WEB DAY 2021
04 MARZO • ONLINE CONFERENCE
A TALE OF FINANCE AND GRAPHQL
PAMELA GOTTI
GIULIA TALAMONTI
CREDIMI SPA
Credimi

- Italian startup based in Milan born in 2016
- Leader digital lender in continental Europe
- 1.5B € loans
Highlights

Time to market for new features decreases
Developers focus on delivering business value
What we got wrong
What we want to improve
Once upon a time...
Once upon a time...
Once upon a time...
Once upon a time...

Should we run away as far as possible?

Definitely
Developers Needs

Go fast
Quick release cycles to bring features to our customers

Versatility
Bringing a new feature live should be easy

Focus on value
Developers should focus on core features, not boilerplate
Credimi 1.0
Problems with Credimi 1.0

- **Boilerplate**: At API and Nginx layer
- **Authorization**: Field level authorization not easy to achieve
- **Heavy APIs**: Representation of some resources required heavy computations on the backend
- **No schema**: No easy way for frontend and backend to share schema
- **Frontend stack maintainability**: Suffers of a low developer experience and so maintainability
In the meanwhile...

Credimi was in continue evolution
Credimi 1.1
Credimi 1.1
Credimi 2.0
Opportunity

CQRS
Clearly separate concerns while obtaining potentially different read models for each use case

Event sourcing
Exploit Kafka as event collector to create a read side database

Modernize the stack
Take advantage of GraphQL to solve our issues with boilerplate, authorization, heavy queries
Modernize the stack
Takes advantages of React community and Apollo ecosystem to improve maintainability

Aggregated data sources
One place to get the data the frontend needs (read side, legacy api, external datasources)

Code generation tools
Exploits GraphQL schema to generate TypeScript types
Exposing the read side

Seamless integration
Dockerized tool easy to set up and to integrate with the existing architecture

Easy to work with
For both backend and frontend

Great community
Transparency and easy to work with
Credimi 3.0
New frontend stack

- Integrated toolchain
  Several toolchain available out of the box for every need

- Smooth transition features
  The old stack hosts the new one (with iframe) for a smooth transition

- Graphql Gateway
  Using Apollo Server to stitching severals schema into one
Let's take a look at the developer experience now
What are the involved entities
Which properties are exposed
Who can see those properties

The developers agree on the resource Graph

```graphql
scalar VatCode

type Contact {
  name: String!
  lastName: String!
}

type Company {
  vatCode: VatCode!
  contacts: [Contact]
}
```
The backend developer brings up a new environment

- A new namespace on k8s with all the needed microservices
- A dedicated PostgreSQL database
- A Hasura instance
Hasura insights
The backend developer applies to the database any migration if needed

```
CREATE TABLE companies (id SERIAL PRIMARY KEY, vat_code VARCHAR(11))
```
The metadata are then exported and versioned.

Every new development can check the validity of both migrations and authorization.
The frontend developer can start developing with mocks as soon as the schema from Hasura is ready:

- We can plug Apollo to the running Hasura to fetch the schema
- We exploit Apollo mocks to get data based on the schema
- If anything needs to change they can act on Hasura UI and then put those changes under versioning
export async function getApolloServer() {
    const originalSchema = await buildSchema()
    const schema = await getMocks(originalSchema)
    const asyncDataSources = await getAsyncDataSources()

    const apolloServer = new ApolloServer(
        introspection: process.env.APP_STAGE => AppStage.production,
        debug: process.env.APP_STAGE => AppStage.production,
        schema,
        extensions: () => new Logger(formatErrorWithContext),
        dataSources: () => ({ ...getDataSources(), ...asyncDataSources })),
    )
Learnings

Better domain comprehension
The exercise of creation of the Graph has improved our big picture vision

More focus
No boilerplate, focus on delivering value

Time to market improved
Hours, not days, to ship features changes

Performance improved
APIs respond in ms, not seconds

Authorization improved
Easy control on who can access what
What we got wrong

Apollo not on steroids
As a result of CQRS need of a more complex management of reads/write

Opinionated framework
Keeping workaround when the framework catches up

One model to rule them all
The Q in CQRS got a wrong twist

Unmanaged solution
Hasura keeps evolving, we are not
Next steps

Apollo as its best
Keeping exploiting optimistic UI

Anticorruption layer
Separating Hasura's models from the outside world

One model per use case
For real

Managed solution
Going Hasura cloud
Any questions?