        })
        .catch(err => {
            return res.status(500).json({ code: "", message: "unable to reject" });
        });
}
```

### Delete

```
const remove = async(req, res) => {
    //id must exist
    if (!mongoose.Types.ObjectId.isValid(req.query.id) || !(await Proposal.exists({ id: req.query.id })))
        return res.status(500).json({ code: "1102", message: "invalid arguments" });

    // verify authorization
    let buyer = await getAuthenticatedBuyer(req, res);
    if (proposal.authorId != buyer.id)
        return res.status(403).json({ code: "1106", message: "not authorized" });

    //proposal must be in 'PENDING' state
    if (proposal.state != 'PENDING')
        return res.status(422).json({ code: "1104", message: "invalid proposal state" });

    proposal.state = 'DELETED';
    proposal.save()
        .then(ok => {
            return res.status(200).json({ code: "1100", message: "success" });
        })
        .catch(err => {
            return res.status(500).json({ code: "1101", message: "unable to remove" });
        });
}
```

### Read (in per i sellers e out per i buyers)

```
const getAllIn = async(req, res) => {
    const buyer = await getAuthenticatedBuyer(req, res);

    if (!buyer.isSeller)
        return res.status(422).json({ code: "1103", message: "invalid user type" });

    const seller = await Seller.findById(buyer.sellerId);
    let proposals = await Proposal.find({
        $and: [
            { _id: { $in: seller.proposals } },
            { state: { $in: ['PENDING'] } }
        ]
    });

    return res.status(200).json({ proposals: proposals, code: "1100", message: "success" });
}

const getAllOut = async(req, res) => {
    const buyer = await getAuthenticatedBuyer(req, res);

    //ricerco le proposte del buyer nella collection Proposals
    let proposals = [];
    for (let i = 0; i < buyer.proposals.length; i++) {
        let result = await Proposal.findById(buyer.proposals[i]);
        if (result != null)
            proposals.push(result);
    }

    return res.status(200).json({ proposals: proposals, code: "1100", message: "success" });
}
```

- Review

### Create

```
const create = async(req, res) => {
  //required params
  if (!req.body.title || req.body.rating == undefined || !req.body.sellerId)
    return res.status(400).json({ code: "802", message: "missing arguments" });

  // params validity
  if (!Number.isInteger(req.body.rating) || !(0 <= req.body.rating && req.body.rating <= 5) || !String.to
  return res.status(400).json({ code: "803", message: "invalid arguments" });

  if (!mongoose.Types.ObjectId.isValid(req.body.sellerId) || !(await Seller.exists({ id: req.body.sellerId })
    return res.status(400).json({ code: "803", message: "invalid arguments" });

  const author = await getAuthenticatedBuyer(req, res);
  const seller = await Seller.findById(req.body.sellerId);
  const review = new Review({
    authorId: author._id,
    sellerId: req.body.sellerId,
    title: req.body.title,
    description: req.body.description,
    rating: req.body.rating
  });

  try {
    await review.save()
    seller.reviews.push(review.id)
    await seller.save()

    return res.status(200).json({ code: "800", message: "success" });
  } catch (error) {
    console.log(error)
    return res.status(500).json({ code: "801", message: "Error while creating review" });
  }
}
```

### Delete

```
const remove = async(req, res) => {
  //required params
  if (!req.body.id)
    return res.status(400).json({ code: "802", message: "missing arguments" });

  // params validity
  if (!mongoose.Types.ObjectId.isValid(req.body.id) || !(await Review.exists({ id: req.body.id }))
    return res.status(400).json({ code: "803", message: "invalid arguments" });

  const review = Review.findById(req.body.id);
  let seller = Seller.findById(review.userId);

  //remove from seller
  seller.reviews = seller.reviews.filter(rid => { return rid != review.id });
  await seller.save().catch(err => {
    return res.status(500).json({ code: "801", message: "unable to save changes" });
  })

  //remove from DB
  await Review.deleteOne({ id: review.id }, err => {
    if (err)
      return res.status(500).json({ code: "801", message: "unable to delete" });
  })

  return res.status(200).json({ code: "800", message: "success" });
}
```

### Read (in per i sellers e out per i buyers)

```
const getAllIn = async(req, res) => {
  const buyer = await getAuthenticatedBuyer;

  if (!buyer.isSeller)
    return res.status(400).json({ code: "807", message: "invalid user type" });

  const seller = await Seller.find({ id: buyer.sellerId });
  const reviews = (await Review.find({ "id": { '$in': [seller.reviews] } }));

  return res.status(200).json({ reviews: reviews, code: "800", message: "success" });
}

const getAllOut = async(req, res) => {
  const buyer = await getAuthenticatedBuyer;
  const reviews = await Review.find({ authorId: buyer.id });
  return res.status(200).json({ reviews: reviews, code: "800", message: "success" });
}
```

- Seller

### Create

```
const create = async(req, res) => {
  let buyer = await getAuthenticatedBuyer(req, res);

  if (buyer.isSeller)
    return res.status(422).json({ code: "", message: "unable to create, the buyer is already a seller"});

  const seller = new Seller({
    userId: buyer.id,
  });
  seller.save()
    .then(ok => {})
    .catch(err => {
      return res.status(500).json({ code: "", message: "unable to create" });
    });

  buyer.sellerId = seller.id;
  buyer.isSeller = true;
  buyer.save()
    .then(ok => {})
    .catch(err => {
      return res.status(500).json({ code: "", message: "unable to save changes" });
    });

  await Mail.send(buyer.email, 'Creazione nuovo annuncio', `Grazie per aver scelto skupply.\nA breve il t
  return res.status(201).json({ code: "", message: "success" });
}
```

### Delete

```
const remove = async(req, res) => {
  const buyer = await getAuthenticatedBuyer(req, res);
  if (!buyer.isSeller)
    return res.status(422).json({ code: "", message: "invalid account type" });

  let seller = await Seller.findById(buyer.sellerId);

  if (seller.items)
    seller.items.forEach(async itemId => {
      let item = await Item.findById(itemId);
      item.state = 'DELETED';
      await item.save().catch(err => {
        return res.status(500).json({ code: "", message: "unable to save changes" });
      });
    });

  Seller.deleteOne({ id: seller.id })
    .then(ok => {})
    .catch(err => {
      return res.status(500).json({ code: "", message: "unable to remove" });
    });

  buyer.isSeller = false;
  buyer.save()
    .then(ok => {
      return res.status(200).json({ code: "", message: "success" });
    })
    .catch(err => {
      return res.status(500).json({ code: "", message: "unable to save changes" });
    });
}
```

### Private Read

```
const getInfo = async(req, res) => {
  const buyer = await getAuthenticatedBuyer(req, res);
  if (!buyer.isSeller)
    return res.status(422).json({ code: "", message: "invalid account type" });

  const seller = await Seller.findById(buyer.sellerId);
  return res.status(200).json({ seller: seller, code: "", message: "success" });
}
```

### Public Read

```
const getPublicInfo = async(req, res) => {
  if (!req.query.username && !req.query.id)
    return res.status(400).json({ code: "", message: "missing arguments" });

  let seller
  let buyer

  if (req.query.username) {
    if (!await Buyer.exists({ username: req.query.username }))
      return res.status(400).json({ code: "", message: "invalid arguments" });

    buyer = await Buyer.findOne({ username: req.query.username });
    seller = await Seller.findById(buyer.sellerId);
  }
  else if (req.query.id) {
    if (!await Seller.exists({ id: req.query.id }))
      return res.status(400).json({ code: "", message: "invalid arguments" });

    seller = await Seller.findById(req.query.id);
    buyer = await Buyer.findById(seller.userId);
  }

  var rating = null;

  //calcolo media recensioni di ogni utente
  if (seller.reviews) {
    const rt = await Review.aggregate([
      {
        "$group": {
          "_id": "ssellerId",
          "requests: { $sum: 1 },
          avgRating: { $avg: "$rating" }
        }
      }
    ]);

    let found = false;
    for (let i = 0; i < rt.length && !found; i++) {
      if ((rt[i]._id).equals(seller._id)) {
        found = true;
        rating = rt[i].avgRating;
      }
    }
  }

  const pub = {
    username: buyer.username,
    rating: rating,
  }

  return res.status(200).json({ seller: pub, code: "", message: "success" });
}
```

- Wishlist

### Inserimento di un item

```
const insertItem = async(req, res) => {
  const data = req.body;
  let id_item = data.id;
  let user = await getAuthenticatedBuyer(req, res);

  if (!id_item)
    return res.status(400).json({ code: "502", message: "missing arguments" });

  //se l'elemento è un duplicato, questo non viene inserito
  const result = await Buyer.findOne({ _id: user._id }, { wishlist: { $elemMatch: { id: id_item } } });
  if (!result) return res.status(404).json({ code: "501", message: "user or item not found" });
  else {
    if (Object.keys(result).length === 0) {
      //item non presente nel carrello, inserimento id
      const result = await Buyer.updateOne({ _id: user._id }, { $push: { wishlist: { id: id_item } } });
      return res.status(200).json({ code: "500", message: "product added in wishlist" });
    } else
      return res.status(200).json({ code: "500", message: "product not added in wishlist" });
  }
}
```

### Rimozione di un item

```
const deleteOneItem = async(req, res) => {
  //remove an item with a defined id
  let id = req.body.id;
  let user = await getAuthenticatedBuyer(req, res);

  if (!id)
    return res.status(400).json({ code: "502", message: "missing arguments" });

  //modifica wishlist
  const items = user.wishlist;
  let id_item;
  let notFound = true;

  for (i = 0; i < items.length && notFound; i++) {
    if (items[i].id == id) {
      //item trovato
      notFound = false;
      id_item = items[i]._id;
    }
  }

  if (notFound) return res.status(404).json({ code: "504", message: "product not found" });

  result = await Buyer.updateOne({ _id: user._id }, {
    $pull: {
      wishlist: {
        _id: id
      }
    }
  });

  if (!result) return res.status(500).json({ code: "501", message: "database error" });
  return res.status(200).json({ code: "500", message: "product removed" });
}
```

### Read

```
const getItems = async(req, res) => {
  let user = await getAuthenticatedBuyer(req, res);

  //una volta trovati gli id, trovo gli articoli presenti nella lista
  let wishlist = user.wishlist;
  let articoli = [];

  for (let i = 0; i < wishlist.length; i++) {
    let result = await Item.findOne({ _id: wishlist[i].id });
    if (result) {
      articoli.push(result);
    }
  }

  //inserire nella risposta gli articoli
  return res.status(200).json({ code: "500", message: "success", wishlist: articoli, wishlist_ids: wishlist });
}
```



## 8. Documentazione API

Le API esposte all'applicazione da Skupply sono state documentate e rese pubblicamente disponibili utilizzando il modulo Swagger UI Express di NodeJS.

In questo modo tutti gli endpoint sono dettagliatamente documentati all'indirizzo <https://skupply.shop:3000/api-docs> oppure lanciando il server di sviluppo questa pagina è raggiungibile da localhost, porta 3000 al path /api-docs.

The screenshot shows the Swagger UI Express interface for the Skupply API. At the top, it displays the title "Skupply 0.1.0 OAS3" and a brief description: "Piattaforma per la compravendita di articoli scolastici". It also includes contact information: "Contact Piccin Andrea - Di Zepp Dorijan - Rossi Simone". Below this, there's a "Servers" dropdown set to "https://skupply.shop:3000 - Production Server" and an "Authorize" button with a lock icon.

The API is organized into several sections:

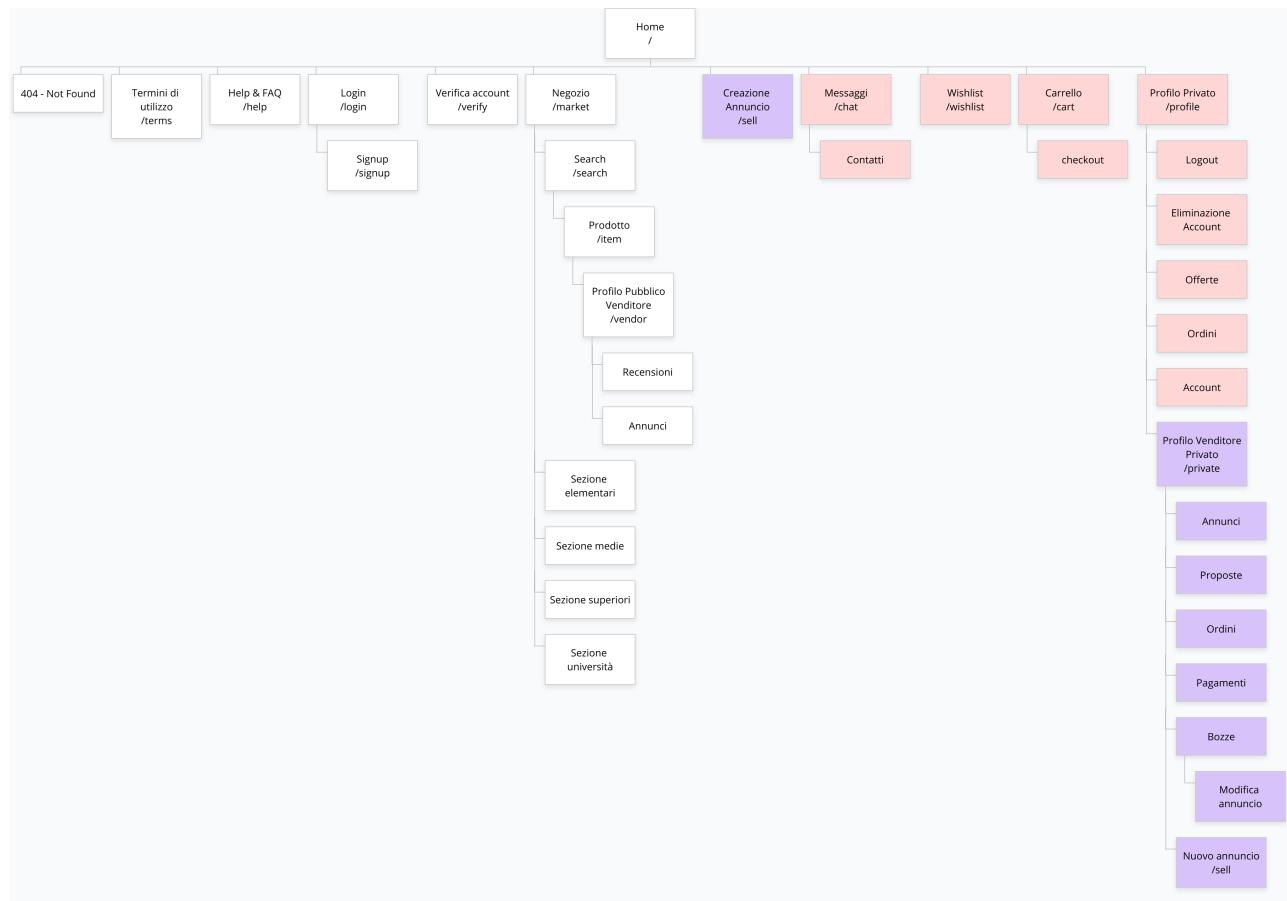
- Buyer**:
  - POST /buyer**: Creazione di un nuovo utente
  - GET /buyer**: Ritorna il profilo dell'utente autenticato
  - PUT /buyer**: Modifica dell'utente autenticato
  - DELETE /buyer**: Eliminazione dell'utente autenticato
  - GET /buyer/find/**: Cerca username
- Seller**:
  - POST /seller**: Creazione di un venditore
  - GET /seller**: Ritorna il profilo del venditore autenticato
  - DELETE /seller**: Eliminazione del venditore autenticato
  - GET /seller/public**: Ritorna le informazioni pubbliche di un venditore
- Search**:
  - GET /search**: Ritorna una lista di articoli
  - GET /search/categories**: Ritorna tutte le categorie presenti
- Email**:
  - GET /email**: Controllo di una email
  - POST /email**: Verifica di una email
- Cart**:
  - POST /cart**: Articoli carrello
  - PUT /cart**: Modifica quantità articolo
  - DELETE /cart**: Rimozione articolo

Alcune procedure richiedono un livello di autenticazione, è possibile ottenere il proprio auth token nella response della API di login ed incollarlo nel popup del bottone "Authorize" in cima alla pagina oppure sul lucchetto delle singole API.

## 9. Implementazione Frontend

In questa sezione si illustreranno le varie pagine create per il frontend del sito con una semplice descrizione delle informazioni e funzionalità presenti.

L'intero sito si sviluppa sulla seguente mappa:



Le pagine denotate da uno sfondo rosso sono pagine alle quali è possibile accedere solo previa autenticazione, mentre quelle con uno sfondo viola richiedono l'autenticazione e l'esistenza di un profilo venditore.



- Home

The screenshot shows the Skupply homepage. At the top is a navigation bar with links: Negozio, Vendi, Messaggi, Wishlist, Carrello, and Profilo. Below the navigation bar is a large banner with the text "La scuola a misura di portafoglio". A small illustration of a student wearing a graduation cap stands next to a large pencil. Below the banner, there is a call-to-action button labeled "Visita il negozio".

This screenshot shows a dark purple section of the website. It features a white box containing text about searching for deals and saving money. To the right of the text is an illustration of a person pushing a shopping cart. The title "Come funziona?" is displayed above the text.

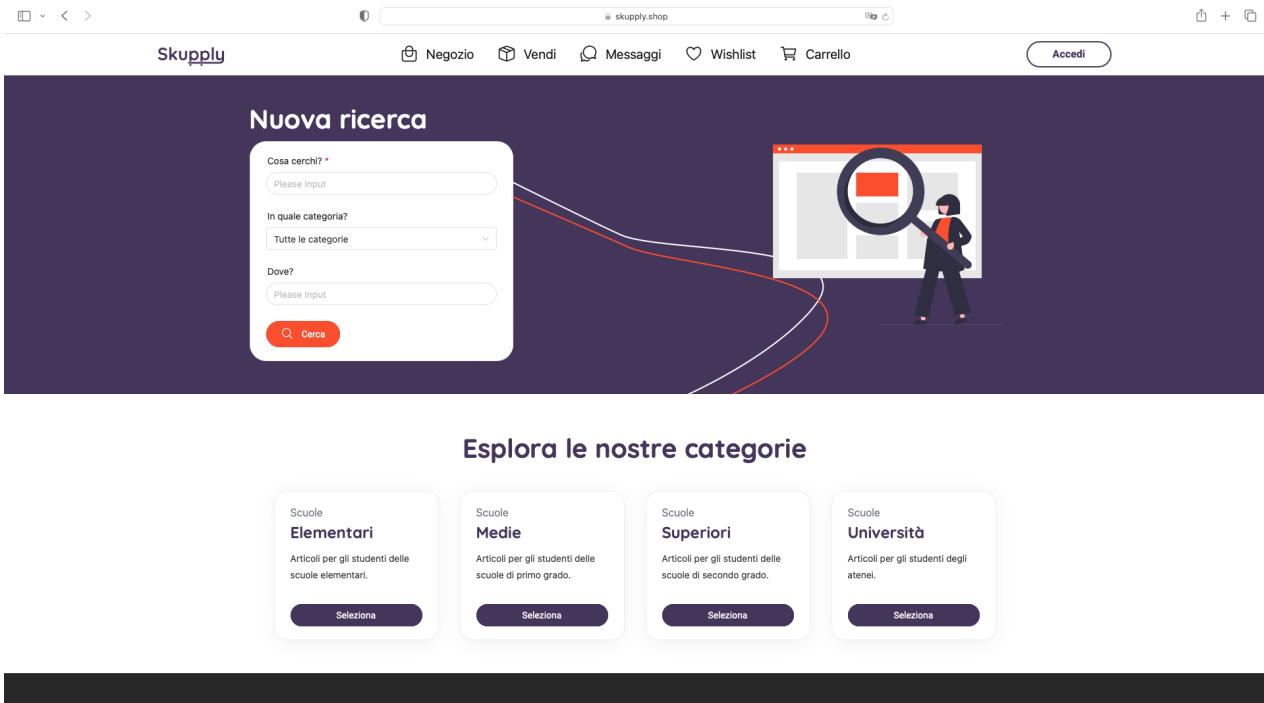
## Descrizione:

La prima pagina che appare all'apertura del sito, oltre a presentare numerosi collegamenti alle varie pagine del sito riporta uno slogan, una breve descrizione e una sintetica guida al funzionamento della piattaforma

## Collegamenti:

- Collegamenti della navbar
- Bottone di collegamento al negozio
- Bottone di collegamento alla vendita

- Negozio



The screenshot shows the homepage of the Skupply shop. At the top, there's a navigation bar with links for 'Negozio', 'Vendi', 'Messaggi', 'Wishlist', 'Carrello', and 'Accedi'. Below the navigation is a search bar titled 'Nuova ricerca' with fields for 'Cosa cerchi?', 'In quale categoria?', and 'Dove?'. To the right of the search bar is a graphic of a person holding a magnifying glass over a computer screen. Below the search bar, there's a section titled 'Esplora le nostre categorie' featuring four cards: 'Scuole Elementari', 'Scuole Medie', 'Scuole Superiori', and 'Scuole Università'. Each card has a description and a 'Selezione' button.

## Descrizione:

Home del negozio. È presente un form per iniziare una nuova ricerca e quattro card che espongono le quattro categorie di prodotto presenti nel sito.

## Collegamenti:

- Collegamenti della navbar
- Inizio della ricerca
- Collegamenti alla ricerca con filtri per categoria



- Ricerca

The screenshot shows a web browser displaying the Skupply shop website. The search bar at the top contains the word "penna". Below the search bar, there are several filters: "Ordina per" (sorted by price ascending), "Filtrati" (filter by delivery available), "Prezzo" (price range from 0.80 to 5.00), and "Valutazione" (rating from 1 to 5 stars). A single product card is visible, featuring a blue pen icon, the text "Penna Blu Trento", the price "€ 1.00", and a "€ 0.00" shipping cost. There is also a "Cuore" (heart) icon for wishlist.

## Descrizione:

Vengono elencati i risultati della ricerca effettuata che possono essere filtrati per disponibilità della spedizione, prezzo e valutazione del venditore.

È anche possibile ordinare i prodotti per prezzo (ascendente o discendente)

## Collegamenti:

- Collegamenti della navbar
- Le card dei risultati portano alla pagina del prodotto
- Il click sull'icona a forma di cuore salva/rimuove l'articolo nella wishlist



- Prodotto

The screenshot shows a product page for a 'Penna Blu' (Blue Pen) listed at € 1.00. The page includes a product image, seller information (Trento, Spedizione: € 0.00, Ritiro a mano disponibile, Quantità: 10, Condizioni: Nuovo), and purchase options (Aggiungi al carrello, Compra subito, Invia una proposta). Below the main product, there's a section for 'Info Venditore' (Seller info) showing the seller's name (Simo) and a 5-star rating.

## Descrizione:

Viene mostrato il prodotto e tutte le sue caratteristiche quali la prezzo, descrizione, posizione, possibilità e costi di spedizione e disponibilità al ritiro a mano.  
È inoltre possibile visualizzare il nome e la valutazione del venditore.

## Collegamenti:

- Collegamenti della navbar
- Possibilità di tornare alla ricerca
- Il click sull'icona a forma di cuore salva/rimuove l'articolo nella wishlist
- Aggiunta del prodotto al carrello
- Possibilità di essere reindirizzati al pagamento concludendo così l'ordine del prodotto
- Invio di una proposta d'acquisto ad un prezzo differente da quello proposto al venditore
- Visualizzazione del profilo (pubblico) del venditore
- Possibilità di iniziare (o riprendere) una chat con il venditore



- Profilo venditore pubblico

Simo Profilo Venditore

Recensioni Annunci

Penna Blu  
Trento  
€ 1.00

Quaderno  
Bolzano  
€ 2.00

Astuccio  
Trento  
€ 14.00

Simo Profilo Venditore

Recensioni Annunci

User Ottimo

Le penne migliori al prezzo perfetto!

**Descrizione:**

Sono riportate le informazioni pubblicamente accessibili di un venditore come l'elenco delle recensioni a suo carico e la lista dei suoi annunci attualmente sul mercato.

**Collegamenti:**

- Collegamenti della navbar
- Il click sull'icona a forma di cuore salva/rimuove l'articolo nella wishlist
- Aggiunta del prodotto al carrello



- Creazione annuncio

The screenshot shows the 'Nuovo annuncio' (New Ad) page on the Skupply website. At the top, there's a red info box about commissions. Below it, there's a placeholder image for the product photo. To the right, there are fields for Title (Titolo), Description (Descrizione), and checkboxes for pickup (Ritiro in persona) and delivery (Spedizione disponibile). Further down are fields for Category (Categoria), Price (Prezzo), Quantity (Quantità), Shipping Cost (Costo spedizione), City (Città), and Conditions (Condizioni). At the bottom right are 'Pubblica' (Public) and 'Salva' (Save) buttons.

## Descrizione:

È possibile riempire il form e creare una nuova inserzione che è possibile salvare come bozza (e rivedere nel proprio profilo venditore) oppure pubblicare direttamente

## Collegamenti:

- Collegamenti della navbar
- Possibilità di procedere con la pubblicazione dell'annuncio appena creato
- Salvataggio dell'annuncio tra le bozze



- Chat

The screenshot shows a web browser window for the Skupply shop. The top navigation bar includes links for Negozio, Vendi, Messaggi, Wishlist, Carrello, and Profilo. On the left, a sidebar titled "Le tue chat" lists three contacts: User00, User01, and Simo. The main area shows a conversation with Simo. The messages are:

- Salve!
- Sono ancora disponibili le penne?
- Certamente!

At the bottom, there is an input field labeled "A quale quantità è interessato?" with a red send button.

## Descrizione:

È possibile scorrere la lista dei propri contatti, selezionarne uno e scambiare messaggi di testo.

## Collegamenti:

- Collegamenti della navbar



- Wishlist

The screenshot shows a web browser window for the Skupply shop. The URL in the address bar is skupply.shop. The page header includes the Skupply logo, navigation links for Negozio, Vendi, Messaggi, Wishlist, Carrello, and Profilo, and standard browser controls. A product card for a "Penna Blu" (Blue Pen) is displayed, featuring a small image of the pen, the product name, the price "€ 1.00", and a "€ 0.00" note. Below the price are two buttons: "Aggiungi al carrello" (Add to cart) and a circular icon with a checkmark.

**Descrizione:**

È riportato l'elenco dei prodotti preferiti.

**Collegamenti:**

- Collegamenti della navbar
- Possibilità di rimuovere un articolo
- Aggiunta al carrello



- Carrello

The screenshot shows a shopping cart interface. At the top, there's a header with the Skupply logo, navigation links (Negozio, Vendi, Messaggi, Wishlist, Carrello), and a user profile link. Below the header, the main content area displays a single item: "Penna Blu" (Blue Pen) from Trento, priced at €1.00. To the right of the item, it says "Totale ordine" (Total order) and "€ 1.00". Below that, it says "di cui € 0.00 di spedizione" (including € 0.00 shipping). A section titled "Checkout disponibile" (Available checkout) shows payment method options: "PayPal" (highlighted in yellow), "Banca MyBank", and "Debit or Credit Card". A small note at the bottom right of the payment section says "Powered by PayPal".

**Descrizione:**

È riportato l'elenco dei prodotti selezionati per l'acquisto, è possibile visualizzarne sia il prezzo singolo che il totale dell'ordine.

Da qui si può procedere con il checkout ed il completamento dell'ordine.

**Collegamenti:**

- Collegamenti della navbar
- Possibilità di rimuovere un articolo
- Modifica della quantità
- Checkout via Paypal

- Profilo privato

The screenshot shows the Skupply private profile page. At the top, it says "Ciao Andrea!" with "Logout" and "Accedi al tuo profilo venditore" buttons. Below this, there's a card for a purchase of a "Penna" (pen) from "Trento" for a final price of "€ 1.00". The card includes buttons for "Tracciamento" (tracking), "Spedito" (shipped), "Conferma ricezione" (confirm receipt), and "Ricevuta" (receipt). Navigation tabs at the top include "Offerte", "Acquisti" (which is selected), and "Account".

The screenshot shows the Skupply private profile page with "Ciao Andrea!" at the top. It features three main sections: "Info personali" (Personal info) with fields for Name (Andrea), Surname (Piccin), Phone (0123456789), and Email (andrea@skupply.shop); "Info d'accesso" (Access info) with fields for Username (ap39), Password (Nuova password), and an "Update" button; and "Info di spedizione" (Shipping info) showing saved addresses: "Via Belluno 3" (Predefinito, Feltre, BL) and "Via Feltre 24" (Belluno, BL), each with an "Update" button.

## Descrizione:

La pagina è suddivisa in tre sezioni:

- Sezione offerte: è possibile visualizzare e/o ritirare tutte le proposte create
- Sezione acquisti: sono elencati gli acquisti effettuati ed il relativo stato
- Account: vengono mostrati tutti i dati personali con la possibilità di aggiornarli

## Collegamenti:

- Collegamenti della navbar
- Operazioni delle varie tab

- Profilo venditore privato

The screenshot shows the Skupply private seller profile interface. At the top, it displays "Ciao Andrea!" and navigation links for Annunci, Proposte, Ordini, Pagamenti, Bozze, and Recensioni. Below this, a product listing is shown for a "Bloc notes, A4" from Verona, priced at € 5.89, with a shipping cost of € 1.34. There is a "Ritira" (Pickup) button and a small info icon.

The screenshot shows the "Modifica prodotto" (Edit Product) form for the same Bloc notes listing. The form includes fields for Title (Titolo), Description (Descrizione), and various settings like pick-up (Ritiro in persona) and delivery (Spedizione disponibile). It also allows changing category (generale), price (Prezzo), quantity (Quantità), shipping cost (Costo spedizione), conditions (Condizioni), and city (Città). Buttons for "Esci" (Exit) and "Salva" (Save) are at the bottom.

## Descrizione:

La pagina è suddivisa in sei sezioni che permettono la gestione degli annunci pubblicati (ritiro), delle proposte (accettazione / rifiuto), degli ordini, la visualizzazione dello stato dei pagamenti ed infine la visualizzazione delle recensioni lasciate dai clienti.

Una volta ritirato un annuncio appare nella sezione bozze dove può essere modificato tramite apposito form e da qui pubblicato o eliminato.



## 10. GitHub Repositories

Per il progetto è stata creata un organizzazione GitHub ad hoc nella quali sono presenti quattro repositories:

- Docs: per la fase di sviluppo dei documenti
- Deliverables: per la consegna dei documenti una volta terminati
- Frontend: per lo sviluppo della parte grafica e dell'interrogazione delle API del backend
- Backend: per lo sviluppo del server che espone le API e ne gestisce l'esecuzione.

## 11. Deployment

Per il deployment ci siamo basati su una macchina virtuale messa a disposizione da Oracle tramite la Oracle Cloud Infrastructure sulla quale sono in esecuzione i processi di frontend (tramite protocollo HTTPS alla porta 443) e di backend (sempre via HTTPS alla porta 3000). Sul server sono state opportunamente configurate delle regole firewall e di accesso tramite protocollo SSH.

Successivamente è stato acquistato (in licenza gratuita per un periodo limitato) il nome di dominio `skupply.shop` ed è stato modificato il record DNS per puntare all'indirizzo IP statico del server di cui sopra.

Infine tramite Let's Encrypt, un ente di certificazione no profit, abbiamo ottenuto il certificato TLS necessario per elevare la sicurezza del sito all'utilizzo del protocollo sicuro HTTPS.

Il deployment sul server avviene in modo automatico tramite GitHub Actions configurate per eseguire l'installazione pulita delle dipendenze, l'eventuale build, la messa in esercizio ed il testing ad ogni commit effettuato sul main.

Se i test danno esito positivo si può procedere con l'accesso via ssh alla macchina server, l'aggiornamento dei dati e il riavvio dei server di frontend e backend.



## 12. Testing

Per il testing è stato fatto uso delle librerie Jest e Supertest, di seguito saranno riportati i principali casi di test ed il loro esito suddivisi per API.

Da sottolineare che durante lo sviluppo delle API è stato deciso di includere alcuni codici di risposta interni utili alla corretta identificazione del risultato sia che si tratti di un errore sia in caso di successo.

- Cart

Test senza autenticazione

```
test('test /cart - senza autorizzazione', async() => {
  const response = await request(app).post('/cart')
  expect(response.statusCode).toBe(403);
  expect({ code: "", message: "Invalid access token" })
});
```

Test con autenticazione

```
test('test /cart - con autorizzazione', async() => {
  const options = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json', 'x-access-token': process.env.ACCESS_TOKEN }
  }

  const result = (await fetch(`${app}/cart`, options).then(response => response.json()))
  expect(result).toMatchObject({ "code": "400", "message": "success" })

  //verifica presenza parametro cart e cart_ids
  expect(result.cart).toBeDefined();
  expect(result.cart).toStrictEqual(expect.arrayContaining([expect.any(Object)]))
  //questo test funziona nel caso in cui sia presente almeno un elemento

  //verifica che sotto l'attributo cart vi sia un array di oggetti contenente alcuni attributi degli
  result.cart.forEach(element =>
    expect(element.quantity).toBeDefined();
    expect(element.quantity).toStrictEqual(expect.any(Number));
    expect(element._id).toBeDefined();
    expect(element.state).toBeDefined();
  )

  expect(result.cart_ids).toBeDefined();
  expect(result.cart_ids).toStrictEqual(expect.arrayContaining([expect.any(Object)]))
  //questo test funziona nel caso in cui sia presente almeno un elemento

  //verifica che l'attributo quantity sia definita e che sia >=0
  result.cart_ids.forEach(element =>
    expect(element.quantity).toBeDefined();
    expect(element.quantity).toBeGreaterThanOrEqual(0);
  )
});
```



- Chat

Test recupero di tutte le chat di un utente

```
test('tests /chat - recupero chat utente', async() => {
  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json', 'x-access-token': process.env.ACCESS_TOKEN },
  }

  const response = (await fetch(`${app}/chat?username=najirod`, options)).then(response => response.json())
  expect(response).toMatchObject({ code: 200, message: "Success" })
  expect(response.chats).toBeDefined();
  expect(response.chats[0]).toHaveProperty('messages');
});
```

Test recupero messaggi da una chat

```
test('tests /chat - recupero messaggi chat', async() => {
  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json', 'x-access-token': process.env.ACCESS_TOKEN },
  }

  const response = (await fetch(`${app}/chat/message?username=najirod&contact=sic`, options)).then(response => response.json())
  expect(response).toMatchObject({ code: 200, message: "Success" })
  expect(response.messages).toBeDefined();
  expect(response.messages[0]).toHaveProperty('sender');
  expect(response.messages[0]).toHaveProperty('text');
  expect(response.messages[0]).toHaveProperty('date');
});
```



- Email

Test di validità senza specificare l'email

```
test('GET /email - Invalid arguments', async() => {
  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json' }
 };

  expect(await fetch(`${url}/email`, options).then(response => response.json())).toMatchObject({ code: 202, message: "Email argument is missing" });
});
```

Test di validità con un email non raggiungibile

```
test('GET /email - Email not reachable', async() => {
  const result = await Buyer.findOne({ $or: [{ username: 'test' }, { email: 'test@gmail.com' }] });
  if (result) { await Buyer.deleteOne({ $or: [{ username: 'test' }, { email: 'test@gmail.com' }] }); }

  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json' }
 };

  expect(await fetch(`${url}/email?email=test@gmail.com`, options).then(response => response.json())).toMatchObject({ code: 204, message: "Email not
});
```

Test di validità con una email corretta

```
test('GET /email - Valid email', async() => {
  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json' }
 };

  expect(await fetch(`${url}/email?email=wasdqwert396@gmail.com`, options).then(response => response.json())).toMatchObject({ code: 203, message: "E
});
```

Test con un email già in uso all'interno della piattaforma

```
test('GET /email - Email already used', async() => {
  await Buyer.deleteOne({ $or: [{ username: 'test' }, { email: 'test@gmail.com' }] });

  const user = new Buyer({
    firstname: 'test',
    lastname: 'test',
    username: 'test',
    email: 'test@gmail.com',
    passwordHash: 'test',
    isVerified: false,
    verificationCode: 'fedcba9876543210'
  });

  await user.save().catch(err => console.log(err))

  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json' }
 };

  expect(await fetch(`${url}/email?email=test@gmail.com`, options).then(response => response.json())).toMatchObject({ code: 205, message: "Email alre
  await Buyer.deleteOne({ $or: [{ username: 'test' }, { email: 'test@gmail.com' }] });
});
```



- Item

Test di ricerca per ID

```
test('tests /item - ricerca per id', async() => {
  const response = await request(app).get('/item')
    .query({ id: '63a031fbff52385f7b1857de' });
  expect(response.statusCode).toBe(200)
  expect({ code: "900", message: "success" })
  expect(response.body.item).toBeDefined();
  expect(response.body.item).toHaveProperty('title', "Testo introduttivo alla programmazione dinamica");
});
```

Test di ricerca senza specificare l'ID

```
test('tests /item - ricerca senza id', async() => {
  const response = await request(app).get('/item')
  expect(response.statusCode).toBe(400)
  expect({ code: "902", message: "missing arguments" })
  expect(response.body.item).toBeUndefined();
});
```

Test con item ID non valido

```
test('tests /item - ricerca con id non valido o non presente', async() => {
  const response = await request(app).get('/item')
    .query({ id: '63a0ffffffffffff' });
  expect(response.statusCode).toBe(400)
  expect({ code: "903", message: "invalid arguments" })
});
```

Test di ricerca per venditore

```
test('tests /item - ricerca items di un venditore', async() => {
  const response = await request(app).get('/item/seller')
    .query({ username: 'ap39' });
  expect(response.statusCode).toBe(200)
  expect({ code: "900", message: "success" })
  expect(response.body.items).toBeDefined();
  expect(response.body.items).toStrictEqual(expect.arrayContaining([expect.any(Object)]));
});
```

Test eliminazione di un prodotto senza essere un venditore

```
test('tests /item - ritiro item', async() => {
  const options = {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json', 'x-access-token': process.env.ACCESS_TOKEN }
  }

  const response = (await fetch(`${app}/item/retire?id=63a031fbff52385f7b1857de`, options)).then(response => response.json())
  expect({ code: "904", message: "invalid user type" })
});
```



- Login

Test di esecuzione con credenziali errate

```
test('test /login - credenziali errate', async() => {
  const options = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      email: "test@skupply.shop",
      password: "passwordErrata"
    })
  }

  const response = (await fetch(`${app}/login`, options).then(response => response.json()))
  expect({ code: "303", message: "wrong credentials", ok: false })
});
```

Test di esecuzione con credenziali corrette

```
test('test /login - credenziali corrette', async() => {
  const options = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      email: "test@skupply.shop",
      password: "passwordCorretta"
    })
  }

  const response = (await fetch(`${app}/login`, options).then(response => response.json()))
  expect({ code: "300", message: "loged in", ok: true })
  expect(response.user).toBeDefined();
  expect(response.user).toHaveProperty("id");
  expect(response.user).toHaveProperty("isSeller");
  expect(response.user).toHaveProperty("sellerId");
  expect(response.user).toHaveProperty("token");
});
```

- Order

### Test creazione e cancellazione ordine

```
test('tests /order - creazione ordine', async() => {
  let options = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      email: 'test@gmail.com',
      password: 'test'
    })
  }

  const access = (await fetch(`${app}/login`, options)).then(response => response.json())
  let token = access.user.token;

  options = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json', 'x-access-token': token },
    body: JSON.stringify({
      seller: "639f6b399b38c1bfc9633360",//id generato casualemente
      article: { id: item._id, quantity: 1 },
      price: 10,
      shipment: 0,
      trackingCode: "trackingCodeTest",
      courier: "corriereTest"
    })
  }

  const response = await fetch(`${app}/order`, options).then(response => response.json())
  expect(response).toMatchObject({ code: '0900', message: 'Success' });

  //cancellazione ordine creato
  await Order.deleteOne({ courier: "corriereTest" , seller: "639f6b399b38c1bfc9633360"});
});
```

### Test recupero degli ordini effettuati dall'utente autenticato

```
test('tests /order - ordini fatti da un utente', async() => {
  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json', 'x-access-token': process.env.ACCESS_TOKEN },
  }

  const response = (await fetch(`${url}/order/getAll`, options)).then(response => response.json())
  expect({ code: "1000", message: "success" })
  expect(response.orders).toBeDefined();
  if (response.orders.length != 0) {
    expect(response.orders[0]).toHaveProperty('buyer');
    expect(response.orders[0]).toHaveProperty('article');
    expect(response.orders[0]).toHaveProperty('price');
    expect(response.orders[0]).toHaveProperty('shipment');
    expect(response.orders[0]).toHaveProperty('state');
  }
});
```



- Review

Test recupero di tutte le recensioni associate ad un utente

```
test('tests /review - recupero recensioni utente', async() => {
  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json', 'x-access-token': process.env.ACCESS_TOKEN },
  }

  const response = (await fetch(`${url}/review/seller/id=639f6b399b38c1bfc9633360`, options)).then(response => response.json())
  expect({ code: "800", message: "success" })
  expect(response.reviews).toBeDefined();
  if (response.reviews.length != 0) {
    expect(response.reviews[0]).toHaveProperty('authorId');
    expect(response.reviews[0]).toHaveProperty('sellerId');
    expect(response.reviews[0]).toHaveProperty('title');
    expect(response.reviews[0]).toHaveProperty('description');
    expect(response.reviews[0]).toHaveProperty('rating');
  }
});
```

Test di recupero di tutte le recensioni scritte dall'utente autenticato

```
test('tests /review - recupero recensioni scritte', async() => {
  const options = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json', 'x-access-token': process.env.ACCESS_TOKEN },
  }

  const response = (await fetch(`${url}/review/out`, options)).then(response => response.json())
  expect({ code: "800", message: "success" })
  expect(response.reviews).toBeDefined();
  if (response.reviews.length != 0) {
    expect(response.reviews[0]).toHaveProperty('authorId');
    expect(response.reviews[0]).toHaveProperty('sellerId');
    expect(response.reviews[0]).toHaveProperty('title');
    expect(response.reviews[0]).toHaveProperty('description');
    expect(response.reviews[0]).toHaveProperty('rating');
  }
});
```

- Search

Test ricerca senza parametri

```
test('tests /search - no parameters', async() => {
  const response = await request(app).get('/search');
  expect(response.statusCode).toBe(400)
  expect({ code: "702", message: "missing arguments" })
});
```

Test ricerca con parola chiave

```
test('tests /search - key parameter', async() => [
  const response = await request(app)
    .get('/search')
    .query({ key: 'programmazione' });
  expect(response.statusCode).toBe(200)
  expect({ code: "700", message: "success" })

  //verifico che l'attributo articles sia presente
  expect(response.body.articles).toBeDefined();
]);
```

Test ricerca per categoria

```
test('tests /search - category parameter', async() => {
  const response = await request(app)
    .get('/search')
    .query({ category: 'università' });
  expect(response.statusCode).toBe(200)
  expect({ code: "700", message: "success" })

  //verifico che l'attributo articles sia presente
  expect(response.body.articles).toBeDefined();
});
```

Test ricerca con limiti di prezzo

```
test('tests /search - key, min price and max price parameters', async() => {
  const response = await request(app)
    .get('/search')
    .query({ key: 'programmazione', "min-price": 0.5, "max-price": 125.7 });
  expect(response.statusCode).toBe(200)
  expect({ code: "700", message: "success" })

  //verifica presenza attributo articles
  expect(response.body.articles).toBeDefined()

  //verifica di alcuni attributi del json di risposta
  expect(response.body.articles).toEqual(expect.arrayContaining([expect.any(Object)]))

  //verifica che vi sia l'attributo categories
  response.body.articles.forEach(element => {
    expect(element).toHaveProperty("categories")
  })

  //verifica che sotto l'attributo categories vi sia un array di oggetti contenente _id ed id
  response.body.articles.forEach(element => {
    expect(element.categories).toStrictEqual(expect.arrayContaining([expect.any(String)]))
  })

  //verifica che l'attributo isPublished sia presente in tutti gli oggetti
  //e che tale valore sia vero
  response.body.articles.forEach(element => {
    expect(element.state).toBeDefined();
    expect(element.state).toBe("PUBLISHED");
  })
});
```