Documentation:

✍🏻 Requeriments:

Create a web app that allows clients to import a service history associated with vehicles and customers, and store the data in a relational database.

The solution must be able to import a CSV like this:

Nombre	Telefono	Vehiculo	Patente	Servicio	Fecha
Alejandro Ibáñez	+58 (426) 345-6789	Chevrolet Tahoe	AY 123 XW	Alineacion y Balanceo	14/09/2021
ÁLVAREZ, SEBASTIÁN	+52 (81) 7654 3210	Subaru Ascent	EY 345 XW	Service 20k	14/05/2022
ÁLVAREZ, SEBASTIÁN	+52 998 234-5678	Audi Q5	BS 123 LK	Service 20k	2021-10-14
Ana González	+58 424-678-9012	Tesla	AA 001 AB	Alin. Balanc.	05/09/2022
Ana González	+58 412-678-9012	Jeep Gladiator	EX 222 VA	Alin. Balanc.	2022-10-24

Expectations for the Import Process:

- **Clean the records**
- **Enrich the data using generative AI**
- **Establish relationships**

Entities and Expected Columns

The system includes 4 entities, each with its detailed columns. Relationships are not detailed here, as I expect them to be proposed by the candidate.

Customer

- `name`
- `alias` (e.g., Federico → Fede)
- `is_company` (boolean)

Vehicle

- `brand`
- `model`
- `plate`

Phone

- `country_code`
- `number`

Appointment

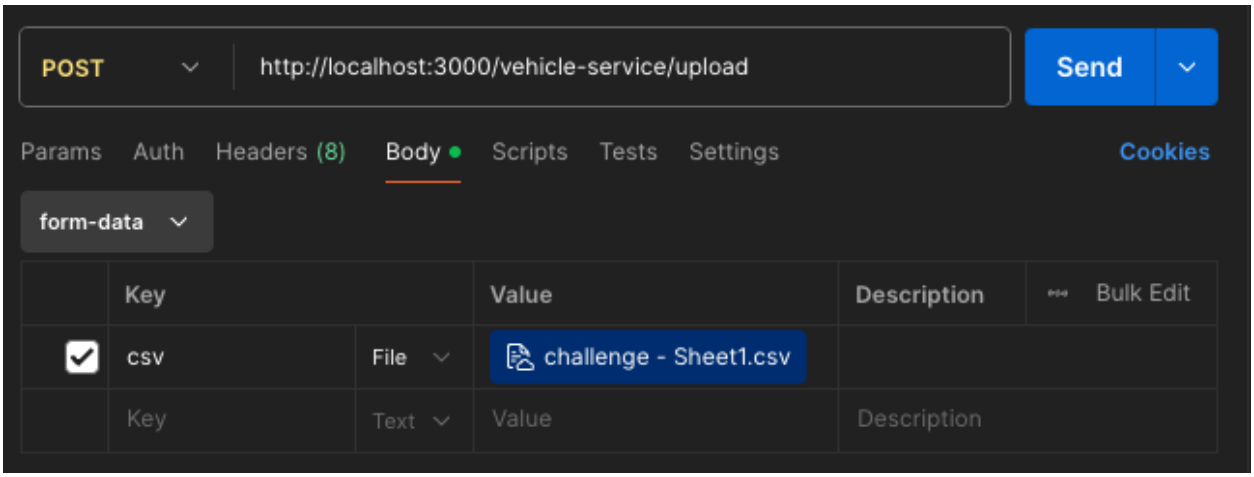
- `type` (enum)
- `date`

- *Given the number of potential edge cases, I recommend addressing the most relevant ones and leaving comments on potential improvements for the future.*

💡 My Solution:

🖌️ Assumptions & Definitions:

- FE is out of this scope. Only Backend. You can test it with postman:

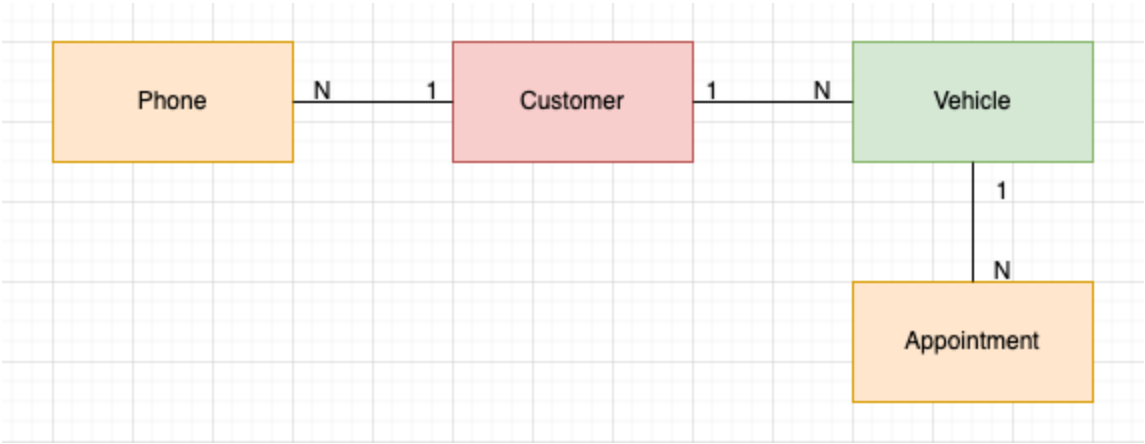


- In the input csv there isn’t a id for customer and a customer can have multiples phones and multiples vehicles, so I can’t use:
  - name + phone
  - name + plate
- Other posibility is: name + country, but I will use customer name only
- So, for the challenge scope I’ll get a **name** as id for customer.
- Vehicle service types:
  - ALIGNMENT_AND_BALANCING = 'Alignment and Balancing'
  - FIRST_SERVICE = 'First Service'
  - SERVICE_KM = 'Service each 10000 km'
  - MAINTENANCE = 'Maintenance',
- Prisma vs TypeORM
  - I choose Prisma because I don't need to perform complex queries, and it offers several advantages, such as seamless integration with NestJS, strong type safety, and an intuitive, productive development experience. These features align perfectly with my project requirements and simplify the development process.
- Stack:
  - Nest js
  - yarn
  - postgress, prisma
  - Docker
  - Integration with OpenAi Api
  - RabbitMQ
- AI Prompts:
  - Hardcoding is not an option.

- In this approach, I decided to load prompts from environment variables. This allows changes without modifying the code, though it requires a restart to take effect. However, this solution does not scale well for a large number of instructions.
- A better solution is to load prompts from a database, with tables for `prompts` and `promptLines`. Using a cache improves performance since prompts don't change frequently. This approach eliminates the need for restarts to apply changes, as we can simply invalidate the cache when updates are made.

- **Processing in Batches:**
  - Due to OpenAI's token limitations, the news is processed in batches. The batch size can be configured via the `batchSize` parameter in the `.env` file.
  - To respond quickly to requests and avoid timeouts during the processing of large batches, I opted for asynchronous processing using RabbitMQ. Each batch is sent as a separate message to the queue.

Main Entities and realtionships:



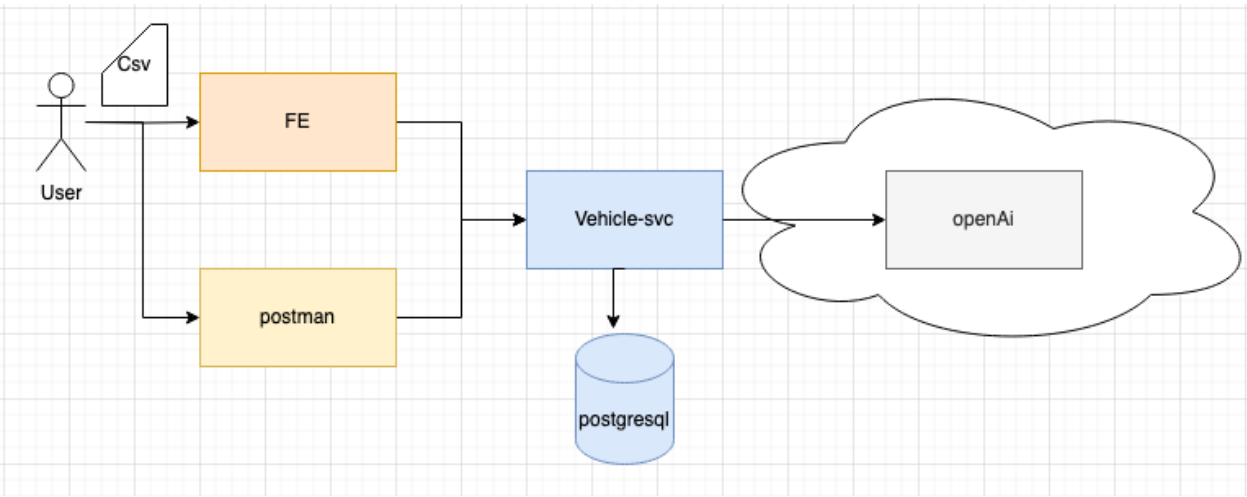
Aggregations:

- Reminder: A personalized reminder or suggestion for the customer

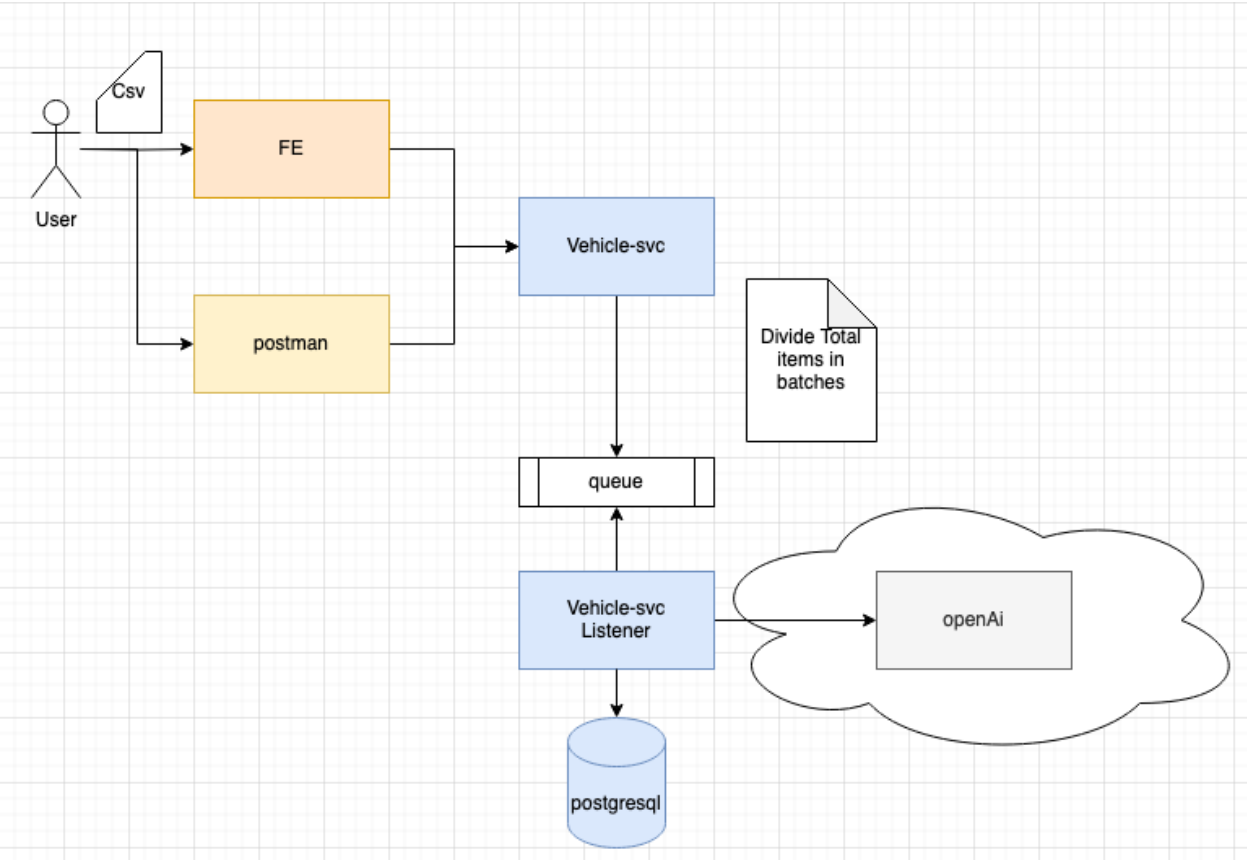
Operative:

- News: Each upload csv.
- FailureNews: A News item with any error. Usefull to check and reprocess it.

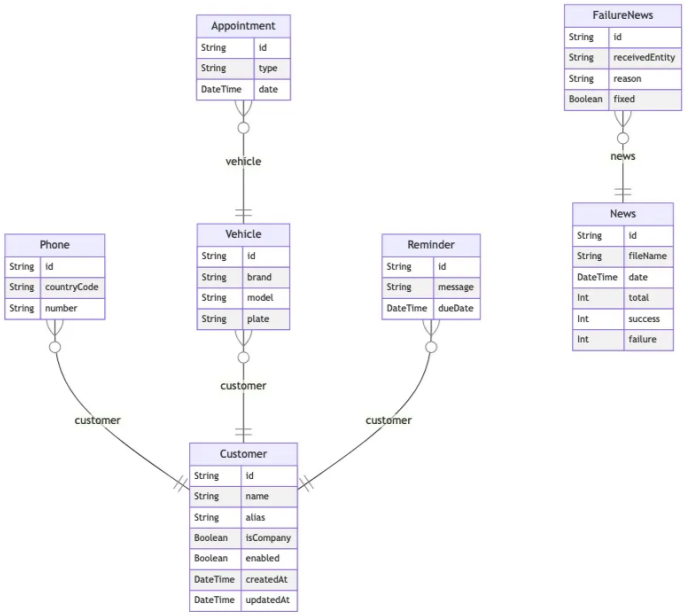
Architecture Diagram:



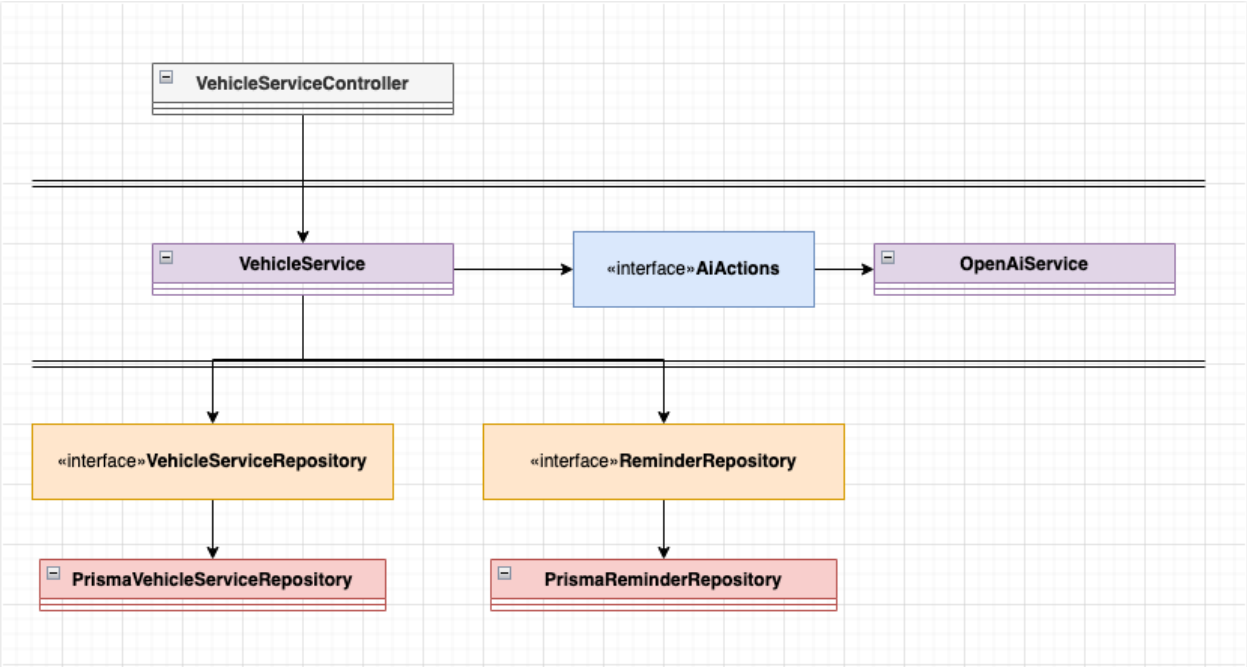
Using queue:


Csv
User
FE
postman
Vehicle-svc
Divide Total items in batches
queue
Vehicle-svc Listener
openAi
postgresql

ERD:

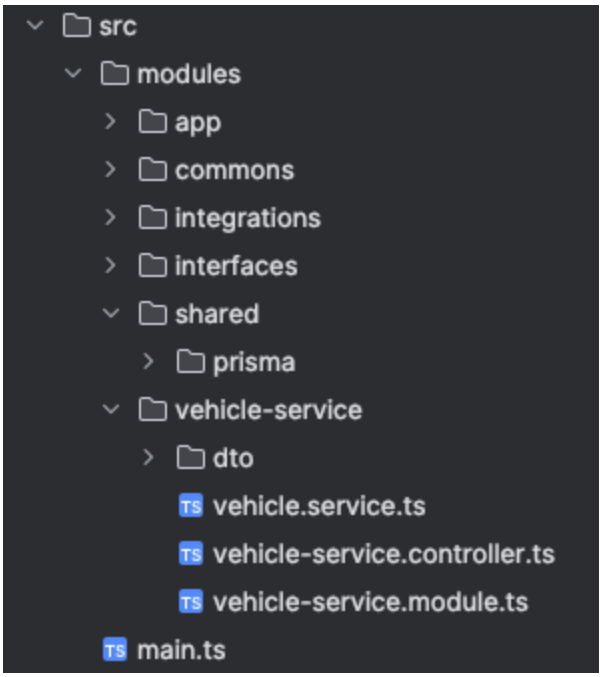

Appointment
String id
String type
DateTime date
vehicle
Vehicle
String id
String brand
String model
String plate
customer
Customer
String id
String name
String alias
Boolean isCompany
Boolean enabled
DateTime createdAt
DateTime updatedAt
Phone
String id
String countryCode
String number
customer
Reminder
String id
String message
DateTime dueDate
customer
FailureNews
String id
String receivedEntity
String reason
Boolean fixed
news
News
String id
String fileName
DateTime date
Int total
Int success
Int failure

Class Diagram (main classes):



🥇 Best Practices:

- I applied the concept of **modular structure** by dividing the application into **functional modules**. Each module represents a logical unit of the system, such as vehicle-service, app, integrations, etc. This approach promotes reusability and simplifies maintenance.



- The VehicleService handles interfaces for AI integration and persistence. This design allows for easy implementation changes, such as replacing OpenAI with another AI provider, switching from Prisma to TypeORM, or even transitioning from a relational database to a non-relational one, like MongoDB, for use cases such as aggregations (e.g., reminders).

🏗️ Iterative process:

Hera are my iterative an incrementar process to develop this challenge:

- [ ] **Feature/ping**
  #1 by lsecotaro was merged 2 weeks ago
- [ ] **OpenAi integration**
  #2 by lsecotaro was merged 2 weeks ago
- [ ] **clean and enrich data using regenerative ai**
  #3 by lsecotaro was merged last week 5 of 10 tasks
- [ ] **Feature/persistence**
  #4 by lsecotaro was merged last week 5 of 10 tasks
- [ ] **clean code**
  #5 by lsecotaro was merged 5 days ago 4 of 10 tasks
- [ ] **Enrich data with reminders**
  #6 by lsecotaro was merged 3 days ago 5 of 10 tasks
- [ ] **Upload from csv file and process by batches**
  #7 by lsecotaro was merged 2 days ago 6 of 10 tasks
- [ ] **Implements rabbitMQ**
  #8 by lsecotaro was merged 5 hours ago 7 of 10 tasks
- [ ] **Add unit test for vehicle.service.ts**
  #9 by lsecotaro was merged 19 minutes ago 5 of 10 tasks

🚧 Pending for the next version

- Implements news and failure news item persistence
- Optimize request to open Ai and persistence
- Use interface for RabbitMQ implementation
- Improve and add more unit tests
- Add nest service to docker-compose