REAcount progress report*

Jesper Kiehn, Jesper Eklund[1,†]

1Softfuturer,Copenhagen

Abstract
A progress report on REAcount, a REA based system, web based ERP system

Keywords
Ontology, REA, web based ERP system, Svelte, ASP .NET, C# , Implementation

1. Introduction

This paper is a progress report for the system REAcount. This system is based on the REA[1] [2] ontology and is an attempt at implementing REA in a system to be used by users with no prior knowledge of accounting or REA using a web browser. The system can be installed on a server or in the cloud on the Azure platform.

The system is planned to be used by users with little or no accounting knowledge and needs to adhere to all the legal requirements of such a system. It needs to be user friendly and assists users enough so they can do the necessary daily operations without much education. The system hence needs to implement the look and feel of other such systems without implementing double entry booking principles. It should provide enough information during the user tasks, so that users can take the decisions they need to while in the process.

The UI needs to be able to guide the users enough so the users can figure out how to do the operations they want to do, for registration of activities that influence the daily operations in the company they work for or own.

Our main motivation for using REA is the belief that REA is more intuitively understandable for non-accountants, who just want to do their daily tasks than regular double entry ledger-based systems. The system must in the end deliver, or at least provide the basis for, the required accounting calculations and reports needed for legal requirements and management control reports. The system must therefore hide, as much as possible, difficult things like costing, reconciliations and periodization. The belief is that this can be done by procedures that implement the most common ways of doing these things with a minimal setup.

2. Background

The REA model differs from a double entry system in what is considered first class objects in the system. In a double entry system, this is accounts, ledgers and postings, in REA it is Resources, Events and Agents and the relations between these entities.

Because this seems to be the first system implementing REA in this way many things are uncharted land.

The system needs to be very user friendly and can't be too different from other ERP systems. We can be different in only so many ways. The hope is to come out with a version aimed at small handy men and craftsmen who are selling services and have a limited stock of goods. Even though these kinds of business have limited items in stock they buy most of the goods from big

Presented at VMBO2025

companies and we must make sure the purchase and handling of these relations to the suppliers are very easy to work with and integrate to.

3. UI design

There are so many implementation choices to be made when implementing REA, that it is very difficult to prescribe one way of doing it. In REA some concepts are fixed and mandatory, and they should drive interoperability.

However, some concepts are needed beyond the fixed concepts. Take stockflow for example are Use/Consume/Produce and Give/Take subtypes or subclasses? What are the properties of addresses? An address can look very different around the world. Is an address a specific kind of location? Is geolocation a part of an address or a separate type of location? How do we implement multi-currency support? All these questions must be answered before we can have one standard specification that guarantees interoperability.

Another way is to make one’s own implementation and then map/from to the interoperability standard when needed. In that way one almost must implement the system 3 times.

Our take is to do our own implementation and then use as much standards as We seem fit and then map when We are forced to do that. We will then modify our implementation when standards become available.

An example is that we are using open maps for addresses, but it does not understand what an apartment is and that is a problem.

We need to have a REST interface for the UI and that also needs knowledge of the datatypes We use and some of them are not known to REA. The geolocation datatype has the ability to read the current geolocation from the device and so on. Another simple example is a date datatype which means that the user can select the date from a dropdown.

3.1. Relationships

Because we don’t want the UI to expose our different view on relations, we make sure to hide that we consider relationships to be first class objects. We do this by folding the relations in one of the objects that has a relation to the other object. This is done in the UI and we can choose to do the relationships as objects in the code or not.

The 1:many and 1:one relationship are folded into the many or the one side of the relation.

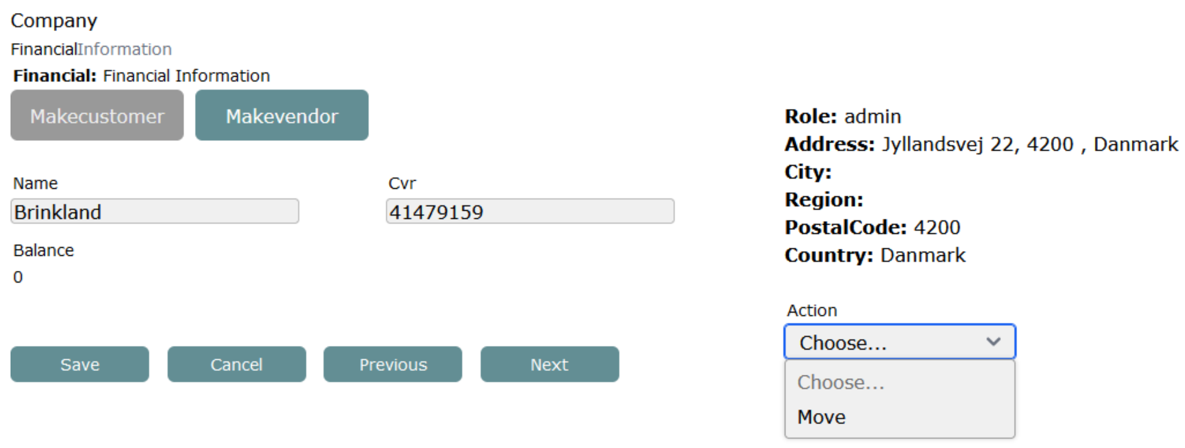
Many to many relations are not folded even when they have no properties. We, however, also try not to expose the users to many to many relations, so often these will be hidden behind actions that perform logic according to some strategy. We also try to make CQRS so there will often not be a direct field to update a property but an action that does the update.

This is what we believe is the smart choice but we must do user evaluations of this implementation strategy.

3.2. Iterative Process

We began by making a CRUD system, Create Update Delete, but for bigger systems this will become difficult to manage and make user friendly enough. Hence, we have started a transition towards Command Query responsibility segregation (CQRS) where all changes are done through actions. These actions can then lead to effects through events and these events are then stored. So instead of just editing a part of an address the user decides that the company has moved. And that command (with parameters) is then sent and that will have the effect of storing the new address but also the reason and time and user doing that change because the company moved. If the system is implemented as a pure CQRS system, then there will be a trace of every change and also the possibility to do simulations because every action and the effects could be reversed based on the state of the system/actions.

The UI differentiates between pure action with no parameters and Subviews that are actions with parameters, and perhaps a separate view with relevant information. The name Subview is subject to change, but it does capture the current UI behavior as it is shown besides the form and shows a subset of the attributes of the entity.



Financial: Financial Information

Name: Brinkland

Cvr: 41479159

Balance: 0

Role: admin
Address: Jyllandsvej 22, 4200 , Danmark
City:
Region:
PostalCode: 4200
Country: Danmark

Action: Choose...

A screen shot showing actions (Make Customer and Vendor), two tabs (financial and information) and a Subview for changing the address of the company by registration of a Move.

3.3. Balancing user expectations with the new system design

Another area where we hide that the system is a REA based system is that we sprinkle information around the UI that helps the user. So, we do show things that are considered reports to the user. Such as calculated Quantity on stock and balances on Agents and Resources.

We also need to show errors and warnings for entities and actions. Some states of entities should be fixed before continuing and this can be shown as a warning on the entity.

The last thing we hide is that the database uses Id's on all entities, but we don't show those in the UI even if the UI knows and uses these. To the user we show the names and types or other attributes that the user thinks are the keys.

The architecture is a 3-tier implementation with a UI frontend with limited logic, a web frontend with business logic and a (or multiple) database backend. Most web APIs need authentication, so the UI supports login. The UI does not (yet) support creation of users with roles, but roles are supported yet not fully implemented in the UI.

We will probably also need some kind of workflow in the UI but currently this has not been designed.

4. What we have done

For now, we only support a web browser view of the UI, but the plan is to support multiple UI factors such as phone and/or tablets.

We support several widgets such as Statistics with navigation and filtered views of entities and graphs based on multiple entities.

4.1. Semi-automatic page layout

We support generation of a few selected fixed layouts of pages but do allow the programmer/designer to make their own layouts and components. We envision a template driven page designer which should support most common use cases out of the box. We have made several prototypes for different page layouts that show that this is a workable vision. The goal is to provide the most common components and make the UI configurable but extendable if the components are not sufficient.

4.2. UI Components

We have implemented components for table (a selection of instances of the same entity type) and a form (a single entity) , a graph and a calendar view with several options to support time-based reservations and some small widgets to support basic statistics. A page can then contain more than one component, and these components can be linked. We also support tabs so we can divide the table and forms into subpages and hence provide more information in the same screen space.

The Dashboard is a special kind of page that contains many widgets and graphs and filtered tables. We have done some prototyping for grouping and support for hierarchies but do not consider them production ready yet. The current view is reflected in the url so it is possible to send the URL via e-mail/use it as a link.

4.3. Navigation

We support the most basic navigation in the UI. We support lookups and navigating to the lookup entity (with filters). We can filter from a graph to the entity(entities) when drilling down from a graph. We support filters defined by user or by the system. We support changing the filter based on current entity and can show related entities based on current record. We support master/detail views for orders. We support dynamic editable properties based on the current entity or by server calculation that can be based on any calculation. We support the same calculations for functions being applicable and hidable for UI parts.

There are probably more properties and more complex UI logic we have not yet implemented but think that we have covered most if not all needed for now.

4.4. Translation

We support translations to different languages, but we don't support switching text direction, so Arabic and Japanese needs to wait until this is supported. Sorting is done on the server without regard for user language, so sorting is based on the default implementation language – which is (Us-)English. This could change in the future

5. Current work items

We probably want to go to full CQRS and implement every functionality as CQRS and with full traceability. This is ongoing.

5.1. Long running processes

Long running processes need special attention when doing asynchronous operations. Generating large documents or lots of transactions takes time and should not be done while the client waits for the result. This should be done in a worker thread using the outbox pattern and a message bus. We have not yet implemented this fully.

5.2. Incremental processing versus one large computation

Some of the reports can take quite a long time to compute and hence we would like to do them incrementally. An automated process/compilation from the one-time computation to incremental computations seems possible and we will be considering how to do it in the future.

Some kind of lazy evaluation is probably the way forward. For now, we try to manually break down the computations in independent steps and do trigger these in the code.

This is quite a large task, and we don't think this will be solved in general for the foreseeable future even if this in principle is (relatively) easy.

5.3. Challenge UI

Currently we only add new entities when we want to challenge the UI. The goal is to make the process of adding new entity types as smooth as possible by having a based object type to inherit from. Currently we miss quite a lot of the necessary REA based types but do not think it will be an issue implementing these. We have done lots of prototypes and are confident that we can do this without raising new UI issues.

5.4. Workflow

The only issue we foresee is that we don't really have a grasp on processes and how to implement workflows. This is an area of research, and any input here would be appreciated.

6. Architecture and runtime changes

We are developing the UI in Svelte which we have very good experiences with.

6.1. Metadata-driven UI and DB

The goal is a UI that is metadata driven by a definition of the entities that can drive the UI if the components live of this metadata. We don't want to have to code business logic separately for the UI. The current metadata format is quite simple but might be more complex in the future. The hope is that it will still be possible to generate this metadata format directly from the entity designed in the backend with a few extra attributes.

The database is generated from the same metadata design with hopefully no extra attributes by using Entity Framework (an ORM from Microsoft).

The hope is that even the database can be made / updated at runtime when/if the processes/entities change.

We are not too confident about how to solve this at runtime, but we can live with a solution that requires a restart of the server when the processes change for quite some time. We are, however, confident that we can change the UI at runtime with little impedance, if it does not require new components.

The architecture is a monolithic CRUD/CQRS partly event sourced hybrid, but we are aiming for a clean CQRS implementation that can evolve into a microservice architecture and try to make design decisions that support this transition. Most of our efforts have been in implementing the UI as that has been the biggest challenge for us. We believe that the foundations we have now enable us to move forward with the desired architecture with ease.

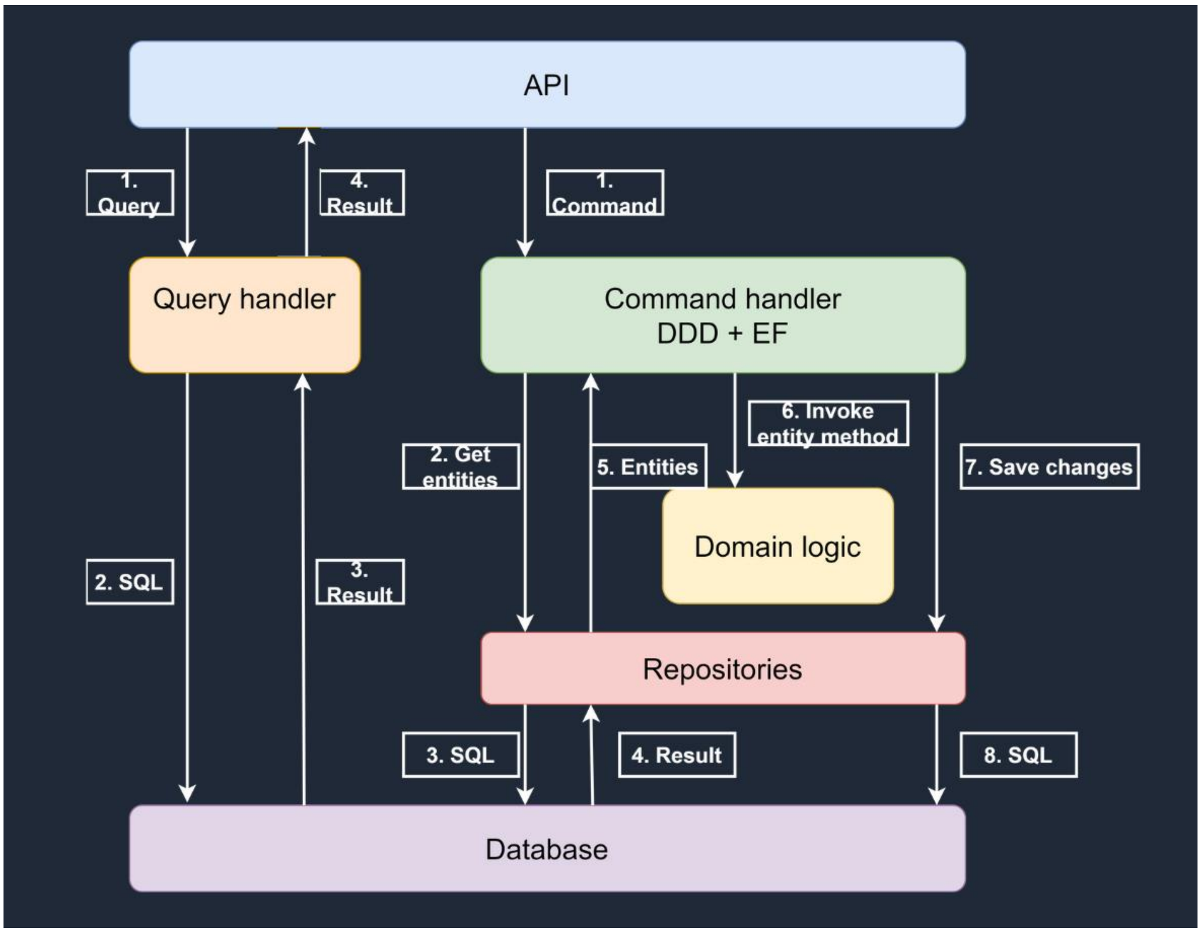


Illustration of the CQRS dataflow we currently are implementing[3].

7. Future work

We need to implement a lot of REA base functionality in the form of base objects that can be extended by inheritance/decorations. Here we believe that the foundational ontology from Sowa will help us avoid design mistakes. We need to further enhance the navigation in the UI. We need to ensure the basic reports needed are supported. We need to extend the basic chart of account functionality perhaps with some choices about the structure of the chart of accounts without giving too much freedom.

We need to build all the necessary compliance functionality. We need to implement quite a lot of integrations into banks and governmental systems. We need to make the system complete CQRS + event sourcing and develop all the commands needed. We also need to ensure that the system can support multiple companies and scales to support the needed data requirements and performance. Lastly there is a huge task in verifying the requirements and the UI with end users.

8. Conclusion

A complex web application is quite a big endeavor and for us it has been quite a learning experience. The nature of the design task is very iterative, and the different concepts need to fit together. This can be very time-consuming because we have not been able to do a big upfront design for the UI and backend. The backend has also presented some design challenges. The need for the UI to be metadata driven has also presented its own design challenges. We are quite proud of the solution we have found but it has taken quite a lot of time to get to this solution. There are still quite a lot of tasks that perhaps more experienced web developers would already have done but we have postponed until we saw the real need for it. Luckily, we have a lot of experience with

UI design and REA between the founders so only web development is new to us. We have learned a lot and have found some quite good sources of inspiration for the coming tasks.

Acknowledgements

I would like to thank Lars Hammer, Alastair Paton and Bill McCarthy for discussion on REA and implementation/design.

References

[1] https://www.williamemccarthy.com.

[2] https://aaahqbookstore.org/catalog/book/rea-accounting-model-accounting-and-economic-ontology

[3] Milan Jovanovic https://www.milanjovanovic.tech/

A. Online Resources

ASP Net: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet
Svelte: https://svelte.dev/blog/svelte-5-is-alive
CQRS: https://www.geeksforgeeks.org/cqrs-command-query-responsibility-segregation/