Universitatea Tehnică a Moldovei

Proiectarea unei aplicații software utilizînd arhitectura hexagonală

Student: _____ **Corolețchi Octavian**
TI-191M

Coordonator teză: _____ **Ciubotaru Constantin,**
Conferențiar univ., dr.

Chişinău, 2020

REZUMAT

Teza de master a fost creată cu scopul de a cerceta și implementa stilul arhitectural hexagonal pentru o aplicație. Această arhitectură poate fi folosită de un grup mare de dezvoltatori în aplicații de proporții mari. Astfel se poate obține o aplicație modulara, cu componente izolate, ușoare de modificat fără a provoca impact mare asupra celorlalte componente.

În prezent dezvoltarea aplicațiilor a luat amploare și sunt necesare de cîte mai multe sisteme pentru a acoperi o zonă cît mai mare. Unele aplicații mari, ajung la un moment dat cînd orice schimbare ia o cantitate mare de timp iar adăugarea de servicii noi, provoacă impact imens asupra întregului ecosistem.

Structura tezei de master cuprinde introducerea, patru capitole, concluzii generale, bibliografie din 7 surse. Structura tezei de master cuprinde cinci capitole:

- analiza domeniului de studiu;
- modelarea și proiectarea sistemului informațional;
- realizarea sistemului;
- documentarea produsului realizat.

În primul capitol se descriu și se cercetează aspecte generale din domeniul dezvoltării aplicațiilor software, avantaje, dezavantaje, etape de dezvoltare utilizînd arhitectura hexagonală, principii de bază.

În al doilea capitol se foloseș limbajul de modelare UML pentru a descrie structura aplicației, elemente prezentate. Se pot observa o serie de scenarii prezente în aplicație, relațiile dintre componentele aplicației. UML este un limbaj ce oferă o reprezentare grafică standartizată înțeleasă de toți. Se poate modela un întreg sistem precum și componentele necesare pentru funcționarea ulterioară a aplicației propuse.

În capitolul trei sunt descrise instrumentele folosite pe parcursul implementării arhitecturii hexagonale. Inițial dezvoltarea aplicației a fost efectuată folosind un stil arhitectural bazat pe nivele, iar apoi a urmat refactorizarea codului pentru a implem,enta arhitectura hexagonală.

În capitolul patru se demonstrează eficacitatea aplicației, prezentarea use-case-urilor. Aceasta a fost efectuată utilizînd instrumentul Postman, pentru simularea request-urilor către API, și executarea operațiunilor necesare.

În realizarea aplicației și cercetării arhitecturii hexagonale sunt respectate standartele prevăzute și beneficiile acestui tip de arhitectură. Astfel această lucrare are drept scop informarea ulterioară a celor din jur, și preluarea arhitecturii hexagonale drept opțiune de dezvoltare a unei aplicații. Într-un final se va obține un produs modular, puternic în timp, iar orice schimbare care poate veni pe parcursul dezvoltării aplicației, va impacta cît mai puțin drumul dezvoltării aplicației.

ABSTRACT

The current master thesis was created in order to research and implement the hexagonal architectural style for an application. This architecture can be used by a large group of developers in large applications. Thus, a modular application can be obtained, with isolated components, easy to modify without causing great impact on the other components.

Currently, application development has grown and more systems are needed to cover as large an area as possible. Some large applications reach a point where any change takes a large amount of time and the addition of new services has a huge impact on the entire ecosystem.

The structure of the master's thesis includes the introduction, four chapters, general conclusions, bibliography from 7 sources. The structure of the master's thesis comprises five chapters:

- analysis of the field of study;
- modeling and designing the information system;
- realization of the system;
- documentation of the realized product.

The first chapter describes and researches general aspects in the field of software application development, advantages, disadvantages, development stages using hexagonal architecture, basic principles.

In the second chapter the UML modeling language is used to describe the structure of the application, elements presented. You can see a series of scenarios present in the application, the relationships between the components of the application. UML is a language that provides a standardized graphical representation understood by all. An entire system can be modeled as well as the components needed for the subsequent operation of the proposed application.

Chapter three describes the tools used during the implementation of the hexagonal architecture. Initially, the development of the application was performed using an architectural style based on levels, and then followed the refactoring of the code to implement the hexagonal architecture.

Chapter four demonstrates the effectiveness of the application, the presentation of use-cases. This was done using the Postman tool, to simulate requests to the API, and perform the necessary operations.

In the realization of the application and research of the hexagonal architecture are observed the provided standards and the benefits of this type of architecture. Thus, this paper aims to further inform those around, and take over the hexagonal architecture as an option for developing an application. In the end, a modular product will be obtained, strong in time, and any change that may come during the development of the application, will impact as little as possible the path of application development.

CUPRINS

INTRODUCERE

Această lucrare are drept scop de a construi o aplicație software web folosind tipul de arhitectură hexagonală. Se vor analiza avantajele și dezavantajele unui astfel tip de arhitectură. După o analiză a arhitecturii hexagonale se vor studia alternative disponibile ș ice tip se poate implementa pentru un anumit caz specific a aplicației software. Drept limbaj de programare se va folosi Java cu un framework puternic în dezvoltare unei aplicații web denumit Spring.

În industria Tehnologiilor Informaționale termenul de arhitectură poate definit printr-o serie de cuvinte abstracte precum fundament al aplicației, standarte și scopuri, direcție tehnica de dezvoltare ulterioară. Toate aceste abstractizări duc într-un final la o singură idee comună, structură a sistemului și viziunea acestuia.

Noțiunea de arhitectură software poate fi definită într-un final ca arhitectura produsului software. Luînd în considerare acest termen, întîlnim mai multe aspecte de luat în calcul pe lîngă partea software. Aici apare un nou termen de arhitectura aplicației.

Arhitectura aplicației reprezintă o totalitate de alegeri cu care va fi construit produsul. Acestea sunt tehnologiile ce vor fi folosite pe parcursul dezvoltării, framework-uri, librării, limbaje de programare, interfețe de programare a aplicației, în continuare folosind termenul de API. [1]

În continuare apare o noua noțiune precum arhitectură de sistem, fiind un termen mult mai complex și abstract. Îl putem considera drept mai multe aplicații combinate care formează sistemul. Întreg sistem rulează pe un suport hardware. Deci putem defini termenul de arhitectură de sistem totalitatea aplicațiilor software, sisteme și hardware.

Un alt termen utilizat în combinație cu arhitectura unei aplicații software este designul aplicației. Ambele termene au unele lucruri în comun, dar nu pot fi definite ca fiind unul și același lucru. În această combinație de termeni, lucrul în comun este procesul de a lua decizii spre alegerea unei arhitecturi potrivite sau un design care să poată fi atribuit unui specific sistem.

Diferențele principale între acești doi termeni, sunt resursele alocate pentru a lua o decizie anumită fiind ea de tip arhitectural sau de design. În acest scop o decizie arhitecturală i se poate atribui aspecte mai importante decît o decizie de design. O decizie arhitecturală poate consuma mai mult timp și programatori alocați pentru a putea fi implementată decît una de design, aceasta fiind întregul fundament pe care e construit aplicația noastră.

Drept exemple pot fi identificate portarea unei aplicații de la un limbaj la altul, această schimbare fiind atribuită unei decizii arhitecturale. Un alt exemplu ar fi o tehnologie specifică folosită pentru logare a unor evenimente, aceasta atribuindu-se către o decizie de tip design. O schimbare arhitecturală ar putea lua saptămîni întregi iar de design cîteva zile.

La momentul proiectării unei aplicații software se dorește să se construiască o arhitectură software care pretinde a atinge scopuri precum dezvoltarea unui soft adaptiv și flexibil cu costuri de dezvoltare minime. Odata cu dezvoltarea produsului, termenele limită și scurtăturile care intervin pe parcursul formării aplicației fac ca aceste scopuri să fie foarte greu de păstrat și crea o asemenea arhitectură.

Una dintre cele mai simple tehnici de arhitectură este cea bazată pe nivele (vezi figura 1).

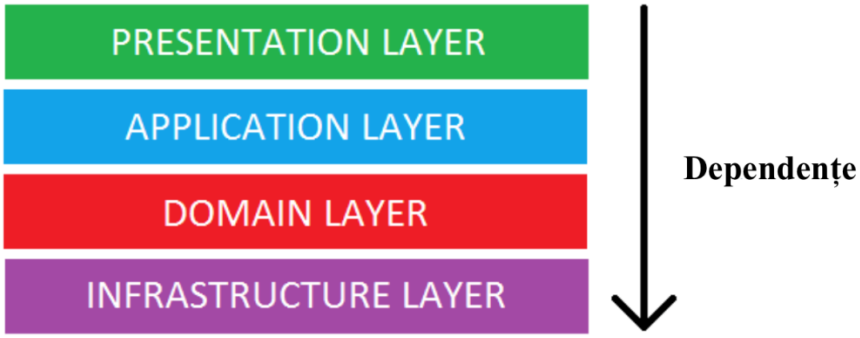

Figura 1 – Arhitectura pe nivele (reprezentare)

Aici dependențele sunt reprezentate de într-o singură direcție, pornind de la nivelul de prezentare, nivelul de aplicație, nivelul de domeniu și ajungînd în final la nivelul de infrastructură. În capitolele următoare se vor analiza de ce acest tip de arhitectură nu este potrivit și cum putem îmbunătăți acest tip folosind arhitectura hexagonală.

Arhitectura hexagonală a fost documentată inițial în anul 2005 de către Alistair Cockburn, și a prins amploare începînd cu anul 2015. Această arhitectură a venit cu intenția să permită aplicațiilor posibilitatea de a fi în mod egal condusă de către utilizatori, programe, teste automate, scripturi și astfel să poată fi dezvoltată și testată în izolare înafara disozitivelor și bazei de date care rulează într-un mod de timp real.

Această arhitectură conține la baza cîteva principii fundamentale ce vor fi enumarate în capitolul 1.4 din această lucrare. Principlaul beneficiu al arhitecturii hexagonale vine contrar opusului ar stilului arhitectural bazat pe niveluri. Aici dependențele nu sunt liniare, provenind de la niveluri superioare spre cele inferioare, ci sunt direcționate către funcționalul de bază al aplicației, către obiectele de domeniu.

Într-un final se dorește ca aplicația software să păstreze opțiuni deschise și adaptate spre schimbări de condiții și factori externi. Astfel se exclude faptul ca aplicația software să preia o greutate enormă și să fie foarte greu de ajustat anumite aspecte în viitor. Astfel se pot exclude costuri enorme de dezvoltare a unui produs software.

Bibliografie

1. Arhitectura Soft, Martin Fowler Blog. ©2019 [citat 01.09.2019] Disponibil: https://martinfowler.com/architecture/
2. Principii S.O.L.I.D ©2020 [citat 01.01.2020] Disponibil: https://muhaz.org/capitolul-concepte-i-abloane-care-au-influenat-dezvoltarea-apl.html?page=3
3. Robert Martin. Agile Software Development: Principles, Patterns, and Practices. Editura Prentice Hall. 2002, 529 p. ISBN 0135974445, 9780135974445
4. Proiectarea Sistemelor Software Complexe ©2020 [citat 01.02.2020] Disponibil: http://www.aut.upt.ro/staff/diercan/data/PSSC/curs-03.pdf
5. UML ©2011 [citat 03.01.2011] Disponibil: https://www.scribd.com/document/46196747/Uml
6. Spring ©2020 [citat 03.10.2020] Disponibil: https://spring.io/
7. Netflix, Schimbări cu ajutorul Arhitecturii Hexagonale ©2020 [citat 10.13.2020] https://netflixtechblog.com/ready-for-changes-with-hexagonal-architecture-b315ec967749