**Μηχανική Λογισμικού για Εφαρμογές Μεγάλης Κλίμακας  
Project 2024-2025 : Συνταγές**

**Ομάδα 1**

**Γεώργιος Βώβος,Νικόλαος Βώβος,Ιωάννης Σανιδάς**

**Π1: Αντικειμενοστρεφής ανάλυση και σχεδίαση με την UML**

Αυτό το παραδοτέο θα είναι μία αναφορά σε μορφή DOCX, DOC ή ODT στην οποία θα περιέχονται UML διαγράμματα ως εικόνες και επεξηγήσεις για τα διάφορα μοντέλα του συστήματος. Η αναφορά αυτή θα έχει τις ακόλουθες ενότητες:

***Π1.1. Διάγραμμα περιπτώσεων χρήσης:   
a. Δημιουργείστε ένα διάγραμμα περιπτώσεων χρήσης για το σύστημα.  
 Δώστε συνοπτικές περιγραφές των περιπτώσεων χρήσης. (5%)***A diagram of a recipe

Description automatically generated

Όπως βλέπουμε παραπάνω, ο χρήστης μπορεί να κάνει τις παρακάτω λειτουργίες

* Δημιουργία μιας νέας συνταγής
* Διαχείρηση Συνταγών
  + Διαγραφή  
    Εδώ, ο χρήστης πρώτα πρέπει να επιβεβαιώσει την διαγραφή.
  + Επεξεργασία

Και οι δύο παραπάνω λειτουργίες, προϋποθέτουν ότι η συνταγή υπάρχει ακόμη στο σύστημα, οπότε πρέπει να γίνει ο σχετικός έλεγχος

* Εκτέλεση Συνταγής  
  Όπως και στις παραπάνω περιπτώσης, πρέπει να είμαστε σίγουροι ότι η συνταγή υπάρχει ακόμη στο σύστημα, γι’ αυτό υπάρχει η ίδια συσχέτηση με το «CheckState»

***b. Αναλύστε πλήρως με το πρότυπο καταγραφής απαιτήσεων χρήσης με μορφή πίνακα κειμένου (πίνακας Περίπτωσης Χρήσης) μία περίπτωση χρήσης της επιλογής σας συμπεριλαμβάνοντας και όλες τις επεκτάσεις της. (5%)***

|  |  |
| --- | --- |
| **Τμήμα Περίπτωσης Χρήσης** | **Σχόλια** |
| Όνομα Περίπτωσης Χρήσης | Επεξεργασία Συνταγής (Edit Recipe) |
| Εμβέλεια | Το σύστημά μας |
| Επίπεδο | Πρωτεύων διεργασία |
| Πρωτεύων Χειριστής | Μάγειρας |
| Ενδιαφερόμενοι και ενδιαφέροντα | Μάγειρας: Θέλει να επεξεργαστεί τα στοιχεία μιας συνταγής και οι αλλαγές του/της να αποθηκευτούν επιτυχώς στο σύστημα. |
| Προσυνθήκες | Η συνταγή υπάρχει στο σύστημα και δεν έχει διαγραφεί από κάποιον άλλον χρήστη |
| Εγγυήσεις Επιτυχίας | Ν/Α – «Κύριο Σενάριο Επιτυχίας» |
| Κύριο Σενάριο Επιτυχίας | Οι νέες αλλαγές έχουν αποθηκευτεί και η συνταγή βρίσκεται σε έγκυρη κατάσταση. |
| Επεκτάσεις | Αν κάτι πάει στραβά κατά την διάρκεια της αποθήκευσης, ο χρήστης θα ειδοποιηθεί ανάλογα από το σύστημα. |
| Ειδικές Απαιτήσεις | Η διασύνδεση χρήστη θα πρέπει να εμφανίζεται σωστά ανεξάρτητα από τον συσκευή του χρήστη (κινητό,ταμπλετ,λαπτοπ) |
| Λίστα παραλλαγών τεχνολογίας και δεδομένων | Η διάρκεια κάθε βήματος θα πρέπει να είναι θετικός ακέραιος. Το όνομα της συνταγής και του κάθε βήματος είναι εποχρεωτικό πεδίο |
| Συχνότητα | Λίγοτερο συχνή από την περίπτωση χρήσης Εκτέλεσης/Εισαγωγής. |

***Π1.2. Δημιουργήστε ένα μοντέλο τομέα ανάπτυξης (Domain Model) για το σύστημα με ιδιότητες που θεωρείτε απαραίτητες, συσχετίσεις και τύπους δεδομένων. (10%)***A diagram of a computer program

Description automatically generated

***Π1.3. Δημιουργείστε ένα διάγραμμα ακολουθίας συστήματος (System Sequence Diagram - SSD) για το σύστημα. (10%)***

A diagram of a person's diagram

Description automatically generated

***Π1.4. Δημιουργείστε διάγραμμα ακολουθίας για την περίπτωση χρήσης «Εκτέλεση συνταγής». (10%)***

A diagram of a recipe execution sequence

Description automatically generated

***Π1.5. Δώστε ένα λεπτομερές διάγραμμα κλάσεων λογισμικού για το σύστημα στο οποίο θα συμπεριλάβετε κλάσεις, συσχετίσεις, τύπους δεδομένων και λειτουργίες. (10%)***

Ξεκινάμε με ένα αυτόνομο κομμάτι του Domain που είναι για την Εκτέλεση μια συνταγής.  
Είναι αυτόνομο γιατί όπως θα πούμε και παρακάτω, η λογική μας δεν έχει dependency στην τάξη Recipe , ούτε καν στην τάξη Step. Το BaseRecipeExecution χρειάζεται μόνο μια λίστα από «steps», όπου «step» είναι μια αφηρημένη έννοια, ένα interface(IRunnableOperation) στην δική μας περίπτωση που πρέπει να έχει μόνο 2 ιδιότητες (Duration,Order)  
A screenshot of a computer

Description automatically generated

Οι κυρίως οντότητες.  
Το Quantity στο StepIngredients είναι string για χάρην απλότητας ώστε να μην χρειάζεται να φτιάξουμε και άλλη οντότητα “Units”, οπότε, το Quantity μπορεί να είναι “200γρ”

A screenshot of a computer

Description automatically generated

Υπάρχουν κι άλλες βοηθητικές τάξεις που δεν έχουν σχέση με «Recipes» , δεν τις δείχνουμε εδώ.

**Π2: Υλοποίηση του συστήματος στη Java ή στην Python (ή C#)**

*Αυτό το παραδοτέο θα πρέπει να είναι το project του IntelliJ Idea ή άλλου αντίστοιχού IDE σε μορφή ZIP. Επίσης θα πρέπει να παραδοθεί και η βάση δεδομένων MySQL που χρησιμοποιήθηκε για το project σε μορφή ZIP καθώς και οδηγίες αναδημιουργίας της βάσης δεδομένων και του ρόλου που απαιτείται για την πρόσβασή της από την εφαρμογή.*

Το website είναι διαθέσιμο στο <https://uthrecipesweb2025team1.azurewebsites.net/>

Ο κώδικας είναι συννημένος στο .zip της εργασίας αλλά και στο <https://github.com/GeorgeVovos/UTH-SE/tree/main/Exercises/04-Recipes>   
**Τεχνολογίες/Frameworks**Language: [C#](https://learn.microsoft.com/en-us/dotnet/csharp/) (.NET 9)  
Back-end/Front-end: [ASP.NET Core Razor Pages](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-9.0&tabs=visual-studio) ( .NET 9)

Data Access: [Entity Framework Core](https://learn.microsoft.com/en-us/ef/core/) (.NET 9) (Object Relational Mapper)

RDBMS: Agnostic. Έχουμε υλοποιήσει db providers για MySql και SqlServer , με 1-2 γραμμές ακόμη μπορούμε τα υποστηρίξουμε [δεκάδες ακόμα](https://learn.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli).

**Σύντομη περιγραφή του solution**

A screenshot of a computer

Description automatically generated  
Θα ξεκινήσουμε από πάνω προς το κάτω στο παραπάνω screenshot(αλφαβητικά)

* Solution Items  
  Μόνο ένα ReadMe file για το πως να κάνετε Build/Run το project τοπικά αν χρειαστεί  
  <https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/ReadMe.md>
* Tests
  + [Integration Tests](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.IntegrationTests/RecipeRepositoryTests.cs)  
    Περιέχει μόνο ένα test που δείχνει πως μπορούμε να γράψουμε tests που τεστάρουν τον «πραγματικό» κώδικα against ένα αληθινό DBMS.  
    Χρησιμοποιεί την βιβλιοθήκη [testcontainers](https://testcontainers.com/) για να «σηκώσει» ένα SQL Server σε ένα container για τις ανάγκες του τεστ.
  + [Unit Tests](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.UnitTests/RecipeExecutionTests.cs)  
    Μερικά unit tests με την χρήση των [xUnit](https://xunit.net/) και [Moq](https://github.com/devlooped/moq)
  + Docker-compose  
    Δείτε το readme για περισσότερα.  
    Με 1 εντολή μπουρούμε να φτιάξουμε ενα MySql container και ένα web server για το website μας
  + [Uth.Recipes.Console](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.Console/Program.cs)  
    Δεν είναι κάτι που ζητούσε η άσκηση, απλώς ένα από project όπου μπρούμε να χρησιμοποιήσουμε όλες τις λειτουργίες χωρίς κάποιο UI.
  + [Uth.Recipes.DataAccess](https://github.com/GeorgeVovos/UTH-SE/tree/main/Exercises/04-Recipes/Uth.Recipes.DataAccess)  
    Υλοποίηση το Data Access Layer.Περιέχει κλάσεις για read/write actions στις βάσεις μας (MySql/SqlServer/Db)
  + [Uth.Recipes.Domain](https://github.com/GeorgeVovos/UTH-SE/tree/main/Exercises/04-Recipes/Uth.Recipes.Domain)  
    Η κυρίως λογική της εφαρμογής μας (Interfaces, Domain classes, etc)
  + [Uth.Recipes.IOC](https://github.com/GeorgeVovos/UTH-SE/tree/main/Exercises/04-Recipes/Uth.Recipes.IOC)  
    Ένα βοηθητικό project (IOC= Inversion of Control) με ένα μόνο αρχείο που κάνει set up το dependency injection container. Το χρειαζόμαστε μιας και θέλουμε να τρέχουμε διαφορετικά applications (Web, Console, Integration tests, etc)
  + [Uth.Recipes.Web](https://github.com/GeorgeVovos/UTH-SE/tree/main/Exercises/04-Recipes/Uth.Recipes.Web)  
    Το Web UI μας. Δεν έχει ιδιαίτερη λογική,μόνο ότι χρειάζεται για να παίξουν τα view μας, αυτός ήταν ένας από τους στόχους μας βέβαια.  
    Ο φάκελος Controllers, περιέχει (limited) APIs που μπορούν να χρησιμοποιηθούν από διάφορους Mobile/Web/3rd party clients.  
    Υπάρχει και Test UI page για τα APIs στο  
    <https://uthrecipesweb2025team1.azurewebsites.net/swagger/index.html>   
    Έχουμε υλοποήση λίγα endpoints και προφανώς δεν έιναι πολύ υψηλής ποιότητας, δεν

ακολουθούμε πολλά REST standards, error handling, pagination και άλλα, αλλά δουλεύει μια χαρά.

Θα βρείτε ένα φάκελο ViewModels.  
Συνειδητά είναι σχεδόν clones των domain κλάσεων, λίγο πιο απλά βέβαια.  
Δεν χρησημοποιήσαμε ούτε composition γιάτι θέλαμε μεγαλύτερη ευελιξία.  
Όπου χρειάζεται, κάνουν map τα domain models σε ViewModel και το αντίθετο.

Θα πρέπει να υλοποιήσετε τα ακόλουθα:

**Π2.1**. Την δυνατότητα καταχώρησης συνταγών και τροποποίηση/διαγραφή τους. (10%)  
<https://uthrecipesweb2025team1.azurewebsites.net/RecipeEdit>   
Έχουμε λίγο validation, μόνο ο τίτλος της συνταγής είναι υποχρεωτικό πεδίο.  
Μπορείτε πάντα να βάλετε steps/ingredients/images αργότερα κάνοντας edit.  
  
<https://uthrecipesweb2025team1.azurewebsites.net/RecipeManagement>   
  
[RecipeRepository.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.DataAccess/Repositories/RecipeRepository.cs)

**Π2.2.** Την υποστήριξη εκτέλεσης μιας συνταγής. (15%)

<https://uthrecipesweb2025team1.azurewebsites.net/RecipeExecution>

[https://github.com/...../RecipeExecution](https://github.com/GeorgeVovos/UTH-SE/tree/main/Exercises/04-Recipes/Uth.Recipes.Domain/RecipeExecution)

**Π2.3.** Την πλήρη γραφική διασύνδεση της εφαρμογής. Στην περίπτωση της Java να χρησιμοποιηθεί το Swing framework, ενώ αν το project υλοποιηθεί με Python να χρησιμοποιηθεί η βιβλιοθήκη tkinter (15%)  
<https://uthrecipesweb2025team1.azurewebsites.net/>

**Π3: Κριτική ανασκόπηση**

***Π3.1.*** *Τις αποφάσεις που πήρατε και αποδείχθηκαν ορθές (π.χ. χρήσιμες ιεραρχίες κληρονομικότητας, διασυνδέσεις, GRASP patterns, Design patterns κ.λπ.) και να εξηγήσετε πως αυτές οι αποφάσεις σας βοήθησαν. (2%)*

Χρησιμοποιήσαμε διάφορα best practices, design/GRASP patterns, OOP features έτσι ώστε να έχουμε maintainable, testable, high quality κώδικα.  
Μερικά από αυτά τα βλέπουμε παρακάτω :

* Dependency Injection  
  Σε όπως το solution οι τάξεις δεν έχουν dependencies σε άλλες concrete τάξεις αλλά σε interfaces που να δίνουν την δυνατότητα να αλλάξουμε υλοιποιήσεις και να κάνουμε unit testing πολύ πιο εύκολα.
* Polymorphism
  + [RecipeExecution.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.Domain/RecipeExecution/RecipeExecution.cs)  
    Μια ωραία υλοποίηση για το execution και τα progress μιας συνταγής.  
    Βασικά, η λογική είναι abstract και reusable για οποιαδήποτε use case που έχει steps με ή χωρίς διάρκεια και δεν έχει καν κάποιο dependency σε Recipes ή Recipe steps.
    - [DurationBasedRecipeExecution.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.Domain/RecipeExecution/DurationBasedRecipeExecution.cs)
    - [StepBasedRecipeExecution.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.Domain/RecipeExecution/StepBasedRecipeExecution.cs)
  + [BaseDbContextProvider.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.DataAccess/DatabaseProviders/BaseDbContextProvider.cs)  
    Μας βοηθάει να είμαστε database agnostic. Έχουμε μια υλοποίηση για κάθε database system που θέλουμε να χρησημοποιήσουμε
    - [MySqlDbContextProvider](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.DataAccess/DatabaseProviders/MySqlDbContextProvider.cs)
    - [SqlServerDbContextProvider](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.DataAccess/DatabaseProviders/SqlServerDbContextProvider.cs)
* Factory pattern / Creator
  + [RecipesDbContextProvider.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.DataAccess/RecipesDbContextProvider.cs)  
    Υπέυθυνη για να μας δώσει ποιόν database provider θα χρησιμοποιήσουμε με βάση τις ρυθμίσεις από το config file μας.
* High cohesion  
  Έχουμε πολλές κλάσεις που έχουν ένα ξεχωριστό σκοπό και όχι μεγάλες «god» κλάσεις.  
  πχ πολλά ξεχωριστά repositories για κάθε entity. Διαφορετικές τάξεις, για database providers, recipe execution, models και άλλα.
* Low coupling  
  Όπως είπαμε και παραπάνω, δεν υπάρχουν πολλά dependencies μεταξύ κλάσεων εκτός και αν είναι απολύτως απαραίτητο. Πολλές από τις τάξεις μας μπορούν να επαναχρησομοποιηθούν σε άλλα projects που μπορεί να έχουν ή όχι σχέση με συνταγές.

**Π3.2.** Τυχόν δύσκολα σημεία που συναντήσατε και πως τελικά τα αντιμετωπίσατε. (3%)   
Δεν μπορούμε να πούμε ότι είχαμε πολλές δυσκολίες. Ένα πρόβλημα ήταν ότι με βάση την εκφώνηση της άσκησης, η εφαρμογή ήταν πολύ data-entry specific χωρίς πολυ domain complexity με αποτέλεσμα πολλά από τα domain entities να μοιάζουν με απλά Data Transfer Objects (anemic).  
Παρόλα αυτά, όπως έχουμε πει παραπάνω, έχουμε καταφέρει να τα κάνουμε πιο «rich» βάζοντας λογική όπως validation, collection managements, κτλ. Προσθέσαμε και επιπλέον βοηθητικές τάξεις που με έδωσαν την ευκαιρία να χρησημοποιήσουμε design patterns και OOP features.

Υπήρχαν αρκετές σημαντικές αποφάσεις που έπρεπε να πάρουμε που θα επηρέαζαν την αρχιτεκτονική της εφαρμογής όπως το UI framework και το Data Access framework.  
  
Για το UI διαλέξαμε το [Razor Pages](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-9.0&tabs=visual-studio) μιας και δεν το είχαμε χρησιμοποιήσει. Αν και έγινε η δουλειά, δεν ήμαστε πολυ ευχαριστημένοι. Θυμίζει το παλιό, κλασικό ASP.NET (mid 2000s) , προσπαθεί να κρύψει το stateless nature του http και να κάνει εύκολο το development. Σίγουρα το κατεφέρνει σε ένα βαθμό αλλά μάλλον κάνει τα πράγματα χειρότερα σε πραγματικές εφαρμογές.  
Για κάποια πραγματική εφαρμογή θα συνηστούσαμε κάποιο front-end framework όπως Angular ή React , ή το [ASP.NET Core MVC](https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-9.0)  
  
Για Data Access χρησιμοποιήσαμε το [Entity Framework Core](https://learn.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli) το οποίο είναι το πιο δημοφιλές ORM στο .NET. Όπως όλα τα ORM,έχει πολλά πλεονεκτήματα και μειονεκτήματα.  
Σίγουρα κάνει τη ζωή μας πιο εύκολη σε πολλά use cases μιας με πολύ λίγες γραμμές μπορούμε να υλοποιήσουμε πολλές βασικές και πολύπλοκες λειτουργείες, με strongly typed κώδικα, χωρίς να γράψουμε καθόλου sql.  
Τώρα, στα μεινονεκτήματα, δεν μας απασχόλησε το performance σε αυτό το project μιας και είναι μικρό το scope. Το μεγαλύτερο πρόβλημα ήταν ότι κυρίως στο Edit Recipe και σε λιγότερο βαθμό στο Add Recipe, έπρεπε να γράψουμε λίγο πολύπλοκο και «ugly» κώδικα ώστε να βοηθήσουμε το ORM να χειριστεί σωστά κάποια object graphs , δείτε τα σχόλια στο [RecipeRepository.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.DataAccess/Repositories/RecipeRepository.cs)  
  
Τέλος, αν και όχι δύσκολο, έπρεπε να πάρουμε κάποιες αποφάσεις για το UI ώστε ο χρήστης να έχει μια καλή εμπειρία. Θέλαμε κάτι απλό ώστα ο χρήστης να μπορεί από 1 μόνο σελίδα να μπορεί να εισάγει όλα τα δεδομένα (συνταγές, εικόνες, βήματα, συστατικά , κατηγορίες , κτλ).

Για τις κατηγορίες, έχουμε μια pre-defined λίστα και δεν αφήνουμε τον χρήστη να εισάγει νέες.  
Οπότε,το UI control,είναι ένα από select.  
Για τα συστατικά, θέλαμε ο χρήστης φυσικά να μπορεί να εισάγει νέα, οπότε το αφήσαμε σαν απλό textbox.Το μεινέκτημα είναι ότι μπορεί να εισάγει το ίδια με μικρές συντακτικές διαφορές.  
Αν και θα μπορούσαμε να το έχουμε σαν editable select ώστε να μπορεί τουλάχιστον να διαλέξει existing records, πάλι δεν θα λύναμε το πρόβλημα των συνατικών λαθών, οπότε το αφήσαμε απλό textbox.  
Αφού τα δεδομένα σταλούν στο backend, τα συστατικά είναι «κανονικό» Entity στο backend και στη βάση σαν πίνακας με όλες τις σχετικές συσχετίσεις και στα 2 επίπεδα.

**Π3.3.** Επίσης θα πρέπει να συζητήσετε τον τρόπο με τον οποίο αναθέσατε αρμοδιότητες με την χρήση των προτύπων GRASP (5%)  
Αναφέραμε στο Π3.1 πως χρησιμοποιήσαμε διάφορε design/GRASP patterns. Ο στόχος μάς ήταν να έχουμε απλό, maintainable κώδικα , χωρίς duplicates , testable , “[S.O.L.I.D](https://en.wikipedia.org/wiki/SOLID)” και με βάση αυτά τα requirements, χρησημοποιήσαμε τα κατάλληλα patterns (Indirection, Factory, Dependency Injection, Polymorphism, etc).  
Δύο παραδείγματα που έχουμε αναφέρει και παραπάνω.  
- Η τάξη [RecipesDbContextProvider.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.DataAccess/RecipesDbContextProvider.cs) είναι η μόνη που ξέρει σε ποιο RDBMS σύστημα θα συνδεθεί η εφαρμογή μας, οπότε αυτή είναι υπεύθυνη για να δημιουργήσει και να επιστέψει τον κατάλληλο database provider.

-H (abstract) τάξη [RecipeExecution.cs](https://github.com/GeorgeVovos/UTH-SE/blob/main/Exercises/04-Recipes/Uth.Recipes.Domain/RecipeExecution/RecipeExecution.cs), και η υλοποιήσεις της, είναι υπέθυνες για την λογική της εκτέλεσης μιας συνταγής. Το UI μας απλώς χρησιμοποιή τις λειτουργίες της για να δείξει τα δεδομένα στον χρήστη.  
Αυτό κάνει τις τάξεις εύκολα testable με unit tests ,που έχουμε ήδη γράψει και μας δίνει την σιγουρία ότι ο κώδικας μας δουλεύει σωστά και ότι τυχόν λάθει από αλλαγές στο μέλλον θα γίνουν detect γρήγορα.