JJ's Reference Architecture

Author: Jan-Joost van Zon
Date: December 2014 – July 2017
[Under Construction]

Appendices

Contents

Appendix A: Layering Checklist

This checklist might be used if you want to bulk-program the architecture for an application by going through all the layers one by one, or if you want to build a feature and make sure you have not fogotten any technical issues.

- Data: Database structure
- Data: Data migration
- Data: Entity classes
- Data: Repository interfaces
- Data: Default repositories
- Data: NHibernate mappings
- Data: SQL queries
- Data: NHibernate repositories (optional)
- Data: Other repositories (optional)
- Data: Other mappings (optional)
- Business: LinkTo
- Business: Unlink
- Business: Cascading: DeleteRelatedEntitiesExtensions
- Business: Cascading: UnlinkRelatedEntitiesExtensions
- Business: Enums
- Business: String Resources
- Business: EnumExtensions
- Business: Validators
    - Delete Validators too
    - Warning Validators (optional)
- Business: SideEffects
    - SetDefaults SideEffects too
- Business: Managers
- Business: Extensions
- Business: RepositoryWrappers

- Business: Dtos (optional)
- Business: Calculations
- Business: Visitors
- Business: Factories (optional)
- Business: Api (optional)
- Business: EntityWrappers (optional)
- Business: Other helpers (optional)
- Presentation: ViewModels
    - Item ViewModels
    - List item ViewModels (some may only need IDNameDto, no ListItem view model)
    - List ViewModels
    - Detail ViewModels
    - DocumentViewModel (optional)
- Presentation: ToViewModel
    - Singular forms
    - WithRelatedEntities forms
    - ToListItemViewModel
    - ToScreenViewModel
    - CreateEmptyViewModel (not every view model needs one)
- Presentation: ToEntity
    - Singular forms
    - WithRelatedEntities forms
    - From screen view model
- Presentation: Presenters
    - List Presenters
    - Detail Presenters
    - (Edit Presenters)
    - Save methods in Detail (or Edit) Presenters.
- Presentation: Views (Mvc / UserControls…)
    - List Views
    - Detail Views
    - Main View (optional)

Appendix B: Knopteksten en berichtteksten in applicaties (resource strings) (Dutch)

Er is een bepaalde structuur waar binnen we werken voor knopteksten en meldingen in onze applicaties. De hele bedoeling is maximale herbruikbaarheid, minimaal vertaal werk en correcte teksten. Dat doen we door hele algemene teksten op plaats X te zetten, zo veel mogelijk domein termen op plaats Y, en alleen wat er dan over is, komt in specifieke projecten te staan. Dit kan het verschil betekenen tussen 100'en of 10000 teksten.

Resources worden op dit moment overal neergezet waar ze niet thuis horen, met verkeerd hoofdlettergebruik en verkeerde interpunctie. En op andere plekken worden resources gewoonweg niet gebruikt en staat alles hard op 1 taal.

Hier moet secuurder mee om worden gegaan. Van ontwikkelaars wordt verwacht zowel de Nederlandse taal als de Engelse taal in de resource files te zetten. Bij twijfel over Engels, vraag het een collega.

Hoofdletters, interpunctie, spelling

Hoofdlettergebruik etc. is conform de taalregels van de betreffende taal.

1) Resources kunnen hele zinnen zijn. Die moeten correct geschreven zijn: In het Nederlands begint dat met een hoofdletter en eindigt het met een punt, tenzij het een vraag is, dan met een vraagteken. Blijkbaar is het nodig om dit aan te geven, want het gebeurt vaak niet goed.
2) Voor Engels gebruiken we de Amerikaanse spelling.
3) Namen van properties, classes en andere titels zoals knopteksten zijn in het Nederlandse zijn als volgt: "Links in artikel", dus alleen beginnen met een hoofdletter. En dus geen punt erachter.
4) Namen van properties, classes en andere titels zoals knopteksten in Engelse titels doen we als volgt: "Table of Contents", dus alle woorden beginnen met een hoofdletter, alleen onbelangrijke woorden zoals 'in', 'and', etc. in kleine letters.
5) Er is dus een verschil in hoofdlettergebruik tussen volzinnen en losse titels.

Assemblies

Resources worden met de assemblies meegecompileerd*.

Termen worden zo veel mogelijk hergebruikt. Daarom zijn er plekken bedacht waar de termen thuis horen. Je moet in deze volgorde op zoek naar een resource die misschien al bestaat: (Update: De hoeveelheid verschillende plekken waar resources staan is een zwakte van dit ordeningssysteem, omdat het verwarrend kan zijn. In toekomstige oplossingen is het wellicht een idee om resource teksten mmer op één centrale plek te zetten. Dubbelzinnigheid van termen in meerdere domeinmodellen is daarbij wellicht meer een uitzondering dan een regel, waar omheen gewerkt kan worden.)

1) 'Save', 'Close', 'Edit', etc. staan in Framework.Resources, toegankelijk via de *CommonResourceFormatter* class.
2) Validatiemeldingen uit Framework.Validation, toegankelijk via de *ValidationResourceFormatter* class.
3) CanonicalModel: een tussenmodel voor uitwisseling van gegevens tussen verschillende systemen, toegankelijk via de *CanonicalResourceFormatter* class.
4) Business layers bevatten alleen vertalingen voor de overige teksten die niet in het canonical model staan.
5) Ook teksten die niet direct domeintermen zijn, maar wel in applicaties worden gebruikt op plekken waar het gaat over een bepaald business domain, mogen in de business layer, zijn resources gezet worden.
6) Presentation layer bevat over het algemeen geen teksten. Die zetten we in de business layer: we hebben al genoeg plekken waar we resources neerzetten.

Tips

1) Gebruik van placeholders zoals {0} is toegestaan, maar dan moet je wel een class erbij maken, die de placeholders vervangt. Zie Framework.Resources voor een voorbeeld. Het is dan verstandig om de resources zelf internal te maken en alleen de class die de placeholders vervangt public te maken. Kijk echter uit dat je het daarbij geschikt houdt voor meerdere

talen, want een creatief met placeholder opgebouwde resource string werkt al gauw niet voor een andere taal.

2) Negeer dat de beschreven werkwijze kan resulteren in berichtteksten met hoofdlettergebruik zoals: 'Het **Ordernummer** is niet ingevuld bij de **Bestelling**.' Als we dit aanpakken, doen we dat met een algoritme, niet met nog meer resources.
3) Het is verstandig om teksten in de applicatie algemeen te houden. Dus bijv. 'Artikelen', i.p.v. 'Artikelen in dit boek'. Dit scheelt vertaalwerk. Ook dit is verstandig: 'Artikel 1: Naam is verplicht.' Daarbij zijn de teksten 'Artikel', 'Naam' en '{0} is verplicht.' waarschijnlijk allang vertaald. Door de 'dubbele punt' notatie aan te houden ('Artikel 1:'), voorkom je het verwerken van de term in een zin, wat voor iedere taal een compleet andere grammatica kan zijn, wat voorkomt dat er complete volzinnen vertaald moeten worden.

** Resources staan niet in een database, omdat het applicaties trager maakt en kun je de code niet draaien op omgevingen waarbij je geen toegang hebt tot de database. In sommige situaties kun je niet eens compileren zonder toegang te hebben tot een specifieke database. Bovendien geeft mee compileren van resources in specifieke projecten ons de mogelijkheid dubbelzinnige termen anders te vertalen per business domein.*