

A Model-Driven Framework to Generate Data and Presentation
Layers' Scaffolding Code for multiplatform Applications



Author

Mohammad Inayatullah

Registration Number

118350

Supervisor

Dr. Farooque Azam

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
ISLAMABAD
AUGUST 2019

A Model-Driven Framework to Generate Data and Presentation
Layers' Scaffolding Code for Multiplatform Applications

Author

Mohammad Inayatullah

Registration Number

118350

A thesis submitted in partial fulfillment of the requirements for the degree of
MS SOFTWARE Engineering

Thesis Supervisor:

Dr. Farooque Azam

Thesis Supervisor's Signature: _____

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD
AUGUST, 2019

Declaration

I certify that this research work titled “*A Model-Driven Framework to Generate Data and Presentation Layers’ Scaffolding Code for multiplatform Applications*” is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Mohammad Inayatullah

FALL 2015-MS-15(CSE) 00000118350

LANGUAGE CORRECTNESS CERTIFICATE

This thesis is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University for MS thesis work.

Signature of Student

Mohammad Inayatullah

FALL 2015-MS-15(CSE) 0000018350

Signature of Supervisor

Copyright Statement

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of NUST College of E&ME. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST College of E&ME, and may not be made available for use by third parties without the written permission of the College of E&ME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST College of E&ME, Rawalpindi.

Acknowledgements

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which you setup in my mind to improve it. Indeed I could have done nothing without your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was your will, so indeed none be worthy of praise but you.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to express special thanks to my supervisor **Dr. Farooque Azam ,Dr. Wasi Haider** for their help throughout my thesis and also for Software Development and Architecture (SDA) and Model Driven Software Engineering (MDSE) courses which they has taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would like to pay special thanks to **Mr. Muhammad Waseem Anwar** for his tremendous support and cooperation. Each time I got stuck in something, he came up with the solution. Without his help I wouldn't have been able to complete my thesis. I appreciate his patience and guidance throughout the whole thesis.

I would also like to thank **Dr. Arslan Shaukat, Dr. Usman Akram** for being on my thesis guidance and evaluation committee.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my family members whose extraordinary support and cooperation always remained a source of motivation for me in accomplishing this tremendous achievement.

Abstract

The use of web and mobile applications is growing very rapidly in the modern era. Due to high end demand of such applications, the software stakeholders want the applications to be available on both mobile and web. In software industry this requires more efforts to develop applications for both mobile and web. Consequently, more resources with different technology experts are needed for the development of multiplatform applications. The software engineers always look for time saving and robust methodology for good, quick and qualitative software development. Web and mobile applications usually composed of three layers i.e. application, business and data. Application layer deals with the UI related concepts that run on browser. On the other hand, business layer deals with the business logic that is usually implemented on server side. Finally, data layer deals the data access from database.

We have performed the literature review in which we found that a methodology is needed where the software engineers can generate scaffolding code for the data and presentation layers considering the modern development technologies of hybrid (ionic) and web apps (angular). Normally in software industry, the system analysts design the class diagram and is handled over to the software developers. The developers start writing code in client side technology, server side technology and also generate database according to the class diagram.

We have proposed the model-based methodology for the development of applications for both mobile and web applications, because, Model Driven Architecture (MDA) is renowned software design approach in software industry that make the software development very rapid and consistent. MDA facilitate the development of multiplatform applications from one UML diagram. Specifically, by applying the principle of “Run everywhere after develop once”, we have designed a profile which have data types and stereotypes of model, class and property meta-class. We have generated the code from class diagram by Acceleo. Our methodology is validated by two case studies demonstrating that the idea is workable. Moreover, one empirical case study was given to 12 industry professional, for evaluating the saving of development effort using the proposed methodology. We found that the proposed approach reduced the amount of effort significantly.

Key Words: *Hybrid App, Web App, Model-Based Scaffolding, CRUD, MDA, Web service*

Contents' Table

Contents

Declaration	i
LANGUAGE CORRECTNESS CERTIFICATE	ii
Copyright Statement	iii
Acknowledgements	iv
Abstract	vi
Table of Contents	vii
Figures' List	x
Tables' List	xii
List of Annexures	xiii
CHAPTER 1: INTRODUCTION	14
1.1 Background Study	15
1.1.1 Single Page Application.....	15
1.1.2 ASP.NET WebAPI	16
1.1.3 Entity Framework Core	17
1.1.4 Model Driven Software Engineering	19
1.2 Problem Statement	20
1.3 Proposed Methodology	20
1.4 Research Contribution.....	21
1.5 Thesis Organization	21
CHAPTER 2: LITERATURE REVIEW	24
2.1 Literature Review.....	24
2.1.1 Research Questions.....	24
2.1.2 Inclusion/Exclusion Criteria	25
2.1.3 Search Process	26
2.1.4 Quality Assessment.....	29
2.1.5 Data extraction & Synthesis.....	29
2.1.6 Actual Data Extraction – Intermediate Results.....	30
2.2 Research Gaps.....	33
CHAPTER 3: PROPOSED METHODOLOGY	36
3.1 Description	36
2.3 Data types.....	38
2.4 Enumerations	39
2.4.1 HttpVerbsEnum	39
2.4.2 StringDVDDataTypeEnum.....	40
2.4.3 NavigationEnum	40

2.4.4	StringValidationEnum	40
2.4.5	FontIconEnum	41
2.4.6	CollectionPlacementEnum.....	42
2.5	Stereotypes	42
2.5.1	PropertyAttributes.....	43
2.5.2	StringTypeAttributes	43
2.5.3	IntegerTypeAttributes.....	44
2.5.4	DateTypeAttributes.....	44
2.5.5	IDAttributes	44
2.5.6	OperationAttributes	44
2.5.7	ModelAttributes	45
2.5.1	ClassAttributes.....	46
CHAPTER 4: IMPLEMENTATION		49
4.1	Web API Transformation	49
4.1.1	Controllers	49
4.1.1	Models	50
4.2	Web App Transformation	50
4.2	Hybrid App Transformation.....	52
4.3	Transformation Engine Architecture	52
4.3	Acceleo Code.....	54
CHAPTER 5: VALIDATION		60
5.1	Auditing System – A Case Study	60
5.1.1	Requirements of Auditing System.....	60
5.1.2	Modeling.....	60
5.1.3	Code Generation	61
5.1.4	Verification	71
5.1.2	Conventional VS Proposed methodology	75
5.2	LEXPERT (Lab Expert)	75
5.2.1	Requirements	75
	Create, Read, Update and Delete Patient	76
	Create, Read, Update Lab.....	76
	Create, Read, Update Lab.....	76
	Create, Read, Update Test	76
	Create, Read, Update Doctor	76
	Create, Read, Update Panel	76
	Create, Read, Update Employee	76
5.2.2	Modeling.....	77
5.2.3	Code Generation	78
CHAPTER 6: COMPARATIVE ANALYSIS.....		81

6.1 Comparison.....	81
CHAPTER 7: DISCUSSION AND LIMITATION	84
7.1 Discussion.....	84
7.2 Limitations.....	85
CHAPTER 8: CONCLUSION AND FUTURE WORK	87
REFERENCES	88

Figures' List

FIGURE - 1.1: TRADITIONAL WEB APPLICATION	15
FIGURE - 1.2: SINGLE-PAGE APPLICATION	16
FIGURE - 1.3: WEB SERVICE USAGE STRUCTURE.....	16
FIGURE - 1.4: LAYER OF ENTITY FRAMEWORK	18
FIGURE - 1.5: DECISION CHART FOR ENTITY FRAMEWORK	18
FIGURE - 1.6: RESEARCH WORK FLOW	21
FIGURE - 1.7: THESIS OUTLINE	22
FIGURE - 2.1: SEARCH PROCESS DETAIL	28
FIGURE - 3.1: PROPOSED METHODOLOGY	37
FIGURE - 3.2: WORKFLOW OF PROPOSED METHODOLOGY	38
FIGURE - 3.3: DATA TYPES FOR CLASS DIAGRAM.....	39
FIGURE - 3.4: HTTP VERBS ENUMERATIONS	39
FIGURE - 3.5: STRING DB DATA TYPE ENUMERATION.....	40
FIGURE - 3.5: NAVIGATION ENUMERATION.....	40
FIGURE - 3.6: VALIDATION ENUMERATION	41
FIGURE - 3.7: FONT ICON ENUMERATION.....	41
FIGURE - 3.8: COLLECTION PLACEMENT ENUMERATION.....	42
FIGURE - 3.8: PROPERTY ATTRIBUTE STEREOTYPE	42
FIGURE - 3.9: OPERATION ATTRIBUTE STEREOTYPE	45
FIGURE - 3.10: MODEL ATTRIBUTE STEREOTYPE.....	45
FIGURE - 3.11: CLASS ATTRIBUTE STEREOTYPE	46
FIGURE - 4.1: ARCHITECTURE OF ANGULAR	51
FIGURE - 4.2: TRANSFORMATION ENGINE ARCHITECTURE	53
FIGURE - 4.2: TRANSFORMATION ENGINE INPUT MODEL INTERFACE	54
FIGURE - 4.3: CODE STRUCTURE OF CODE GENERATOR	55
FIGURE - 4.5: MAIN MODULE OF ACCELEO PROJECT	55
FIGURE - 4.6: ANGULAR MODULE OF ACCELEO PROJECT	56
FIGURE - 4.7: IONIC MODULE OF ACCELEO PROJECT	57
FIGURE - 4.8: ASP.NET WEB API MODULE OF ACCELEO PROJECT	58
FIGURE - 5.1: CLASS DIAGRAM OF AUDITING SYSTEM WITH APPLIED STEREOTYPES.....	61
FIGURE - 5.2: FILES STRUCTURE OF ANGULAR'S GENERATED CODE	61
FIGURE - 5.3: FILES STRUCTURE OF IONIC'S GENERATED CODE	62
FIGURE - 5.4: FILES STRUCTURE OF WEB API'S GENERATED CODE	62
FIGURE - 5.5: GENERATED CODE FOR MODEL OF EF CORE	63
FIGURE - 5.6: GENERATED CODE FOR CONTEXT OF EF	63
FIGURE - 5.7: GENERATED CODE FOR ASP.NET WEB API CONTROLLER	64
FIGURE - 5.8: GENERATED CODE FOR APP.MODULE.TS	64
FIGURE - 5.9: GENERATED CODE FOR ANGULAR'S APP.COMPONENT.TS	65
FIGURE - 5.10: GENERATED CODE FOR ANGULAR'S APP.COMPONENT.HTML	65
FIGURE - 5.11: GENERATED CODE FOR ANGULAR'S VIEW COMPONENTS – EMPLOYEE.....	66
FIGURE - 5.12: GENERATED CODE FOR CODE FOR ANGULAR'S COMPONENTS – EMPLOYEE.....	66
FIGURE - 5.13: GENERATED CODE FOR ANGULAR'S VIEW OF COMPONENTS-LIST – EMPLOYEE....	67
FIGURE - 5.14: GENERATED CODE FOR CODE OF COMPONENTS-LIST – EMPLOYEE	67
FIGURE - 5.15: GENERATED CODE FOR ANGULAR'S CODE COMPONENTS-LIST – EMPLOYEE	68
FIGURE - 5.16: GENERATED CODE FOR ANGULAR'S MODEL CLASSES – EMPLOYEE.....	68

FIGURE - 5.17: GENERATED CODE FOR ANGULAR’S SERVICES – EMPLOYEE	69
FIGURE - 5.18: GENERATED CODE FOR IONIC’S VIEW – EMPLOYEE.....	69
FIGURE - 5.19: GENERATED CODE FOR IONIC’S MODULE – EMPLOYEE	70
FIGURE - 5.20: GENERATED CODE FOR IONIC’S TYPESCRIPT OF PAGE – EMPLOYEE	70
FIGURE - 5.21: CODE DEPLOYED FOR AUDITING SYSTEM’S WEB API (C#).....	71
FIGURE - 5.22: CODE DEPLOYED FOR AUDITING SYSTEM’S ANGULAR CODE.....	71
FIGURE - 5.23: CODE DEPLOYED FOR AUDITING SYSTEM’S AIONIC CODE	72
FIGURE - 5.24: COMPILING ANGULAR APP	72
FIGURE - 5.25: COMPILING IONIC APP	73
FIGURE - 5.26: COMPONENT VIEW OF ANGULAR’S WEB APP	73
FIGURE - 5.27: COMPONENT LIST VIEW OF ANGULAR’S WEB APP	74
FIGURE - 5.28: PAGE VIEW OF IONIC’S HYBRID APP.....	74
FIGURE - 5.29: DISPLAY OF ANGULAR’S APP	76
FIGURE - 5.30: CLASS DIAGRAM OF LEXPERT	77
FIGURE - 5.31: STRUCTURE OF GENERATED CODE FOR WEB APP (ANGULAR)	78
FIGURE - 5.32: STRUCTURE OF GENERATED CODE FOR HYBRID APP (IONIC)	79
FIGURE - 5.33: STRUCTURE OF GENERATED CODE FOR RESTFUL SERVICE (ASP.NET WEB API)	79

Tables' List

TABLE 2.1 - SEARCH PROCESS DETAIL	26
TABLE 2.2 - EXTRACTION OF DATA & SYNTHESIS	30
TABLE 2.3 - INTERMEDIATE RESULTS	30
TABLE 4.1 - HTTPVERBS TRANSFORMATION RULE (CRUD)	49
TABLE 4.2 - TRANSFORMATION RULES FOR RESTFUL API.....	50
TABLE 4.3 - TRANSFORMATION RULES FOR ANGULAR CODE (WEB APP).....	51
TABLE 4.4 - TRANSFORMATION RULES FOR RESTFUL API.....	52
TABLE 5.1 - COMPARISON OF CONVENTIONAL TIME AND MODEL DRIVEN TIME.....	82

Annexures' List

ANNEXURE - A.....	90
ANNEXURE - B.....	91
ANNEXURE - C.....	92
ANNEXURE - D.....	93

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

This first chapter delivers a comprehensive introduction of the performed research work which is categorized in different sections. Section 1.1 presents background study. In Section 1.2 problem statement is described. Section 1.3 is proposed methodology. Section 1.4 is research contribution and Section 1.5 represents thesis organization.

1.1 Background Study

The background study introduces the concepts being used in this research which are;

- 1) Single Page Applications (SPA)
- 2) ASP.NET WebAPI
- 3) Entity Framework Core.

The details of the following are given in subsequent sections.

1.1.1 Single Page Application

Single Page Application is also called SPA, SPA is a web application type when user can interact with web application then user should feel to work like on desktop applications. Only single page is loaded in the browser and contents are dynamically rewriting rather than retrieving and loading of pages from server. Single Page Application (SPA) avoid the interruption caused by rendering the pages on the server.

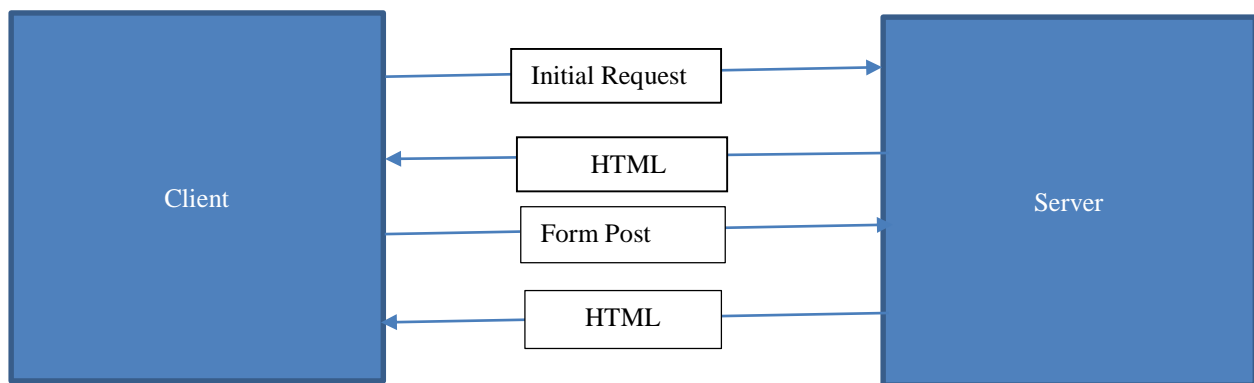


Figure - 1.1: Traditional Web Application

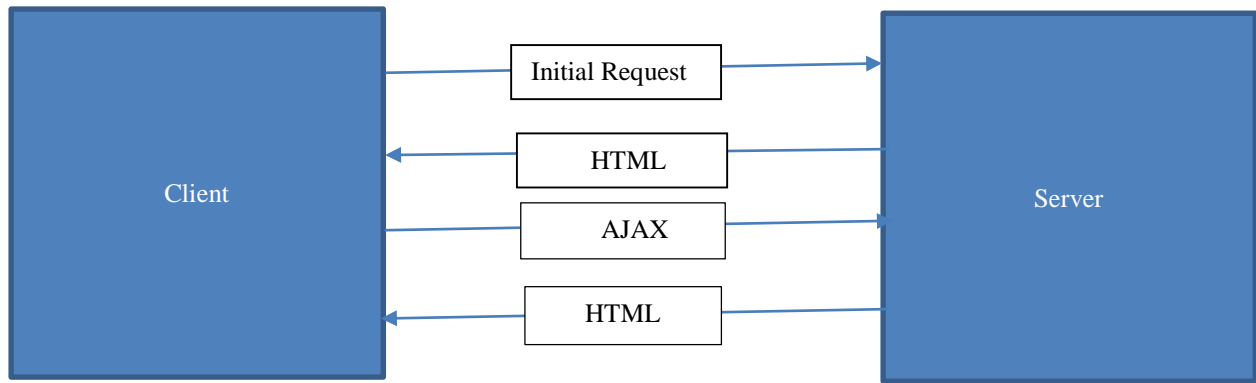


Figure - 1.2: Single-Page Application

1.1.2 ASP.NET WebAPI

ASP.NET API is a framework of .NET for (APIs) web services development. Developers may create web API using this framework. The client like web browser, mobile devices or desktop applications can consume the web service. ASP.NET WebAPI have very good support to RESTful web services. RESTful API is an API (application program interface) which uses HTTP requests to HTTP verbs like GET, POST, PUT and DELETE. This is useful to define an abstraction on the top of the software frameworks which allow stakeholders to participate in the design and development of apps. The structure of the web service is shown in **Figure 1.3**.

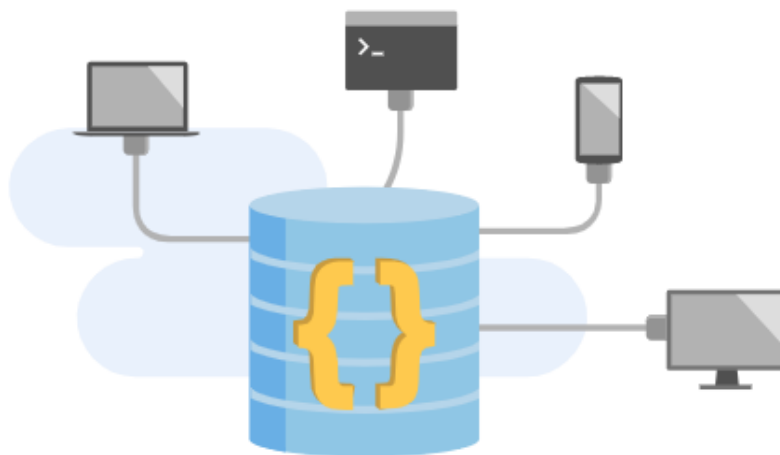


Figure - 1.3: Web Service Usage Structure

1.1.3 Entity Framework Core

Entity Framework is an ORM (Object Relational Model) released by Microsoft with .NET 3.5 in 2008 [1]. Object Relational Mapper (ORM) is used to implement manipulation of database using object-oriented programming language. ORM increase the productivity of software development. Entity Framework is a means of interaction between underlying relational database and application developed in .NET technologies. EF makes it easy for software engineers to map database objects with .NET object model (C#, VB, etc.). Before .NET 3.5 the developers were using ADO.NET for save or retrieve data from underlying database. In .NET framework the developer can easily execute the queries using LINQ. Classes in the model represents the tables in database. The underlying database can be generated from model classes of object oriented programming language and also model classes of object oriented programming language can be generated from database. Entity Framework is used in the data layer of application as shown in **Figure 1.4**. Current version of Entity Framework is EF6. Three approaches of development are existing in Entity Framework.

1. Model First Approach
2. Code First Approach
3. Database First Approach

In Model First Approach the entities along with concerned relationship and inheritance are created on design using EDMX. This is a type of model driven approach to design the class diagram on entity designer. The underlying database is generated from EDMX file and for each class the table is created in database [2].

In Code First Approach the model classes along with concerned relationship and inheritance are developed in the code like C# or VB. The underlying database is generated from the code. The tables are created in database according to classes in code [3].

In Database First Approach the relational database is created first and the entity model is generated according to the tables in the database.

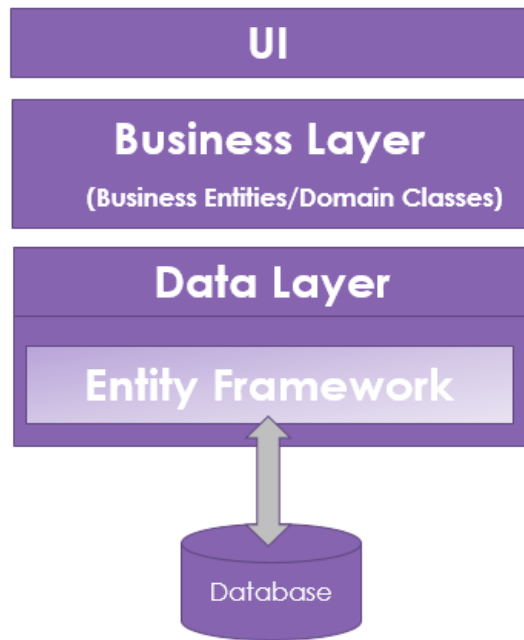


Figure - 1.4: Layer of Entity Framework

When the developer is choosing that what approach of development in Entity framework is needed then the following flow chart can help the developer to choose the entity framework approach.

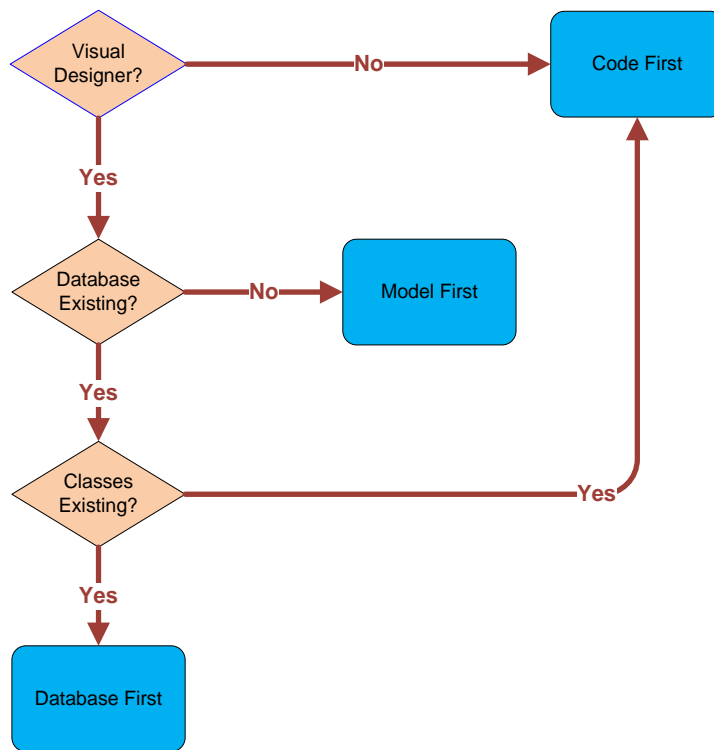


Figure - 1.5: Decision chart for Entity Framework

Microsoft has also released the .NET Core 1.0 in 2014 [4]. .NET core is cross platform, open source platform for developing applications. Currently latest version of .NET Core is 3.0. The Entity Framework Core (EF Core) was released in June 2016 as an ORM for .NET Core. Same as EF, the EF Core has LINQ which is a database query language. The productivity of development increases due to LINQ. .NET core has two development approaches, one is Code First and second is Database First. Model First approach is not existing in EF Core. In software development industry the Code First approach has more importance as compared with others. In .NET core the Model First Approach is not existing so for Model Driven Developer there is double effort one is to design the system in UML class diagram and second is to code for UML class diagram in C#. Normally the developers want to apply the rule of less typing, more code.

1.1.4 Model Driven Software Engineering

Model-driven software engineering (MDSE) is a software design approach to abstract the complexity of development of software systems. It makes the design and development of software applications simple. This is commonly useful in different domains like web applications, embedded systems etc. and is used in complex application development. Model Driven approach uses models as a combination of guidelines which can be used in structuring and organizing design specifications. This approach creates abstraction layer to separate application and business logic from platform specific technology. The functionality of system can first be defined as a platform independent model (PIM). The model which is platform independent, is a type model for a system which have no any implementation information particular to technology. A platform specific model is a model of a system that has implementation information particular to technology. Transformation techniques convert platform independent models that identify the operations of systems to produce platform specific models that identify the details of how those systems use the capabilities of their platforms to provide their operations.

The MDA model is related to multiple standards. Among them UML (Unified Modelling Language) is one of the most powerful language which has been used by many software engineers. UML provide a standard way to make visualize the design of a software system. UML diagrams are of two categories, Behavioral UML diagrams and Structural UML diagrams . The diagrams which represent the structure of the system are structural whereas the diagrams which describes the functionality of the software system are behavioral. UML conceptual models can be

customized and extended using a UML profile diagram. A profile is a lightweight extension mechanism to the UML standard.

1.2 Problem Statement

Emerging of technologies is very fast, the developers when receive the UML diagrams from system analyst, the developer get starting the writing code from scratch. In case of class diagram the developer have to implement the code for

- 1) Database tables
- 2) Server side model classes
- 3) Client side model classes in typescript or JavaScript

This is a difficult task to write the code on three sides. Writing the same class model in the different place there may be more chance of error and also time consuming. Due to high end demand of mobile apps the application owner want the application should be on both mobile and web app. This is the reason the developing the code for both, hybrid and web app, is a tedious task. There is a need of model driven methodology to generate code for mobile app, web app and service. There are some more reason to develop a generator from class diagram to classes of client side and server side technologies.

Developers are currently lacking modeling tools to generate code of EF Core model, web api controller and at the same time UI code like angular, react, vue.

1.3 Proposed Methodology

The methodology for the research is systematic which is depicted for the below diagram given in figure 1.6. The diagram show the steps of systemic research that is performed. First of all the Systemic Literature review is performed and then the research gap found from literature review. In research gap the problem is identified. A comprehensive literature is performed for the problem. The solution for the problem is proposed. The proposed solution of the problem is model driven scaffolding code generation of CRUD operation for web based applications. UML profile and stereotypes for different purposes are defined. In the research a tool is developed for transformation of code from class diagram to underlying technologies. The transformation is verified by two case studies one is auditing system and other is LEXPERT (Pathology Lab Management System)

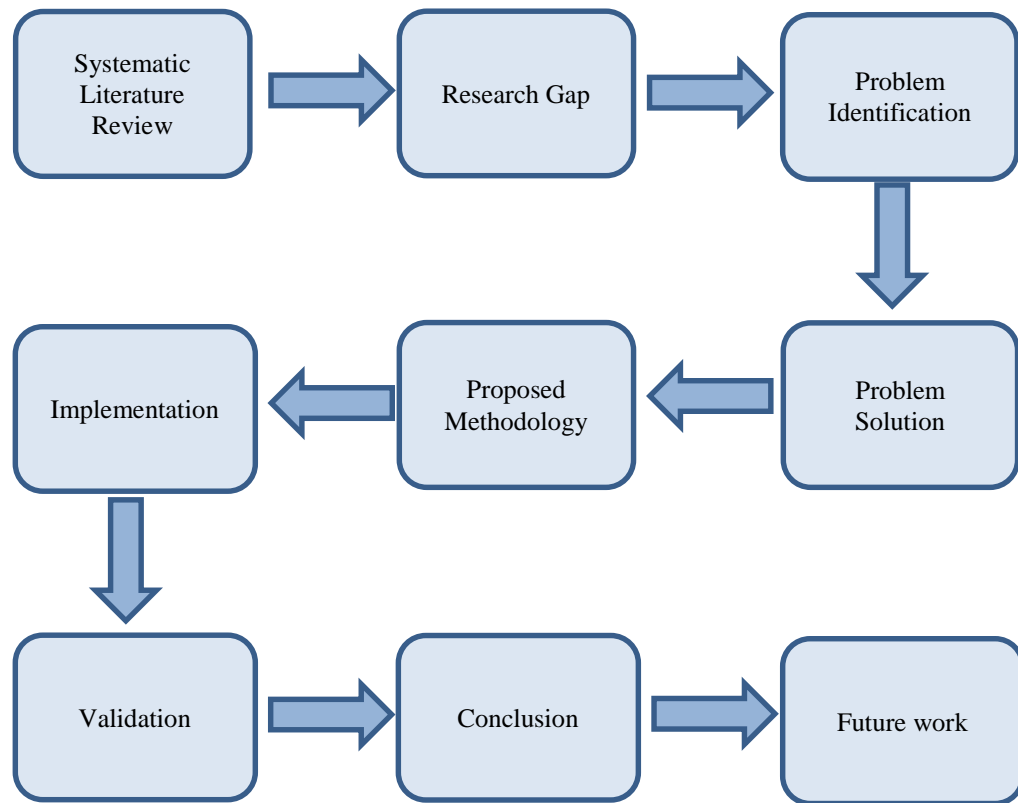


Figure - 1.6: Research Work Flow

1.4 Research Contribution

The contribution made in this research work are as follows:

- Defining stereotypes for class diagram of web applications both for server side and Client side (UI).
- Transformation engine to generate client side (UI) and backend code implementation by transforming class diagram model to underlying code in Angular, Ionic , ASP.NET Web API, EF Core. The transformation engine is developed using Acceleo.
- Validation of the proposed work by deploying it to Auditing System and LEXPERT (Pathology Lab Management System)

1.5 Thesis Organization

Figure 1.7 represent the thesis organization. **Chapter 1:** Deals with introduction consisting of background study about the concepts used in research, problem statement, research contribution

and thesis organization. **Chapter 2:** Comprises of systemic literature review and research gap that is found in the systematic literature review. **Chapter 3:** Consist of description of the proposed methodology for the identified problem. This is a model driven base methodology. **Chapter 4:** Describes the implementation of the proposed methodology in detail. The transformation engine is discussed along with the architecture of the code of transformation engine. **Chapter 5:** The validation of proposed methodology is discussed in this chapter. The validation is done by two case studies. One is Auditing system and other is LEXPERT (Lab Expert, Pathology Lab Management System). The proposed methodology is validated by these two case studies correctly. **Chapter 6:** composed of analysis of proposed work with the previous research work and the conventional software development. **Chapter 7:** A brief discussion on the work done is presented in this chapter. It also contains the limitations in our research. **Chapter 8:** Concludes the research and future work for the research is recommended.

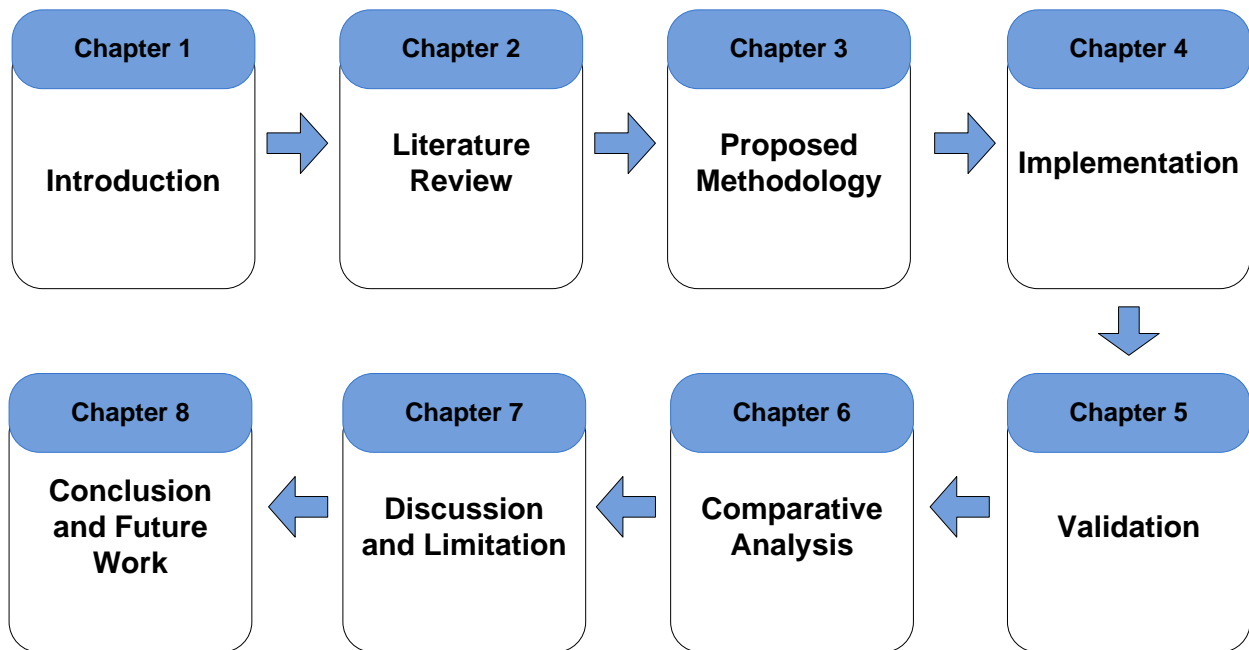


Figure - 1.7: Thesis Outline

Chapter 2

Literature Review

CHAPTER 2: LITERATURE REVIEW

Research work conducted from 2010 to 2019 in code generation for web application is described in this chapter. System Literature review is performed in literature review. Out of 27156 papers 15 are selected for literature review. From these papers we find the research gap which also mentioned in this chapter.

2.1 Literature Review

Context: Model driven code transformation for web is a specialized research area under Model-Driven Engineering (MDE). So much research is done in this area resulting in an enormous amount of publications. **Objective:** A SLR is necessary to give an extensive examination of the work done in Model code transformation in web applications. **Method:** A detailed systematic literature review is conducted on Model driven code transformation with web as a focal point. We defined different criteria according to our research questions to identify and shortlist a number of key studies related to our research area. After examining different studies on the basis of our inclusion/exclusion criteria, we shortlisted 30 publications. **Results:** Our systematic literature review show the general status of the essential features of the model driven code transformation for web and finalized the tool which are needed to develop for the code transformation in web. For example, related to code transformation in web artifact, it was observed that creating domain specific languages plays a central role in a lot of model driven web technologies. **Conclusion:** Our findings propose that tool is needed to be developed which can generate domain specific code from class diagram for latest web and hybrid technologies like angular, ionic, react reactjs and vue etc.

2.1.1 Research Questions

1. From 2010 to 2019, what most significant researches have been reported where UML to code transformation has been utilized.
2. What tools are available for model tool transformation from UML to code
3. Technologies that are used for transformation of UML to code
4. For what domain of software the transformation tools are developed
5. What is the need of developer as transformation tool from UML to Code in web technologies.
6. Which technology needs more work in MDSE based transformation from UML To Code

2.1.2 Inclusion/Exclusion Criteria

A concrete standard needs to be defined to approve and disapprove of a particular research. The following six criteria are defined to select or reject a research work:

1. Relevance of Subject:

Select only that research work that is pertinent to our research area. We should be able to find the answers to the research questions that are defined above. Unconcerned research work will be rejected.

2. 2010-2019:

Chosen research work must be published from 2010 to 2019. The research work which is published before 2010 will be discarded so that to assure the consideration of most current research.

3. Publisher:

Selected Research work will be from following eminent scientific databases

- IEEE
- SPRINGER
- ELSEVIER
- ACM

4. Crucial-effects:

The performed research which is to be chosen must have constructive outcomes with respect to Model-Driven transformation of UML to Code. Those research work will be discarded which have no critical results on Model-Driven transformation and UML to Code generation.

5. Results-oriented:

The research work which is to be chosen must be result oriented. That research work which is verified through weak validation method will be discarded.

6. Repetition:

That research work which is identical in area of Model-Driven transformation, UML to Code generation then only one of them will be selected and the remaining will be discarded.

2.1.3 Search Process

Inclusion/Exclusion criteria which are defined in the section given above, limited our performed search on some filters. According to conditions we have chosen four databases for researching the papers to perform Systematic Literature Review. The databases that are chosen contains high standard conference proceedings and journal. We also get some help from the books, which are in stock of these databases. The databases which we choose for searching materials are IEEE, SPRINGER, ELSEVIER and ACM. We apply the filter on the year of publication of the paper i.e. from 2010 to 2019. In our search process we use operators (AND/OR) that we may get precise accurate results The result of this process is depicted in Table 2.1.

Table 2.1 - Search Process Detail

<u>Sr.#</u>	<u>Search Term</u>	<u>Operator</u>	<u>Search Result</u>			
			<u>IEEE</u>	<u>Springer</u>	<u>Elsevier</u>	<u>ACM</u>
<u>1</u>	UML Code Transformation	AND	94	2720	4	52
<u>2</u>	Model Driven Code Generation	AND	641	6298	19	470
<u>3</u>	Acceleo Transformation	AND	8	126	0	1
<u>4</u>	Model Driven Angularjs	AND	1	24	0	0
<u>5</u>	Model Driven PHP	AND	22	1483	3	7
<u>6</u>	Model Driven Javascript	-	32	935	5	46
<u>7</u>	Code from UML	AND	463	4361	12	110
<u>8</u>	Model Driven SPA	AND	41	373	0	4
<u>9</u>	Model Driven ASP.NET	AND	1	103	160	0
<u>10</u>	Model Driven web development	AND	465	7662	17	393

We further narrow our research for getting the most desired material. We perform following steps:

- Some performed search terms are cited and then by defining inclusion/exclusion we come up with 27156 results.
- After the title seeing and removing duplicates we left out 2322.
- After reading the abstract we feel that 1075 researches are not satisfying our scope of research. So we discard those 15234 researches.
- We discard 634 papers after going through the overview of papers.
- 201 papers are rejected on the basis of general study.
- Full text was not available for 56 papers, so we discard those papers.
- There were about 34 researches which were eBooks and webpages. We discard 34 researches.

On the detailed study of remaining papers we finalized 150 papers which are related to our preferred study. The remaining are rejected. So at the end we have 150 papers remaining, from which we are performing our SLR on 15 papers. This complete process is shown in **Figure 2.1**.

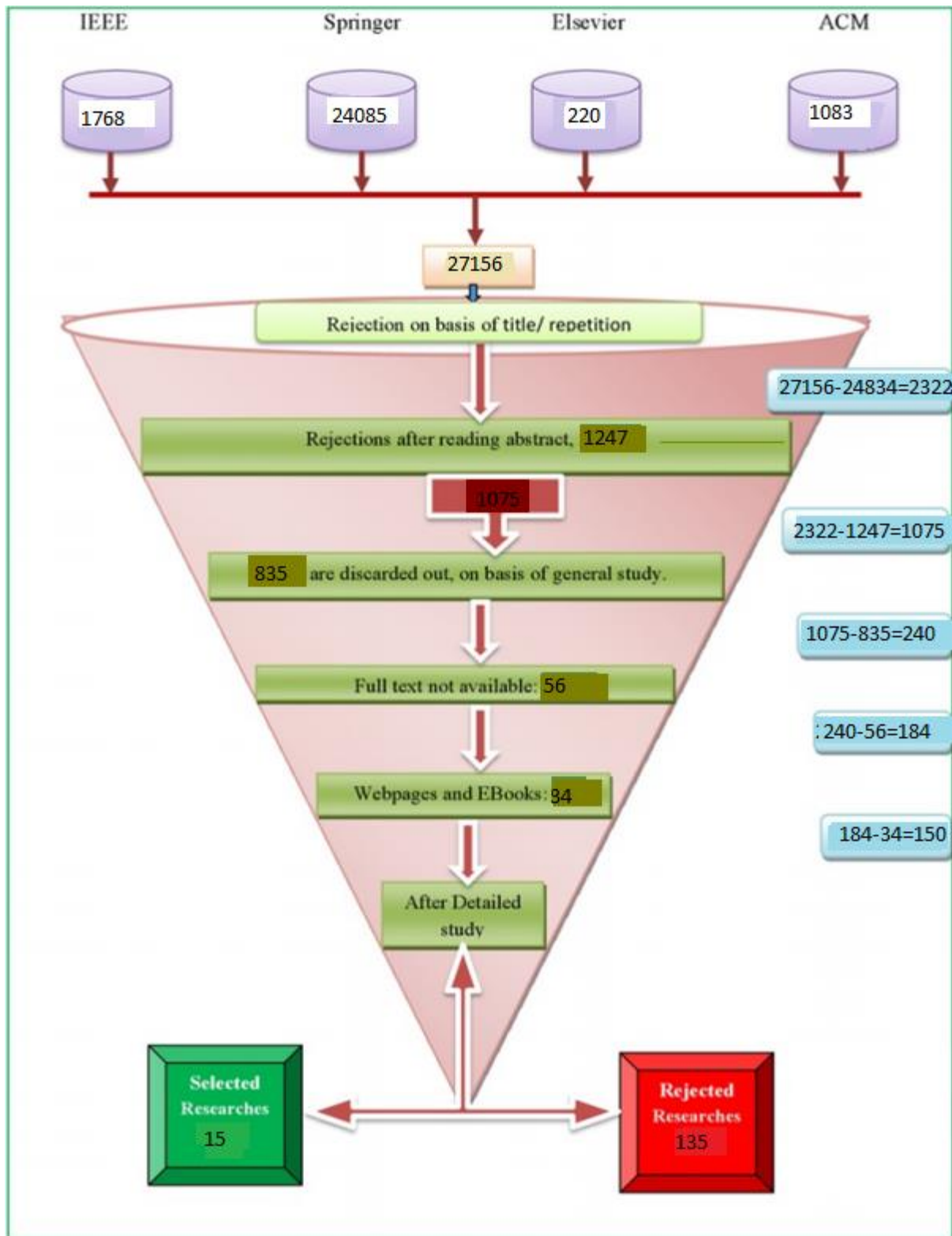


Figure - 2.1: Search Process Detail

2.1.4 Quality Assessment

The quality of studies is important to be assessed likewise inclusion/exclusion criteria:

- A methodical inclusion/exclusion criteria still needs to be provided
- Variations of study results needs to be validated
- The effectiveness of individual studies needs to be weighted when outcomes are incorporated
- To control the understanding of discoveries and check the quality of deductions.
- The understanding of results needs to be controlled and the quality of derivations needs to be checked
- Further research analysis needs to be guided

2.1.5 Data extraction & Synthesis

The objective of this subject is to propose a tenet for systematic reviews acceptable for model driven software engineering research worker. A scientific research may suggest after the evaluation of all the present accessible studies that is applicable to a selected research question or the selected topic or area of interest for development. The objective of the systematic review is to display a good analysis of the topic of study by employing a procedure that is dependable, detailed and auditable. The guideline has been custom-made to replicate the particular issues of software engineering research. It doesn't consider the effect of inquiry sort on the audit techniques, nor will it determine personally systems required to attempt meta-examination. The extraction of data & synthesis is given in **Table 2.2**

Table 2.2 - Extraction of Data & Synthesis

Sr.#	Title	Description
a	Record Information	The information that contains Title of research, author, year of publication, the details of publisher and research type (whether it is journal or conference)
b	Overview	The fundamental proposition and target of the research work that is selected
c	Outcome	Outcome of the research
d	Data gathering	The data collected is qualitative or quantitative
e	Presumption	To prove outcomes, some presumptions are defined
f	Validations	Approval strategy to accept the proposition

2.1.6 Actual Data Extraction – Intermediate Results

We found 135 researches related to our point of interest. Among these researches we perform our SLR on 15 researches. The general overview of these selected researches is given in **Table 2.3**.

Table 2.3 - Intermediate Results

1	Wutthichai Chansuwath [5]	In this paper a UML profile and transformation tool is developed for AngularJS application. This tool is for the use of system architect/analyst. The tool generate the templates for the developer to make the application development fast. If the system analyst is not aware of angular then this will be difficult for the him/her to use the developed profile.
---	------------------------------	---

2	Hanane BENOUDA. [6]	In this paper a tool is discussed which is developed by Author. The tool is generating the code from class diagram using acceleo. Mainly the CRUD operations are implemented. The tool is generating code for cross platform mobile apps. This work can be enhanced for other mobile applications like android or iOS.
3	Sarrah Roubi [7]	This paper is discussing the model driven approach for RIA application using IFML(Interaction Flow Modeling Language). Authors have defined metamodel for RIA and also developed the transformation tool.
4	Sarrah Roubi [8]	The authors have published paper in 2015 for RIA. Here the author has discussed the approach for RIA MVC. They have defined metamodels and developed the generating tool. They have suggested to include more components in the future
5	Salvador Martinez et al [9]	In the paper the suite WebRatio is discussed which is developed by authors for generating code for mobile applications. The tool is generating the code for Apache Cordova framework
6	Kriengkri Pongpanjanthra [10]	A tool is discussed in this paper developed in StarUML generating the UI of web portal from class diagram of navigation of web site. Angular is used as front end technology for generating UI. Functional requirements are not implemented here in this tool. This tool is just generating the UI of site with navigation functionality.
7	Jose Luis Herrero Agustin [11]	In this paper a model driven architecture for web application development is proposed. In this proposed method the focus is on ajax tools. WCF which .NET technology for web service is focused for backend code generation. A profile is developed with web domain.

8	Siti Azreena Mubin. [12]	In this paper a design mode called UEWDM (UML-Extension Web Design Model) is proposed for the development of web application. UEWDM is extension of existing web design model. The focus of proposed designed model is on association and relationship. In three main stages the proposed design model is divided, namely navigational design, conceptual design and user interface design.
9	Tomas Cerny [13]	In this paper three profiles are designed for the web application which target the technologies like JPA, Hibernate. The profiles are for class diagram. The future work is suggested to do this same work for the advanced technologies
10	Nilber Vittorazzi Almeida [14]	In this paper a code generation tool is presented which transform code from FramWeb to CSharp code. The CRUD features are generate for CSharp code in this tool.
11	Aljendro Cortines [15]	In this paper a web based development tool is proposed which can generate code using the agile methodology. And whenever changes are made to the model those changes are also replicate on code also. Future work is suggested to do the same work for modern framework of web technologies
12	Zuriel Morales [16]	In this paper a tool is discussed which is developed by authors. In the tool the web application code is generated from model of web for PHP technology. In the model the archetiche can place the entities of Form, input etc
13	Piero Fraternali [17]	In this paper Model driven approach for the web development is propsoed which insteam of WEBML. The tool is for RIAs which generate code from class diagram to java base mvc. View is generate in LZX

14	Rober Rodriguez-Echeverria [18]	The tool AutoCRUD developed by the authors is presented in this paper. AutoCRUD is for webRatio which generate CRUD operations in IFML (Interacting Flow Modeling Language). For future work two things are suggested (1) enhancing the register functionality to provide pattern-based IFML (2) From pre-existing IFML model register patters
15	Sofia Larissa da Costa [19]	In this paper the authors describe the designed Stereotype for UI to improve the development of Web Portals by using model driven approach to build the UI (User Interaction) for Web applications. The proposed UI Stereotype for Web Portals, can generate different Web portals with the same intention and from a common set of metamodels, fostering reuse in UI Engineering.

2.2 Research Gaps

This section discusses the research gaps from the literature review. After analyzing the above literature, it has been observed that many researches have worked on transformation of code from models but we found that there is a very limited work to transform code from class diagram to its specific technology [9, 10, 11]. The transformation from class diagram that is done is for old technologies and also only for client side or server side technologies. Now a day in market SPA (Single Page Application) is cry of the day in web development applications. Latest technologies like Angular, React, vue etc. are emerged for Single Page Applications.

Wutthichai Chansuwath [1] has worked on transformation to angular but in this paper the author has build profiles for parts of angular application. In this like activity this is a tedious task for the System Analyst to make the diagram. There is a need if the Analyst to develop the methodology where is system analyst is not aware of underlying technologies and the class diagram is build up, after that the developer transform the class diagram to the scaffolding code of CRUD operations in many technologies like angular, react, vue and server side technologies like ASP.NET Web API and Entity Framework.

We also find that in Framework which have three famous approaches. One is Code First approach, second is Database first approach and thirist is Model (EDMX) first approach. Entity framework core is an Object Relational Mapper ORM in .NET core in which this is appreciated to follow the code first approach although database first approach is also existing, but model (EDMX) first approach is not existing in Entity Framework core. In code first approach which is highly recommended, the code writing is a time consuming task. So in there is need of a methodology to be developed for transformation of class diagram to Entity Framework classes.

Chapter 3

Proposed Methodology

CHAPTER 3: PROPOSED METHODOLOGY

As we have discussed that developing web applications in software industry is growing to Single Page Application (SPA). Angular, React and Vue are well known client side technologies for SPA (Single Page Application). System Analyst build up class diagram for the system.

As we discussed that software industry is growing towards Single Page Application development (SPA) in web technologies. Developing such applications need two side development one is client side and other is server side. It means that there will be entities models on both sides, server side and client side. This chapter described our proposed methodology for the resolving the issue discussed in the research gap.

In UML, a profile provides a light-weight generic extension mechanism that UML models may be customized for particular platforms and domains. which are based on various elements: Stereotypes (allow you to increase UML vocabulary) , Tag values(used to extend the UML properties so that we can add additional information in the specification of a model element) and Constraints(to specify conditions that must be held true at all time). These all are applied to specific element of model such as Classes, Operations, Attributes, Activities etc.

3.1 Description

In our proposed methodology the system analyst has to design the class diagram by applying the designed stereotypes and the software developer will generate the client side and server side code will be generated for multiple technologies of Web UI and Mobile applications. The code generate from the class diagram is the scaffolding of CRUD operations for client side and server side. The developer can easily change the code after getting the scaffolding. When the system analyst build up the class diagram for software developers then the developer will transform the Class diagram to code of underlying technologies like Angular, Ionic, React and React Native. The specialty of our proposed methodology is that the code generated is for both web and mobile application. The code transformer is written in ACCELEO which is tool of eclipse. Our proposed methodology is depicted in the **Figure 3.1**.

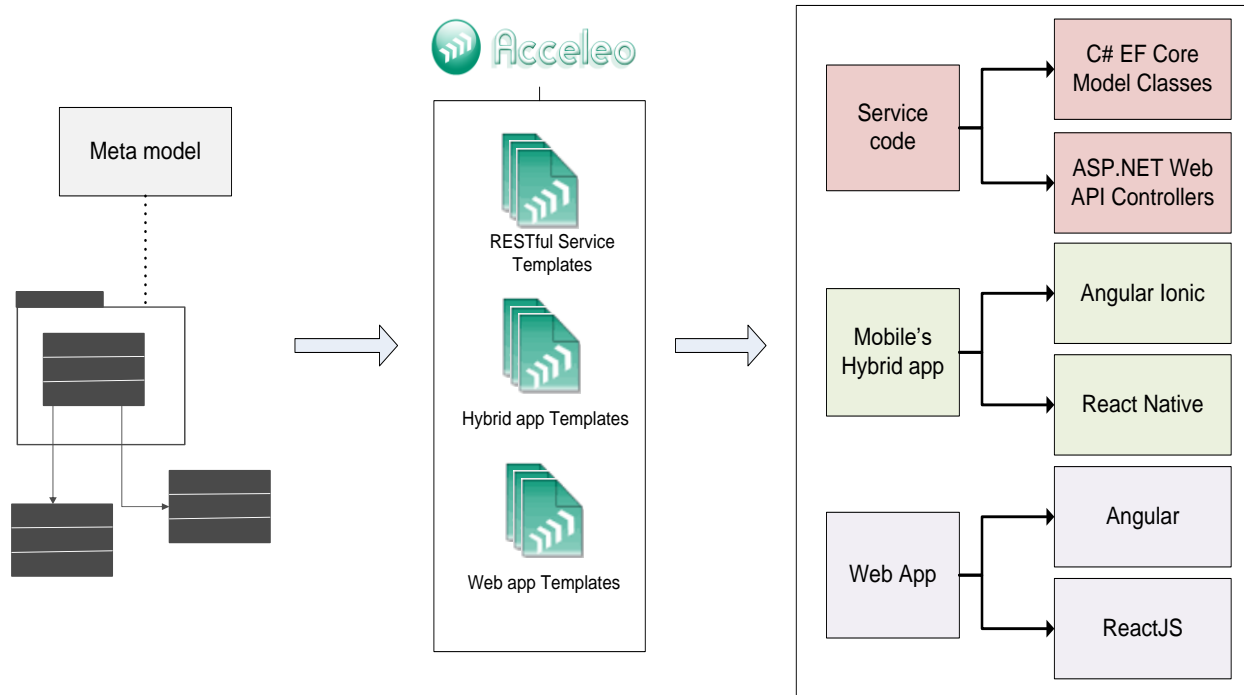


Figure - 3.1: Proposed Methodology

Our proposed methodology is developed in Unified Modeling Language tool Papyrus based on Eclipse. Our proposed profile provides stereotypes to properties of the class diagram from where both server side and client side code is to be generated. These stereotypes are the extensions of UML meta-classes and it provide support to modeling using multiple UML diagrams. In our proposed methodology we have focused to make the profile very simple and easy that new user can use the profile very easily and have no hurdles to apply the stereotypes of profiles to the class diagram. The detail of stereotypes is described in this chapter.

On client side we have focused technologies like Angular, Ionic, React and vuejs and on server side we have focused the C# with ASP.NET WebAPI core and Entity Framework Core. In our proposed methodology we have create some data types according to the data types of C#. These data types are described in this chapter. We have create a structure of stereotypes related to Application Structure are defined. The details of each of the stereotype along with its functionality is discussed in the sub section. Transformation rules are defined for model to text transformation. We have focused to make it so much easy for the system analyst to create the class diagram as per the profile defined. The workflow of proposed methodology is shown in **Figure 3.2**.

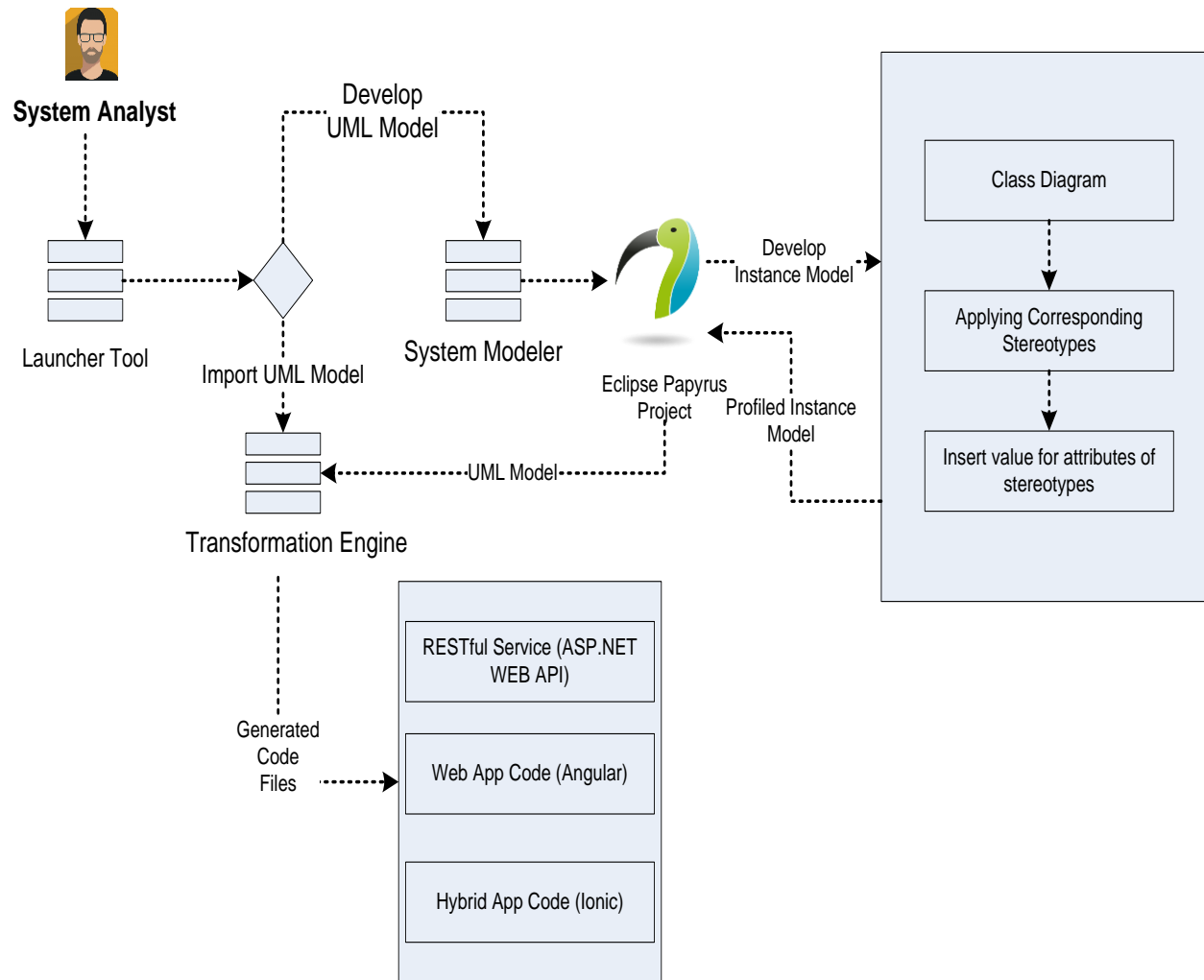


Figure - 3.2: Workflow of Proposed Methodology

2.3 Data types

As discussed that we are targeting multiple technologies to generate code for. The data types of C# are more from other concerned technologies so we focused on the C# to design the data types in the model. We designed the data types like Int32, byte, sbyte, short, ushort, int, uint, long, ulong, float, double, decimal, char, bool, string and DateTime. The C# data types are chosen for modeling because C# has more data types than other so we can easily implement the transformation for other technologies. The data types designed I given in the **Figure 3.3**.



Figure - 3.3: Data Types for Class Diagram

2.4 Enumerations

As we need enumeration in our model for stereotype, so the enumerations designed in our proposed methodology is described one by one on this section.

2.4.1 HttpVerbsEnum

HttpVerbsEnum is for the action method of CRUD operation. This is to be applied to the MetaClass of operation. The Enumeration is given in the **Figure 3.4**.

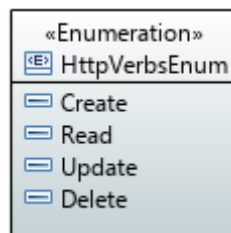


Figure - 3.4: Http Verbs Enumerations

2.4.2 StringDVDataTypeEnum

“StringDBTypeEnum” is the enumeration of list of database string data types. This is to be applied the MetaClass of Property for the string data types. This is to specify for database that what string data types should be in the database. The Enumeration is given in the **Figure3.5**.

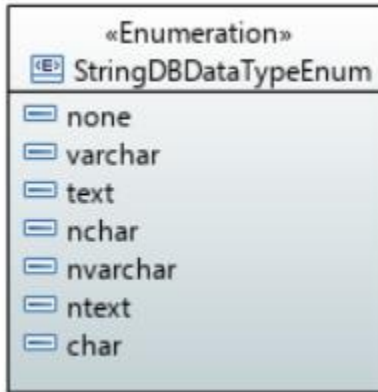


Figure - 3.5: String DB Data Type Enumeration

2.4.3 NavigationEnum

In our UI the navigation panel is to set on left, right or top side. The navigation placement is possible by “NavigationEnum” Enumeration. The “NaviagationEnum” is to be applied to MetaClass of Property. The Enumeration is given in the **Figure 3.5**.

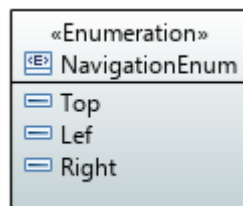


Figure - 3.5: Navigation Enumeration

2.4.4 StringValidationEnum

Normally in the applications the fields have validation like email, URL, onlyalphabets etc. The validation means that the user can enter data with some specific restriction otherwise the data entered will not be acceptable. So for this purpose we have designed the enumeration

“StringValidationEnum”. The “StringValidationEnum” is to be applied to MetaClass of Property. The Enumeration is given in the **Figure 3.6**.

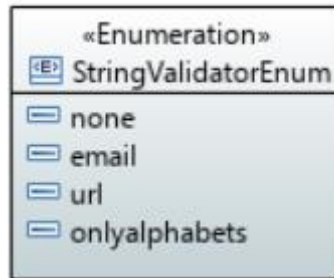


Figure - 3.6: Validation Enumeration

2.4.5 FontIconEnum

In the UI we are applying the font icon to each field for that purpose the “FontIconEnum” is designed. The “FontIconEnum” is to be applied to MetaClass of Property. The Enumeration is given in the **Figure 3.7**.

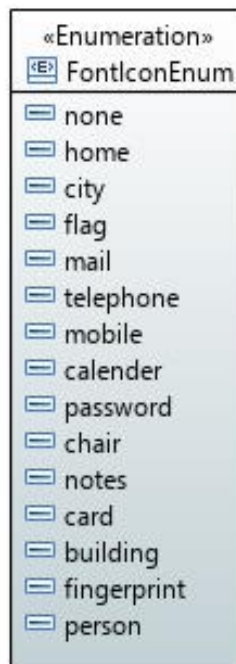


Figure - 3.7: Font Icon Enumeration

2.4.6 CollectionPlacementEnum

In the Web UI the tabular collection of objects is to be placed with editor page or separate page, for this purpose the “CollectionPlacementEnum” enumeration is designed. The “CollectionPlacementEnum” is to be applied to MetaClass of Property. The Enumeration is given in the **Figure 3.8**.

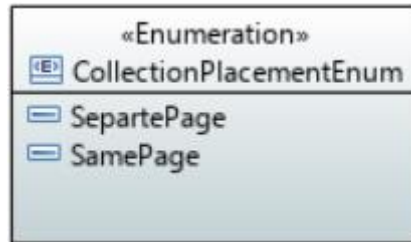


Figure - 3.8: Collection Placement Enumeration

2.5 Stereotypes

The stereotypes of UML model designed in our proposed methodology is discussed one by one in given below sections. The diagram of stereotypes is shown in

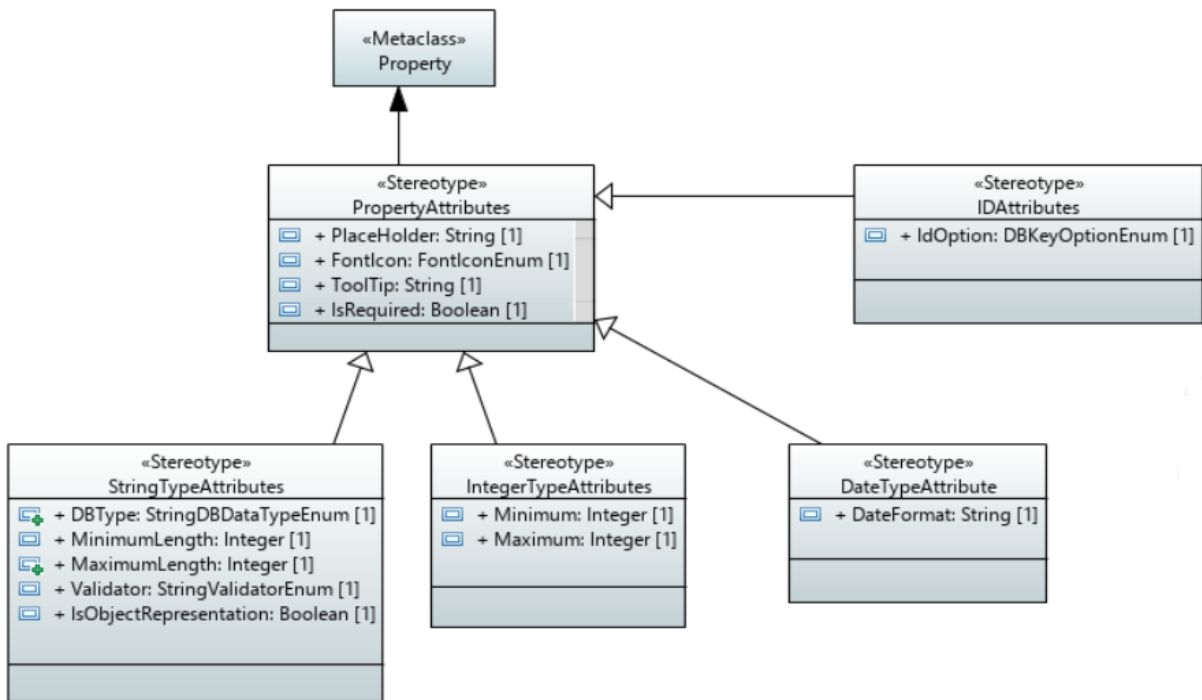


Figure - 3.8: Property Attribute Stereotype

2.5.1 PropertyAttributes

The “*PropertyAttributes*” stereotype defined is of *Meta-Class* property. The “*PropertyAttributes*” stereotype is given in **Figure 3.8**. The properties in “*PropertyAttributes*” are discussed below

- **Placeholder:** In the UI of application, placeholder is used for the helping text inside the control. By this property the system analyst can write the placeholder for a field of class. “*Placeholder*” is of type string.
- **FontIcon:** In the UI of application there is an option to show the font icon with each field of class. “*FontIcon*” is used for this purpose. “*FontIcon*” if of enumeration type.
- **ToolTip:** “*ToolTip*” is for the purpose of showing tip for the user of control. In the UI to show the tip the text is to be set in this property of stereotype.
- **IsRequired:** “*IsRequired*” is for the purpose of required validation. Which this property of stereotype is checked it means that user will enter this value otherwise the form will not be submitted.

“*PropertyAttributes*” has four sub inherited stereotypes, named “*StringTypeAttributes*” and “*IntegerTypeAttribute*”, “*DateTypeAttribute*”, “*IDAttribute*”. Each of these is discussed below in detail

2.5.2 StringTypeAttributes

“*StringTypeAttributes*” stereotype is inherited from “*PropertyAttributes*”. The properties of “*StringTypeAttributes*” is discussed in detail below

- **DBType:** “*DBType*” is of type “*StringDBDataTypeEnum*” enumeration. This for the purpose of specifying the string fields of the class for database. When the C# code to generate for Entity Framework core there is need for string data types to be specified for according to database. In our methodology we are focusing on SQL Server database.
- **MinimumLength:** This is for specifying the minimum length of string type. This property is of integer type.
- **MaximumLength :** This is for specifying the maximum length of string type. This property is of integer type.

- ***IsRepresentative*** : In the UI the collection of items is to be shown in dropdown, list or radio buttons. To show objects in collection one field need to display for each item. For that purpose we have include the property “*IsRpresentative*” in the “*PropertyAttributes*”. This is of type Boolean, means that this can be set true or false.
- ***Validator***: This property of stereotype is for the validation of controls. Validation can be email, URL, onlyalphabets or any other thing. Validator is of type “*StringValidatorEnum*” enumeration.

2.5.3 IntegerTypeAttributes

“*IntegerTypeAttributes*” stereotype is inherited from *PropertyAttributes*. The properties of “*IntegerTypeAttributes*” is discussed in detail below

- ***Minimum***: This is for specifying the minimum of integer type. This property is of integer type.
- ***Maximum***: This is for specifying the maximum of integer type. This property is of integer type.

2.5.4 DateTypeAttributes

“*DateTypeAttributes*” stereotype is inherited from *PropertyAttributes*. The properties of “*DateTypeAttributes*” is discussed in detail below

- ***DateFormat***: For the developer this is required to give the format of date. This is possible from this property of “*DateTypeAttribute*”

2.5.5 IDAttributes

“*IDAttributes*” stereotype is inherited from *PropertyAttributes*. The properties of *IDAttributes* is discussed in detail below

- ***IsID***: “*IsID*” show that the field to which the stereotype is applied is to be ID (key) field of the class.

2.5.6 OperationAttributes

“*OperationAttributes*” stereotype is of type meta-class *Operation*. This is for the operation class diagram. The model of *OperationAttributes* is given in the figure.

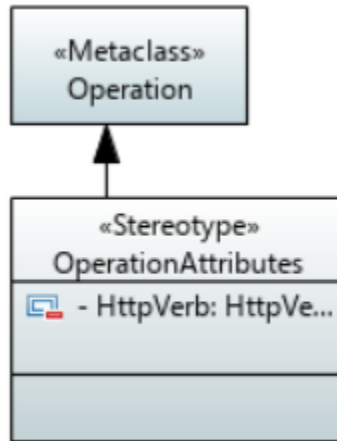


Figure - 3.9: Operation Attribute Stereotype

The detail of “*Operationattribute*” is discussed below.

- **HttpVerb:** “*Httpverb*” is property of type “*HttpVerbsEnum*” enumeration type. For specifying the operation in the CRUD operation this stereotype can be used.

2.5.7 ModelAttributes

“*ModelAttributes*” stereotype is of type meta-class Model. This for the Model of the diagram. The model of “*ModelAttributes*” is given in the figure.

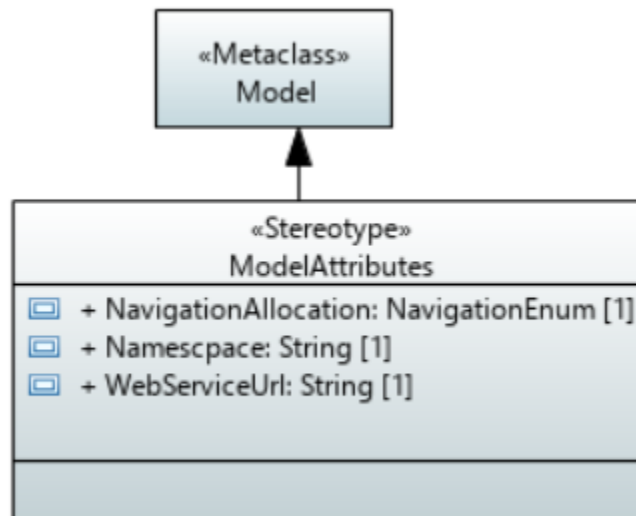


Figure - 3.10: Model Attribute Stereotype

The detail of *ModelAttributes* is given discussed below

- **NavigationAllocation:** This property of stereotype “*NavigationAllocation*” is of type “*NavigationEnum*” enumeration. One can set the Navigation panel placement in UI by left or top.
- **Namespace:** This property of stereotype *Namespace* is of type string type. One can write the namespace of application by using this stereotype.
- **WebServiceUrl:** The code generator is generating code for web API (web service). To consume that web service in UI, the web service URL is to be mentioned in this property of stereotype.

2.5.1 ClassAttributes

“*ClassAttributes*” stereotype is of type meta-class Class. This for the Class of the diagram. The model of “*ClassAttributes*” is given in the figure.

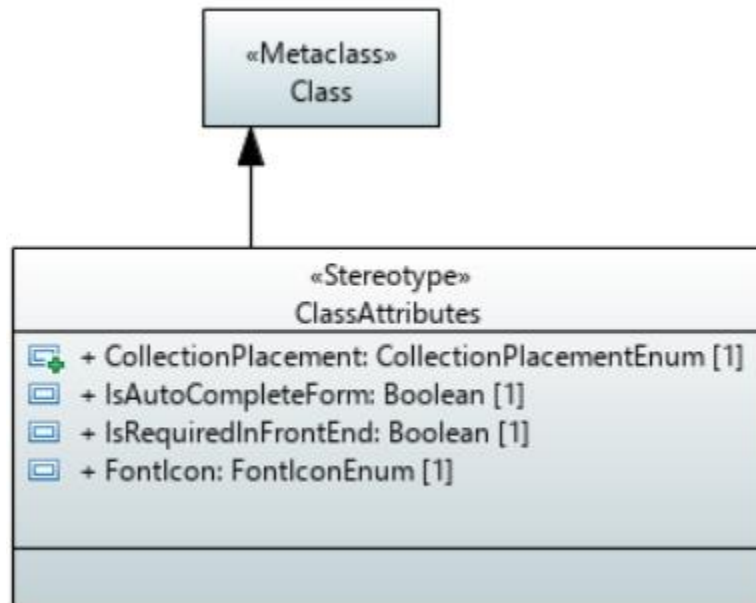


Figure - 3.11: Class Attribute Stereotype

The detail of *ClassAttributes* is given discussed below

- ***IsCollectionOnSamePage:*** This property of stereotype “*IsCollectionOnSamePage*” is of type *CollectionPlacementEnum* enumeration. On UI the tabular collection of the objects is to be placed in the same of page of editor or different page. This is possible by applying this stereotype.
- ***IsAutoCompleteForm:*** This property of stereotype “*IsAutoCompleteForm*” is of type Boolean. On UI when the user enter some data in the fields next time the field can be auto populated by writing the first character of the previous data. This is called autocomplete. In our methodology this is possible by “*IsAutoCompleteForm*” property of stereotype.

Chapter 4

Implementation

CHAPTER 4: IMPLEMENTATION

The implementation of our proposed methodology is discussed in detail in this chapter. Papyrus modeling editor is used for modeling. Aceleo is used for the transformation from model to text. The transformation engine transform the class diagram to Service side code like ASP.NET Web API, Web App code like Angular and Hybrid App Code like Ionic.

4.1 Web API Transformation

For web API the class diagram is transformed to C#. This is done for the controllers of web API and model classes of Entity Framework core.

4.1.1 Controllers

The controllers is the concept of MVC where the action methods are existing to respond in to the action coming from view. The controller concept is also implemented in the ASP.NET Web API. In Web API the action methods are according to the Http Verbs (GET, POST, PUT, and DELETE) which is CRUD operation.

In our implementation we have generate controller for each class in the class diagram inside the folder named *Controllers* on the root of project. The code for action methods of CRUD operation is generated according to the “*OperationAttributes*” which is stereotype of meta-class operation. The action methods of controller are beautified by HTTP verbs annotation. **Table 4.1** show the implementation of HTTP verbs.

Table 4.1 - HttpVerbs transformation rule (CRUD)

No.	HttpVerbsEnum	HTTP Verbs	C# annotation
1.	Create	POST	[HttpPost]
2.	Read	GET	[HttpGet]
3.	Read	GET	[HttpGet(“{id}”)]
4.	Update	PUT	[HttpPut(“{id}”)]
5.	Delete	DELETE	[HttpDelete(“{id}”)]

Two action methods are generating for the GET HTTP Verb. One is without parameters which is for retrieving all the record from database and other is with parameter of id of the class which is

for reserving of one record of specific id. All other action methods are according to the rules of controller of ASP.NET Web API.

4.1.1 Models

Our methodology is generating server side model class code in C# for Entity Framework. As we have discussed that Entity Framework Core and Entity Framework 7 have no features of EDMX (A modeling tool in visual studio). We are transforming the class diagram to C# model classes focusing on Entity Framework Core but this can also work for Entity Framework 7. From these model classes the database tables is to be generated by using command *Add-Migration* and *Update-Database*. For each class in diagram class is to be generated in C#. Implementation rules are described in the following **Table 4.2**.

Table 4.2 - Transformation rules for RESTful API

No .	Category	Language	Generated File Naming	Generated File Path	UML
1	API Controller	C#	Class name + “Controller.cs”	WebAPI/Controller	Class
2	EF Model Class	C#	Class name + “.cs”	WebAPI/Model	
3	Conext	C#	Model name + “Context.cs”	WebAPI/Model	Model

4.2 Web App Transformation

Transformation of UI for Web App is done for angular 8. As discussed the code for angular 8 is in typescript. In our transformation engine the code is generated with best architecture approach of UI angular. On HTML side the reactive form approach code is generating in our code generator. We have adopted the three layer architecture on the UI side code. Services are generated which communicate with the Web API. The data retrieval and data sending is possible by services. Classes on typescript are generated according to the class diagram. The architecture of the Angular is depicted in **Figure 4.1**

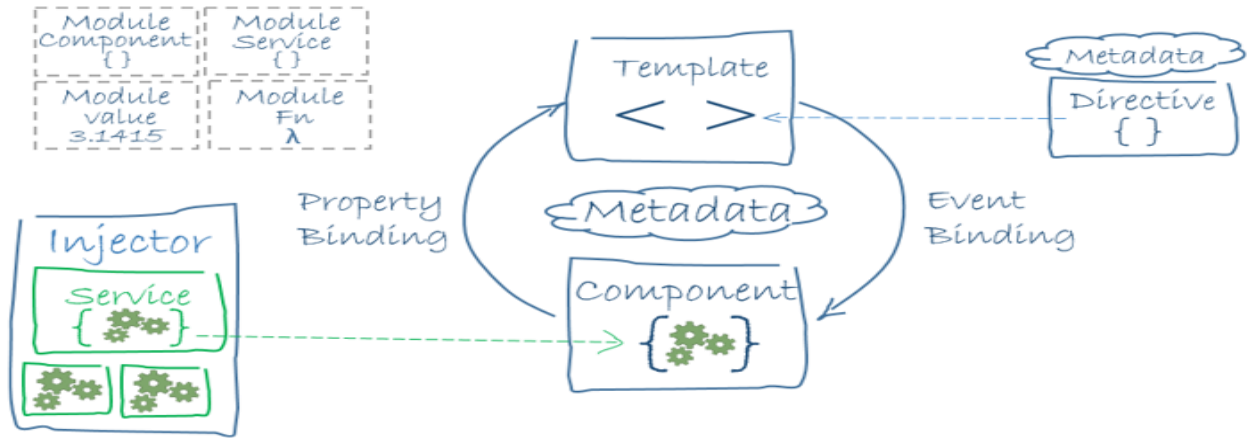


Figure - 4.1: Architecture of Angular

The file types that we are generating by our code generator is listed below

- Model classes
- Services
- View App Component
- Code App Component
- Module
- View Component
- Code Component
- View Component-List
- Code Component-List

The class are generated according to the classes in the model of class diagram. The description of transformation rules for angular’s web app are described in **Table 4.3**.

Table 4.3 - Transformation rules for Angular code (Web App)

No .	Category	Language	Generated File Naming	Generated File Path	UML
1	App Module	Typescript	app.Module.ts	Angular/App/	Model
2	App Routing	Typescript	routing.module.ts		
3	AppComponent	HTML	app.Component.html		
		TypeScript	app.Component.ts		
4	Inerface	TypeScript	Class name + “.model.ts”	Angular/App/shared/	Class
5	Service	TypeScript	Class name + “.service.ts”		
6	Component	HTML	Classname + “.component.html”	“Angular/App/” + class name + “s/” + class name	
7	Component	Typescript	Class name + “.component.ts”		
8	Component List	HTML	Class name + “.component-List.html”		
		Typescript	Class name + “.component-List.ts”		

4.2 Hybrid App Transformation

Transformation of hybrid App is done for Ionic. In our transformation engine the code is generated with best architecture approach of Ionic. We have adopted the three layer architecture on the UI side code. Services are generated which communicate with the Web API. The data retrieval and data sending is possible by services. Classes on typescript are generated according to the class diagram. The transformation rules of hybrid app in ionic is given in **Table 4.4**.

Table 4.4 - Transformation rules for RESTful API

No	Category	Language	Generated File Naming	Generated File Path	UML
1	App Module	TypeScript	“app.module.ts”	Ionic/App/	Model
2	App Routing	TypeScript	“app-routing.module.ts”		
3	App Component	HTML	“app.Component.html”		
		TypeScript	“app.Component.ts”		
4	Interface	TypeScript	Class name + “model.ts”	Ionic/App/ Model/	Class
5	Service	TypeScript	Class name + “service.ts”	Ionic /App/Services/	
6	Page	HTML	Class name + “.page.html”	Ionic/App/ +Class name+”s/”	
		TypeScript	Class name + “.page.ts”		
		TypeScript	Class name + “.module.ts”		
7	List Page	HTML	Class name + “-list.page.html”	Ionic/App/ +Class name+”s/”	
		TypeScript	Class name + “-list.page.ts”		
		TypeScript	Class name + “-list.module.ts”		

4.3 Transformation Engine Architecture

The architecture of the transformation engine is shown in **Figure 4.2**. The tool used for model to text transformation is Aceleo. There transformation engine comprises of two main components i.e. User Interface (UI) and Aceleo transformation.

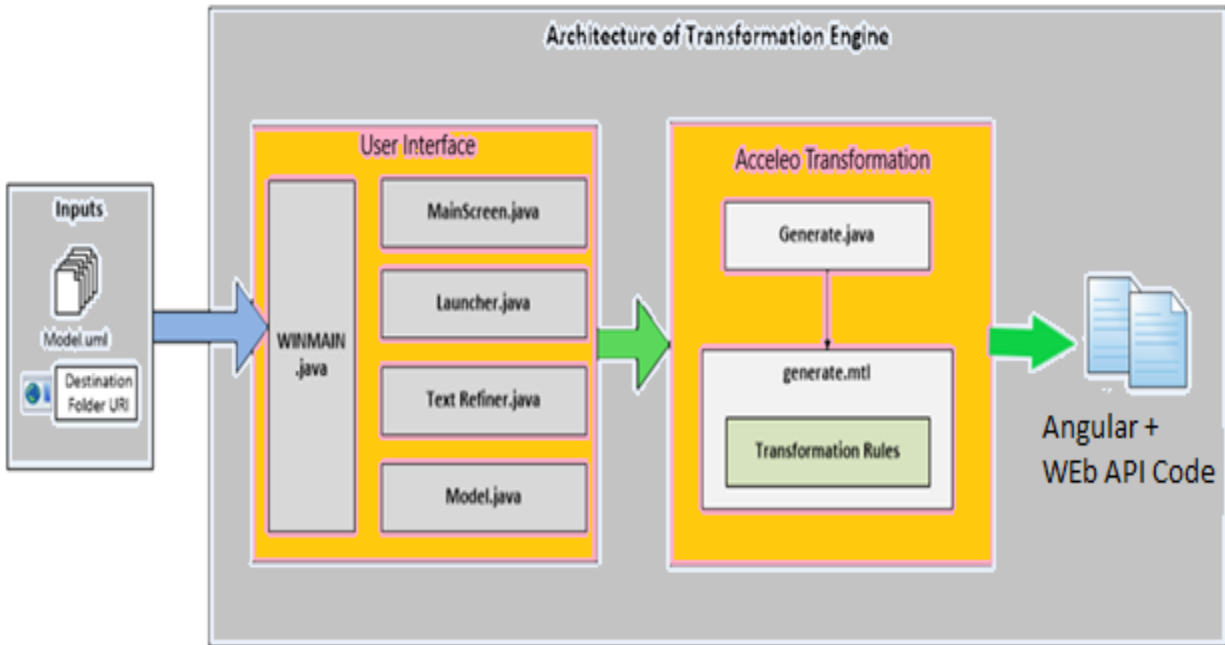


Figure - 4.2: Transformation Engine Architecture

Main Interface: The main interface of the transformation engine consists of three main sub-components which are WinMain, Main Screen, Launcher and Text Refiner. The main executor of the transformation engine is Main Screen which provide list of actions it can perform and provide graphical user interface with help of buttons and text field. The java classes which implement these functionalities are *Launch* and *WinMain*. *Text Refiner* is a java class that convert the strings into proper format for further use. The interface for transformation engine is provide in **Figure 8**. Transformation Engine takes the UML model and path of output folder for generating the code from model using the browse button. Generate button is provided to generate the required outputs. Console shows the progress of the transformation process. A *Reset* is provided to clear all fields i.e. input model path, output folder path and console. *Close* button is provided to closes the interface from the screen.

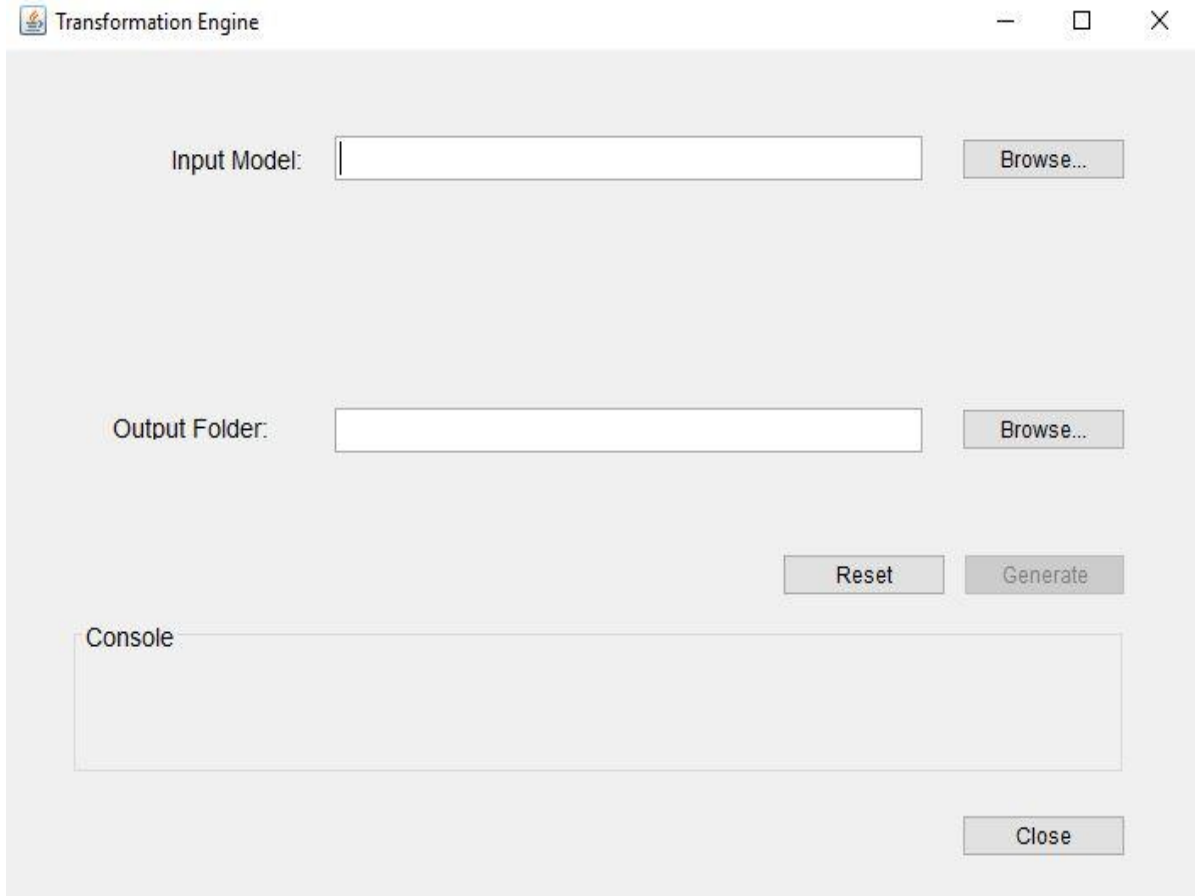


Figure - 4.2: Transformation Engine Input Model Interface

4.3 Acceleo Code

The code structure of acceleo project is organized according to the target code. As the generator is generating the code for angular web app, ionic hybrid app and ASP.NET web API, we have organized the code structure according to the target code. The code structure and all the code of angular web app, ionic hybrid app and ASP.NET Web API are shown in **Figures 4.3**. The main module of the acceleo code is calling all the sub modules like angular, ionic and web api. The code of main module of acceleo project is shown in **Figure 4.5**. The code for generating of angular's web app is shown in **Figure 4.6**. The code for generating ionic app is shown in **Figure 4.7** and the code for RESTful service is show in **Figure 4.8**.

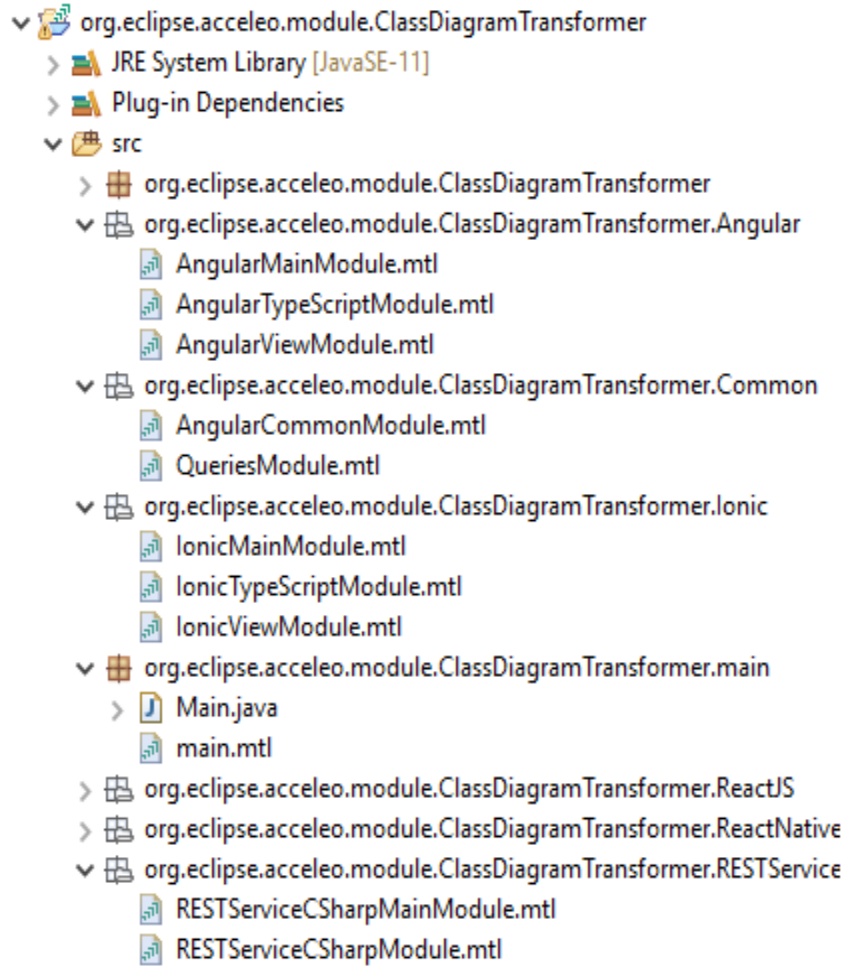


Figure - 4.3: Code structure of code generator

```
[comment encoding = UTF-8 /]
[module main('http://www.eclipse.org/uml2/5.0.0/UML' /)]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::Angular::AngularMainModule/]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::Ionic::IonicMainModule/]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::ReactJS::ReactJSMainModule/]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::ReactNative::ReactNativeMainModule/]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::RESTService::RESTServiceCSharpMainModule/]

[template public mainModel(aModel : Model)]
  [comment @main /]
  [generateAngularCode(aModel)/]
  [generateIonicCode(aModel)/]
  [generateRESTServiceCSharpCode(aModel)/]
[/template]
```

Figure - 4.5: Main module of Acceleo Project

```

[comment encoding = UTF-8 /]
[module AngularMainModule('http://www.eclipse.org/uml2/5.0.0/UML')]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::Angular::AngularTypeScriptModule/]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::Angular::AngularViewModule/]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::Common::AngularCommonModule/]

[template public generateAngularCode(aModel : Model)]
[file ('Angular/App/app.module.ts', false, 'UTF-8')]
    [modelToAngularAppModule(aModel)/]
[/file]

[file ('Angular/App/routing/routing.module.ts', false, 'UTF-8')]
    [modelToAngularRoutingComponent(aModel)/]
[/file]

[file ('Angular/App/app.component.html', false, 'UTF-8')]
    [modelToAngularAppComponentView(aModel)/]
[/file]

[file ('Angular/App/app.component.ts', false, 'UTF-8')]
    [modelToAngularAppComponentCode(aModel)/]
[/file]

[for (c: Class | aModel.ownedElement->filter(Class))]
    [file ('Angular/App/shared/'+c.name+'.model.ts', false, 'UTF-8')]
        [classToTypeScriptInterface(c)/]
    [/file]

    [file ('Angular/App/shared/'+c.name+'.service.ts', false, 'UTF-8')]
        [classToTypescriptService(c)/]
    [/file]

    [file ('Angular/App/'+c.name+'s/'+c.name+'/'+c.name+'.component.html', false, 'UTF-8')]
        [classToAngularComponentView(c)/]
    [/file]

    [file ('Angular/App/'+c.name+'s/'+c.name+'/'+c.name+'.component.ts', false, 'UTF-8')]
        [classToAngularComponentCode(c)/]
    [/file]

    [file ('Angular/App/'+c.name+'s/'+c.name+'/'+c.name+'-list.component.html', false, 'UTF-8')]
        [classToAngularComponentListView(c)/]
    [/file]

    [file ('Angular/App/'+c.name+'s/'+c.name+'/'+c.name+'-list.component.ts', false, 'UTF-8')]
        [classToAngularComponentListCode(c)/]
    [/file]
[/for]
[/template]

```

Figure - 4.6: Angular module of Acceleo Project

```

[comment encoding = UTF-8 /]
[module IonicMainModule('http://www.eclipse.org/uml2/5.0.0/UML')]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::Ionic::IonicTypeScriptModule/]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::Ionic::IonicViewModule/]

[template public generateIonicCode(aModel : Model)]
[file ('Ionic/App/app.module.ts', false, 'UTF-8')]
    [modelToIonicAppModule(aModel)/]
[/file]
[file ('Ionic/App/app-routing.module.ts', false, 'UTF-8')]
    [modelToIonicRoutingModule(aModel)/]
[/file]
[file ('Ionic/App/app.component.ts', false, 'UTF-8')]
    [modelToIonicAppComponentCode(aModel)/]
[/file]
[file ('Ionic/App/app.component.html', false, 'UTF-8')]
    [modelToIonicAppComponentView(aModel)/]
[/file]
[file ('Ionic/App/app.component.scss', false, 'UTF-8')]
    [modelToCss(aModel)/]
[/file]
[for (c: Class | aModel.ownedElement->filter(Class))]
    [file ('Ionic/App/Model/'+c.name+'.model.ts', false, 'UTF-8')]
        [classToTypeScriptInterface(c)/]
    [/file]
    [file ('Ionic/App/Services/'+c.name+'.service.ts', false, 'UTF-8')]
        [classToTypeScriptService(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+'.module.ts', false, 'UTF-8')]
        [classToIonicPageModule(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+'.page.ts', false, 'UTF-8')]
        [classToIonicPageCode(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+'.page.spec.ts', false, 'UTF-8')]
        [classToIonicPageSpecsCode(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+'.page.html', false, 'UTF-8')]
        [classToIonicPageView(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+'.page.scss', false, 'UTF-8')]
        [classToCss(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+ '-list.module.ts', false, 'UTF-8')]
        [classToIonicPageListModule(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+ '-list.page.ts', false, 'UTF-8')]
        [classToIonicPageListCode(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+ '-list.page.html', false, 'UTF-8')]
        [classToIonicPageListView(c)/]
    [/file]
    [file ('Ionic/App/'+c.name.toLowerCase()+ '/' +c.name.toLowerCase()+ '-list.page.scss', false, 'UTF-8')]
        [classToCss(c)/]
    [/file]
[/for]
[/template]

```

Figure - 4.7: Ionic module of Acceleo Project

```

[comment encoding = UTF-8 /]
[module RESTServiceCSharpMainModule('http://www.eclipse.org/uml2/5.0.0/UML')]
[import org::eclipse::acceleo::module::ClassDiagramTransformer::RESTService::RESTServiceCSharpModule/]

[template public generateRESTServiceCSharpCode(aModel : Model)]
  [file (aModel.name+'Api/Model/'+aModel.name + 'Context.cs', false, 'UTF-8')]
    [modelToContextClass(aModel)/]
  [/file]

  [for (c: Class | aModel.ownedElement->filter(Class))]
    [file (c.getModel().name+'Api/Model/'+c.name + '.cs', false, 'UTF-8')]
      [classToEFCorePOCO(c)/]
    [/file]

    [file (c.getModel().name+'Api/Controllers/'+c.name + 'Controller.cs', false, 'UTF-8')]
      [classToWebAPICoreController(c)/]
    [/file]

  [/for]
[/template]

```

Figure - 4.8: ASP.NET Web API module of Acceleo Project

Chapter 5

Validation and verification

CHAPTER 5: VALIDATION

After implementing a proposed methodology, validation is an important step in the field of research and especially in Model Driven Software Engineering. In this section the proposed and implemented methodology is discussed in detail. We have validated our proposed methodology by two case studies. One is Auditing System discussed in **Section 5.1** and other is LEXPERT (Lab Expert which is Pathology lab Management System) discussed in **Section 5.2**.

5.1 Auditing System – A Case Study

Auditing System is the web based application for entering the data of audit performed by auditor. This is a small application already developed in ASP.NET Core. We choose this as best case study for our proposed methodology. This case study is divided into four sections. Firstly, the requirements of the real time chat application are discussed in **Section 5.1.1**. Secondly, **Section 5.1.2** contains the UML class diagram applied profile to present the system architecture of the required system. **Section 5.1.3** shows the transformation results in the form of generated code. And finally, **Section 5.1.4** contains verification of the system.

5.1.1 Requirements of Auditing System

This system is for keeping the record of auditing that is auditing for a company. The functional requirements for this system are given below

- There should be many auditing companies.
- One auditor can do job only in one company
- There should be many employees in one company
- Auditing of client company can be done by only one company
- The user could create, read, update and delete auditing company
- The user could create, read, update and delete auditing employee
- The user could create, read, update and delete auditing client
- The user could create, read, update and delete auditing audit

5.1.2 Modeling

The class diagram of auditing system is designed and the designed profile and data types are applied. The class diagram after applying the profile is show in **Figure 5.1**.

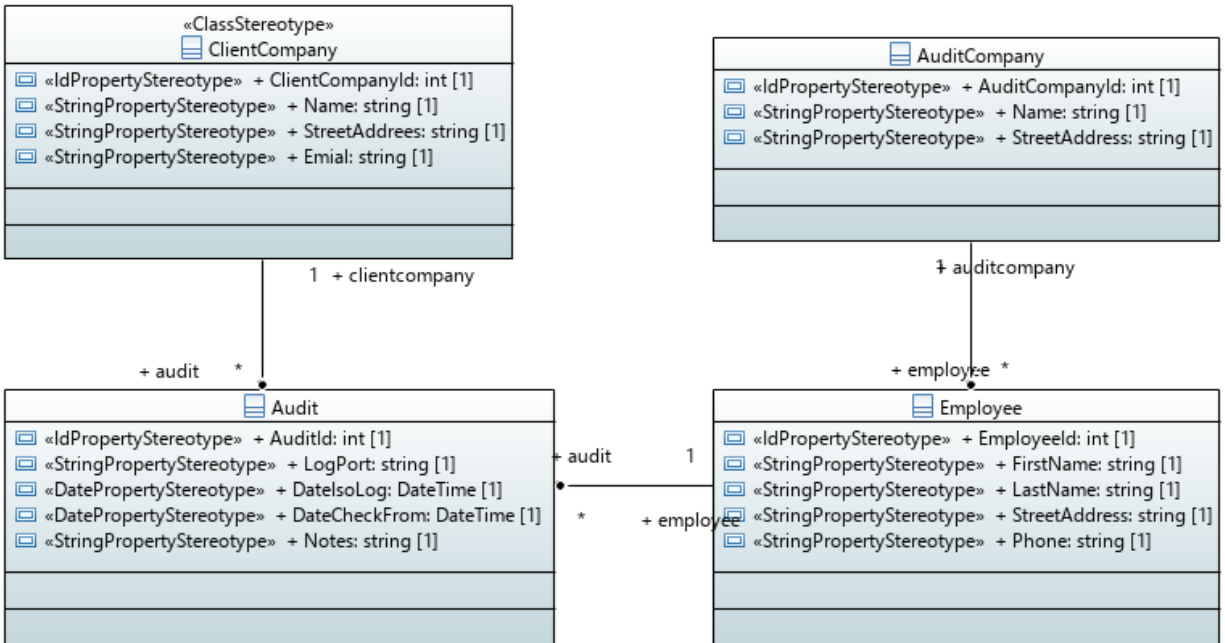


Figure - 5.1: Class diagram of Auditing System with applied stereotypes

5.1.3 Code Generation

This section highlights the code generation process from our proposed transformation engine. Required input for the transformation is Class diagram which is UML model with extension .uml. UML model with .uml extension is selected as input model and target folder on is provided as output folder for generated code files. Code is generated for ASP.NET Web API, Angular and Ionic. The structures of generated files are shown in the **Figure 5.2, 5.3 and 5.4**. The generated code is show in Figures 5.5 to 5.17

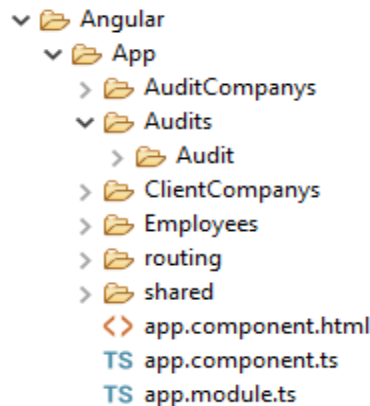


Figure - 5.2: Files Structure of Angular's Generated Code

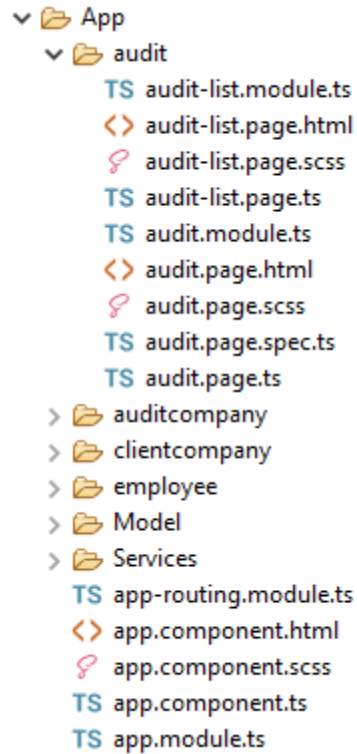


Figure - 5.3: Files Structure of Ionic's Generated Code

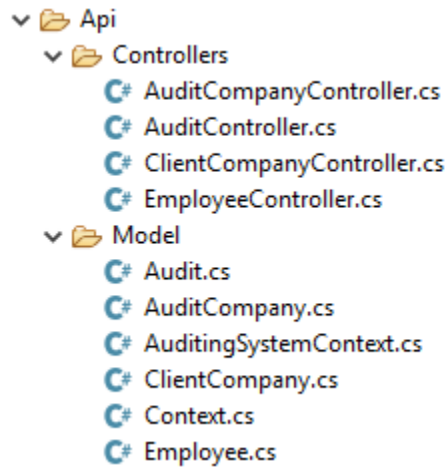


Figure - 5.4: Files Structure of Web API's Generated Code


```

1      using System;
2      using System.Collections.Generic;
3      using System.ComponentModel.DataAnnotations;
4      using System.ComponentModel.DataAnnotations.Schema;
5      using System.Linq;
6      using System.Threading.Tasks;
7
8      namespace AuditingSystemApi.Models
9      {
10         public class Employee
11         {
12             public Employee ()
13             {
14             }
15
16             [Key]
17
18             public int EmployeeId { get; set; }
19
20             [Column(TypeName = "varchar(40)")]
21             public string FirstName { get; set; }
22
23             [Column(TypeName = "varchar(40)")]
24             public string LastName { get; set; }
25
26             [Column(TypeName = "varchar(150)")]
27             public string StreetAddress { get; set; }
28
29             [Column(TypeName = "varchar(11)")]
30             public string Phone { get; set; }
31
32
33             public int AuditCompanyId { get; set; }
34             public AuditCompany AuditCompany { get; set; }
35
36             public IList<Audit> Audits { get; set; }
37
38         }
39     }
40
41
42

```

Figure - 5.5: Generated Code for Model of EF Core

```

1      using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
2      using Microsoft.EntityFrameworkCore;
3      using System;
4      using System.Collections.Generic;
5      using System.Linq;
6      using System.Threading.Tasks;
7
8      namespace AuditingSystemApi.Models
9      {
10         public class AuditingSystemContext : IdentityDbContext
11         {
12             public AuditingSystemContext(DbContextOptions<AuditingSystemContext> options) : base(options)
13             {
14             }
15
16             public DbSet<Employee> Employee { get; set; }
17             public DbSet<ClientCompany> ClientCompany { get; set; }
18             public DbSet<Audit> Audit { get; set; }
19             public DbSet<AuditCompany> AuditCompany { get; set; }
20
21             protected override void OnModelCreating(ModelBuilder modelBuilder)
22             {
23                 base.OnModelCreating(modelBuilder);
24             }
25         }
26     }
27

```

Figure - 5.6: Generated Code for Context of EF

```

1 using System.Threading.Tasks;
2 using Microsoft.AspNetCore.Http;
3 using Microsoft.AspNetCore.Mvc;
4 using Microsoft.EntityFrameworkCore;
5 using System.Collections.Generic;
6 using System.Linq;
7 using AuditingSystemApi.Models;
8
9 namespace AuditingSystemApi.Controllers
10 {
11     [Route("api/[controller]")]
12     [ApiController]
13     public class EmployeeController : ControllerBase
14     {
15         private readonly A Case Study - Auditing SystemContext _context;
16
17         public EmployeeController(A Case Study - Auditing SystemContext context)
18         {
19             _context=context;
20         }
21
22         // GET: api/Employee
23         [HttpGet]
24         public IEnumerable<Employee> GetEmployee()
25         {
26             return _context.Employee;
27         }
28
29         // GET: api/Employee/5
30         [HttpGet("{id}")]
31         public async Task<IActionResult> GetEmployee([FromRoute] int id)
32         {
33             if (!ModelState.IsValid)
34             {
35                 return BadRequest(ModelState);
36             }
37
38             var employee = await _context.Employee.FindAsync(id);
39
40             if (employee == null)
41             {
42                 return NotFound();
43             }
44
45             return Ok(employee);
46         }
47
48         // PUT: api/Employee/5
49         [HttpPut("{id}")]
50         public async Task<IActionResult> PutEmployee([FromRoute] int id, [FromBody] Employee employee)
51         {
52             if (!ModelState.IsValid)
53             {
54                 return BadRequest(ModelState);
55             }
56
57         }
58     }
59 }

```

Figure - 5.7: Generated Code for ASP.NET Web API Controller

```

1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { FormsModule, ReactiveFormsModule } from '@angular/forms'
4 import { HttpClientModule } from '@angular/common/http'
5 import { BrowserModuleAnimationsModule } from '@angular/platform-browser/animations';
6 import { ToastrModule } from 'ngx-toastr';
7 import { AppComponent } from './app.component';
8 import { MatToolbarModule, MatIconModule, MatSidenavModule, MatListModule, MatButtonModule } from '@angular/material';
9
10 import { RoutingModule } from './routing/routing.module';
11 import { MainNavComponent } from './main-nav/main-nav.component';
12 import { LayoutModule } from '@angular/cdk/layout'
13
14 import { EmployeeComponent } from './Employees/Employee/Employee.component';
15 import { EmployeeListComponent } from './Employees/Employee/Employee-list.component';
16 import { EmployeesService } from './shared/Employee.service';
17 import { ClientCompanyComponent } from './ClientCompanys/ClientCompany/ClientCompany.component';
18 import { ClientCompanyListComponent } from './ClientCompanys/ClientCompany/ClientCompany-list.component';
19 import { ClientCompanyService } from './shared/ClientCompany.service';
20 import { AuditComponent } from './Audits/Audit/Audit.component';
21 import { AuditListComponent } from './Audits/Audit/Audit-list.component';
22 import { AuditService } from './shared/Audit.service';
23 import { AuditCompanyComponent } from './AuditCompanys/AuditCompany/AuditCompany.component';
24 import { AuditCompanyListComponent } from './AuditCompanys/AuditCompany/AuditCompany-list.component';
25 import { AuditCompanyService } from './shared/AuditCompany.service';
26
27 @NgModule({
28   declarations: [
29     AppComponent,
30     MainNavComponent,
31     EmployeeComponent,
32     EmployeeListComponent,
33     ClientCompanyComponent,
34     ClientCompanyListComponent,
35     AuditComponent,
36     AuditListComponent,
37     AuditCompanyComponent,
38     AuditCompanyListComponent,
39   ],
40   imports: [
41     BrowserModule,
42     FormsModule,
43     HttpClientModule,

```

Figure - 5.8: Generated Code for app.module.ts

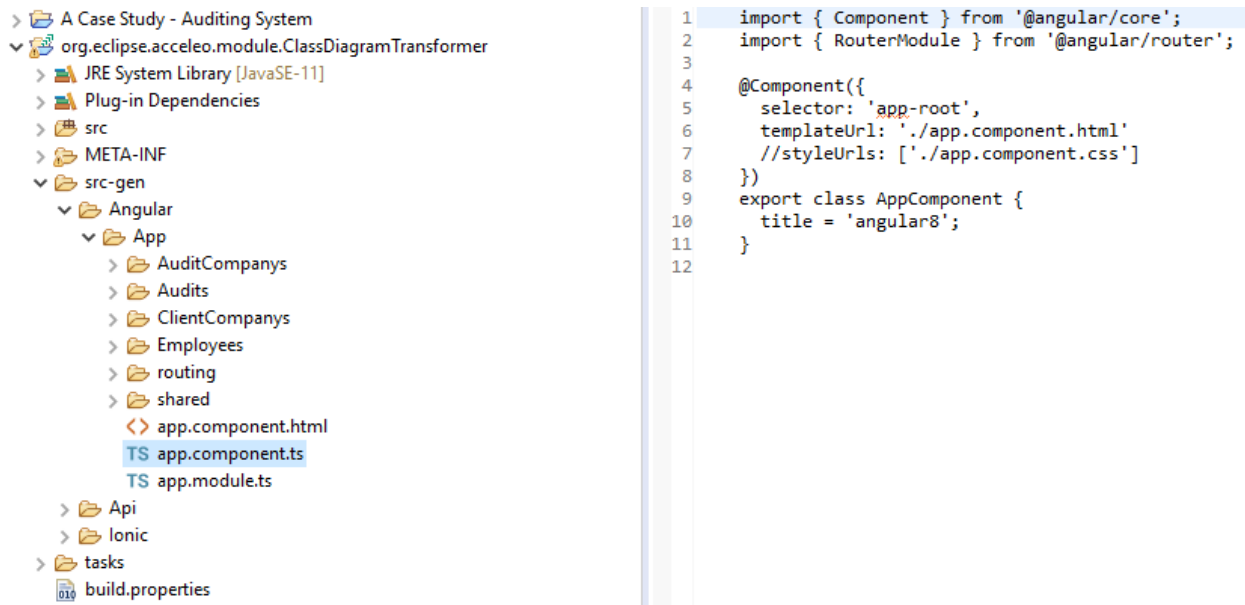


Figure - 5.9: Generated Code for Angular's app.component.ts

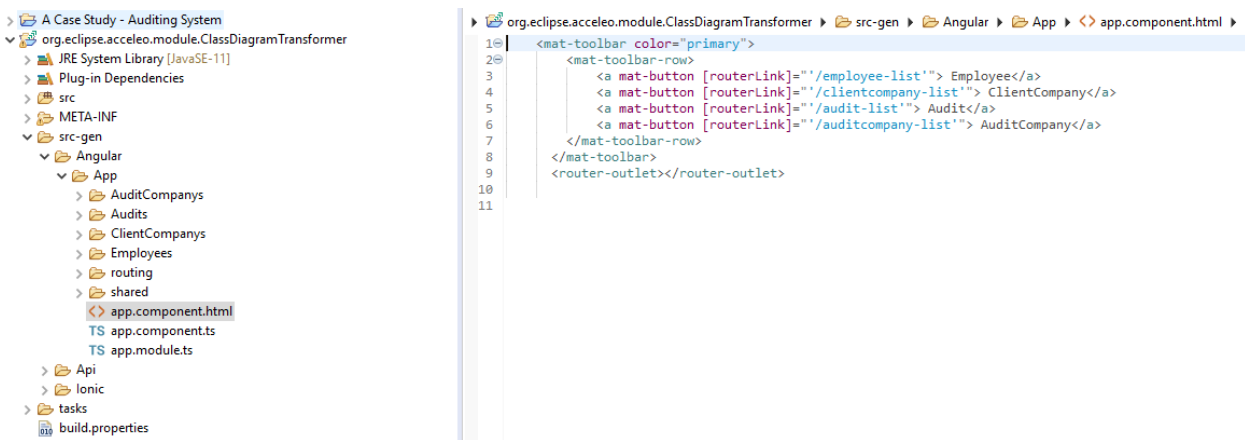


Figure - 5.10: Generated Code for Angular's app.component.html

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

```

```

<div class="row">
  <div class="col-md-5" style="margin: 10px">
    <form [formGroup]="employeeForm" autocomplete="off" (ngSubmit)="onSubmit()">
      Employee <button class="btn btn-basic btn-lg" (click)="navigateToEmployeeList()><i class="fas fa-arrow-alt-circle-left"></i> Back</button>
      <input type="hidden" name="employeeId" [value]="service.formData.employeeId">
      <div class="form-group">
        <div class="input-group">
          <div class="input-group-prepend">
            <div class="input-group-text bg-white">
              <i class="fas fa-user-circle" [ngClass]="{ 'green-icon': employeeForm.controls['firstName'].valid, 'red-icon': employeeForm.c
            </div>
          </div>
          <input name="firstName" formControlName="firstName" type="text" class="form-control" placeholder = "First Name" type="text">
        </div>
      </div>
      <div class="form-group">
        <div class="input-group">
          <div class="input-group-prepend">
            <div class="input-group-text bg-white">
              <i class="fas fa-user-circle" [ngClass]="{ 'green-icon': employeeForm.controls['lastName'].valid, 'red-icon': employeeForm.cc
            </div>
          </div>
          <input name="lastName" formControlName="lastName" type="text" class="form-control" placeholder = "Last Name" type="text">
        </div>
      </div>
      <div class="form-group">
        <div class="input-group">
          <div class="input-group-prepend">
            <div class="input-group-text bg-white">
              <i class="fas fa-home" [ngClass]="{ 'green-icon': employeeForm.controls['streetAddress'].valid, 'red-icon': employeeForm.cont
            </div>
          </div>
          <input name="streetAddress" formControlName="streetAddress" type="text" class="form-control" placeholder = "" type="text">
        </div>
      </div>
      <div class="form-group">
        <div class="input-group">
          <div class="input-group-prepend">
            <div class="input-group-text bg-white">
              <i class="fas fa-phone" [ngClass]="{ 'green-icon': employeeForm.controls['phone'].valid, 'red-icon': employeeForm.controls['p
            </div>
          </div>
          <input name="phone" formControlName="phone" type="text" class="form-control" placeholder = "Phone" type="text">
        </div>
      </div>
    </form>
  </div>

```

Figure - 5.11: Generated Code for Angular's View Components – Employee

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

```

```

import { Component, OnInit } from '@angular/core';
import { FormBuilder, NgForm, FormGroup, Validators } from '@angular/forms';
import { ToastrService } from 'ngx-toastr';
import { Router } from '@angular/router';
import { EmployeeService } from 'src/app/shared/Employee.service';

import { AuditCompanyService } from 'src/app/shared/AuditCompany.service';
import { AuditCompany } from 'src/app/shared/AuditCompany.model';

@Component({
  selector: 'app-employee',
  templateUrl: './Employee.component.html',
  styles: []
})
export class EmployeeComponent implements OnInit {
  employeeForm: FormGroup;
  constructor(private service:EmployeeService, private toastr:ToastrService, private router:Router,private formBuilder: FormBuilder
, private AuditCompanyService:AuditCompanyService
) { }

  ngOnInit() {
    this.createFormBuilder();
    this.employeeForm.valueChanges.subscribe(value=>{
      this.service.formData=value;
    })

    this.AuditCompanyService.refreshList();
  }

  createFormBuilder() {
    if(this.service.formData==null)
      this.setDefaultFormValues();
    this.employeeForm=this.formBuilder.group({
      firstName:[this.service.formData.firstName,[Validators.pattern(""),Validators.required,Validators.minLength(1),Validators.maxLength(40)]],
      lastName:[this.service.formData.lastName,[Validators.pattern(""),Validators.required,Validators.minLength(1),Validators.maxLength(40)]],
      streetAddress:[this.service.formData.streetAddress,[Validators.minLength(1),Validators.maxLength(150)]],
      phone:[this.service.formData.phone,[Validators.minLength(11),Validators.maxLength(11)]],
      auditCompany: [null,
        [Validators.required]
      ],
    });
  }

```

Figure - 5.12: Generated Code for Code for Angular's Components – Employee

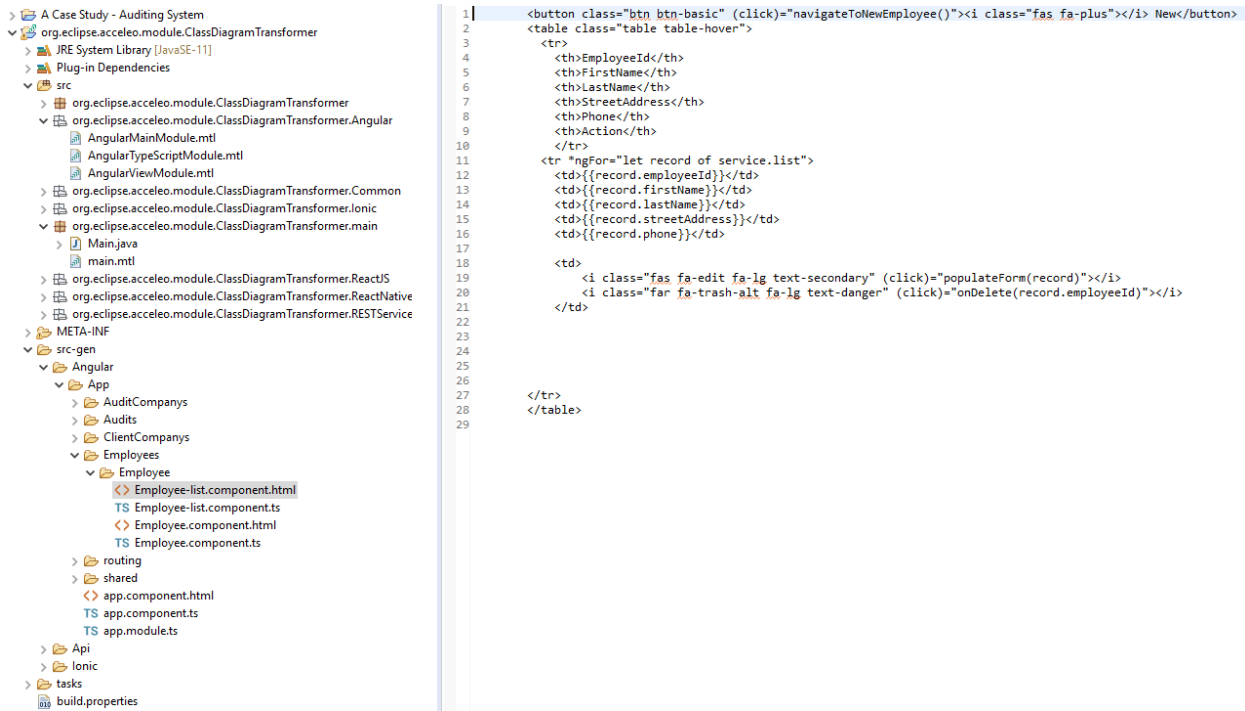


Figure - 5.13: Generated Code for Angular's View of Components-List – Employee

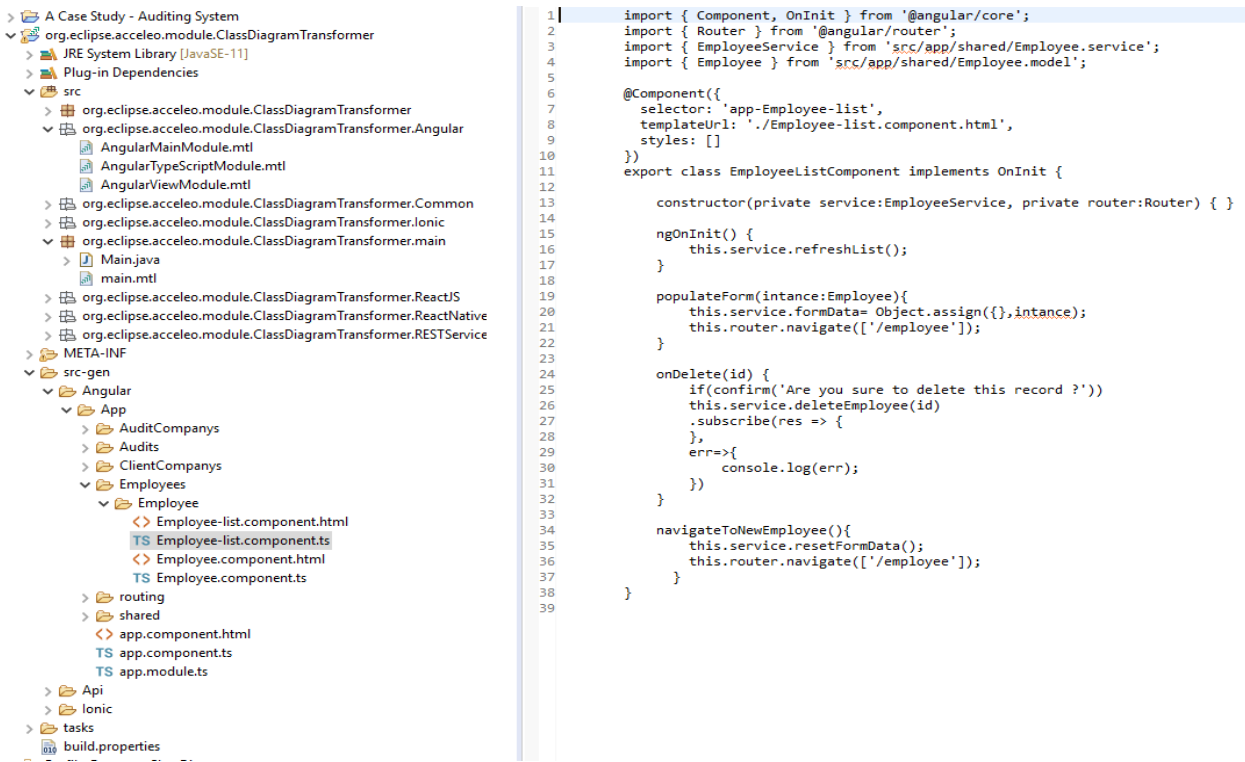


Figure - 5.14: Generated Code for Code of Components-List – Employee

```

1 import { Component, OnInit } from '@angular/core';
2 import { FormBuilder, NgForm, FormGroup, Validators } from '@angular/forms';
3 import { ToastrService } from 'ngx-toastr';
4 import { Router } from '@angular/router';
5 import { AuditService } from 'src/app/shared/Audit.service';
6
7 import { ClientService } from 'src/app/shared/Client.service';
8 import { Client } from 'src/app/shared/Client.model';
9
10 import { EmployeeService } from 'src/app/shared/Employee.service';
11 import { Employee } from 'src/app/shared/Employee.model';
12
13 @Component({
14   selector: 'app-Audit',
15   templateUrl: './Audit.component.html',
16   styles: []
17 })
18 export class AuditComponent implements OnInit {
19
20   auditForm: FormGroup;
21   constructor(private service: AuditService, private toastr: ToastrService, private router: Router, private formBuilder: FormBuilder
22     , private clientService: ClientService
23     , private employeeService: EmployeeService
24   ) { }
25 }
26
27 ngOnInit() {
28   this.createFormBuilder();
29   this.auditForm.valueChanges.subscribe(value=>{
30     this.service.formData=value;
31   })
32   this.clientService.refreshList();
33   this.employeeService.refreshList();
34 }
35
36 }
37
38 createFormBuilder() {
39   if(this.service.formData==null)

```

Figure - 5.15: Generated Code for Angular’s Code Components-List – Employee

```

1 import { Component, OnInit } from '@angular/core';
2 import { FormBuilder, NgForm, FormGroup, Validators } from '@angular/forms';
3 import { ToastrService } from 'ngx-toastr';
4 import { Router } from '@angular/router';
5 import { AuditService } from 'src/app/shared/Audit.service';
6
7 import { ClientService } from 'src/app/shared/Client.service';
8 import { Client } from 'src/app/shared/Client.model';
9
10 import { EmployeeService } from 'src/app/shared/Employee.service';
11 import { Employee } from 'src/app/shared/Employee.model';
12
13 @Component({
14   selector: 'app-Audit',
15   templateUrl: './Audit.component.html',
16   styles: []
17 })
18 export class AuditComponent implements OnInit {
19
20   auditForm: FormGroup;
21   constructor(private service: AuditService, private toastr: ToastrService, private router: Router, private formBuilder: FormBuilder
22     , private clientService: ClientService
23     , private employeeService: EmployeeService
24   ) { }
25 }
26
27 ngOnInit() {
28   this.createFormBuilder();
29   this.auditForm.valueChanges.subscribe(value=>{
30     this.service.formData=value;
31   })
32   this.clientService.refreshList();
33   this.employeeService.refreshList();
34 }
35
36 }
37
38 createFormBuilder() {
39   if(this.service.formData==null)

```

Figure - 5.16: Generated Code for Angular’s Model Classes – Employee

```

> A Case Study - Auditing System
> org.eclipse.acceleo.module.ClassDiagramTransformer
> JRE System Library [JavaSE-11]
> Plug-in Dependencies
> src
  > org.eclipse.acceleo.module.ClassDiagramTransformer
  > org.eclipse.acceleo.module.ClassDiagramTransformer.Angular
  > org.eclipse.acceleo.module.ClassDiagramTransformer.Common
  > org.eclipse.acceleo.module.ClassDiagramTransformer.Ionic
  > org.eclipse.acceleo.module.ClassDiagramTransformer.main
  > org.eclipse.acceleo.module.ClassDiagramTransformer.ReactJS
  > org.eclipse.acceleo.module.ClassDiagramTransformer.ReactNative
  > org.eclipse.acceleo.module.ClassDiagramTransformer.RESTService
> META-INF
> src-gen
  > Angular
    > App
      > AuditCompany.s
      > Audits
      > ClientCompany.s
      > Employees
      > routing
      > shared
        TS Audit.model.ts
        TS Audit.service.ts
        TS AuditCompany.model.ts
        TS AuditCompany.service.ts
        TS ClientCompany.model.ts
        TS ClientCompany.service.ts
        TS Employee.model.ts
        TS Employee.service.ts
        <> app.component.html
        TS app.components.ts
        TS app.module.ts
    > Api
    > Ionic
  > tasks
  build.properties

```

```

1  import { Injectable } from '@angular/core';
2  import { Employee } from './Employee.model';
3  import { HttpClient } from '@angular/common/http';
4
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class EmployeeService {
9    formData:Employee;
10   readonly rootURL='https://localhost:44397/api/'
11   list:Employee[];
12   constructor(private http:HttpClient) { }
13
14   postEmployee(){
15     return this.http.post(this.rootURL+'Employee',this.formData);
16   }
17
18   putEmployee(){
19     return this.http.put(this.rootURL+'Employee/'+this.formData.employeeId,this.formData);
20   }
21
22   deleteEmployee(id){
23     return this.http.delete(this.rootURL+'Employee/'+id);
24   }
25
26   refreshList(){
27     this.http.get(this.rootURL+'Employee')
28     .toPromise()
29     .then(res => this.list = res as Employee[]);
30   }
31
32   resetFormData(){
33     this.formData={
34       employeeId:0,
35       firstName:'',
36       lastName:'',
37       streetAddress:'',
38       phone:'',
39
40     }
41     auditcompany:null,
42   }
43 }
44
45

```

Figure - 5.17: Generated Code for Angular's Services – Employee

```

> A Case Study - Auditing System
> A Case Study - Lexpert
> org.eclipse.acceleo.module.ClassDiagramTransformer
> JRE System Library [JavaSE-11]
> Plug-in Dependencies
> src
  > META-INF
  > src-gen
    > Angular
    > Api
    > Ionic
      > App
        > audit
        > auditcompany
        > clientcompany
        > employee
          TS employee-list.module.ts
          <> employee-list.page.html
          <> employee-list.page.scss
          TS employee-list.page.ts
          TS employee.module.ts
          <> employee.page.html
          <> employee.page.scss
          TS employee.page.spec.ts
          TS employee.page.ts
        > Model
        > Services
          TS app-routing.module.ts
          <> app.component.html
          <> app.components.scss
          TS app.components.ts
          TS app.module.ts

```

```

1  <ion-header>
2  <ion-toolbar>
3    <ion-buttons slot="start">
4      <ion-menu-button></ion-menu-button>
5    </ion-buttons>
6  <ion-title>
7    Employee
8  </ion-title>
9  </ion-toolbar>
10 </ion-header>
11
12 <ion-content>
13 <form [formGroup]="employeeForm" autocomplete="off">
14 <ion-button color="primary" (click)="navigateToEmployeeList()">Back</ion-button>
15
16 <ion-item class="input-item">
17 <ion-label position="floating">First Name</ion-label>
18 <ion-input type="text" formControlName="firstName" clearInput required</ion-input>
19 </ion-item>
20 <ion-item class="input-item">
21 <ion-label position="floating">Last Name</ion-label>
22 <ion-input type="text" formControlName="lastName" clearInput required</ion-input>
23 </ion-item>
24 <ion-item class="input-item">
25 <ion-label position="floating">Street Address</ion-label>
26 <ion-input type="text" formControlName="streetAddress" clearInput required</ion-input>
27 </ion-item>
28 <ion-item class="input-item">
29 <ion-label position="floating">Phone</ion-label>
30 <ion-input type="text" formControlName="phone" clearInput required</ion-input>
31 </ion-item>
32
33 <ion-item>
34 <ion-label position="floating"></ion-label>
35 <ion-select formControlName="auditCompany">
36 <ion-select-option value="undefined" checked="true">Choose AuditCompany</ion-select-option>
37 <ion-select-option *ngFor="let comp of AuditCompanyService.list" [value]="comp">{{ comp.name }}</ion-select-opt
38 </ion-select>
39 </ion-item>

```

Figure - 5.18: Generated Code for Ionic's View – Employee

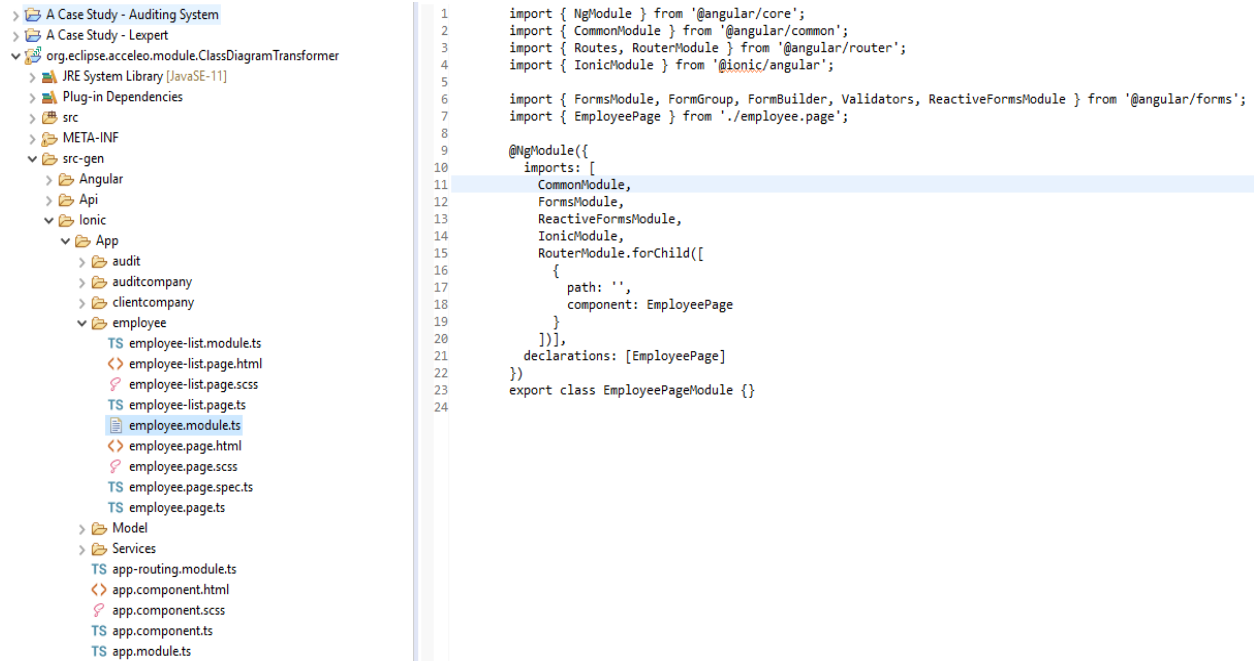


Figure - 5.19: Generated Code for Ionic’s Module – Employee

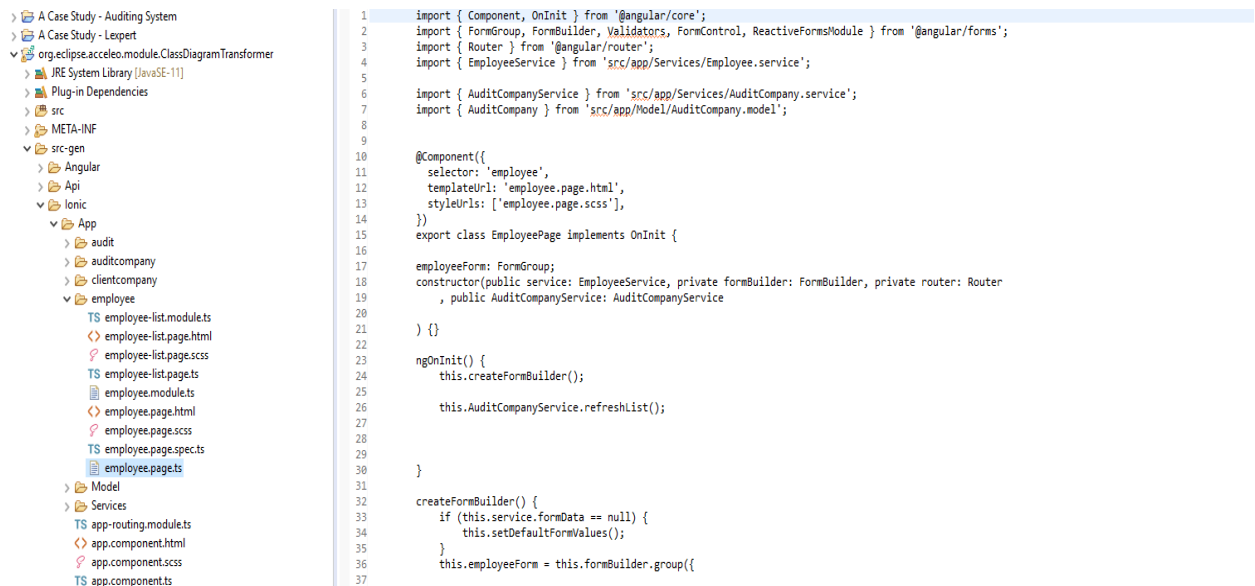


Figure - 5.20: Generated Code for Ionic’s Typescript of page – Employee

5.1.4 Verification

For verification of generated code, its compilation and execution are necessary. Therefore, we deploy this code in angular application. We created an angular application and pasted our generated code accordingly. The code and generated output are shown in **Figures 5.21 – 5.28**.

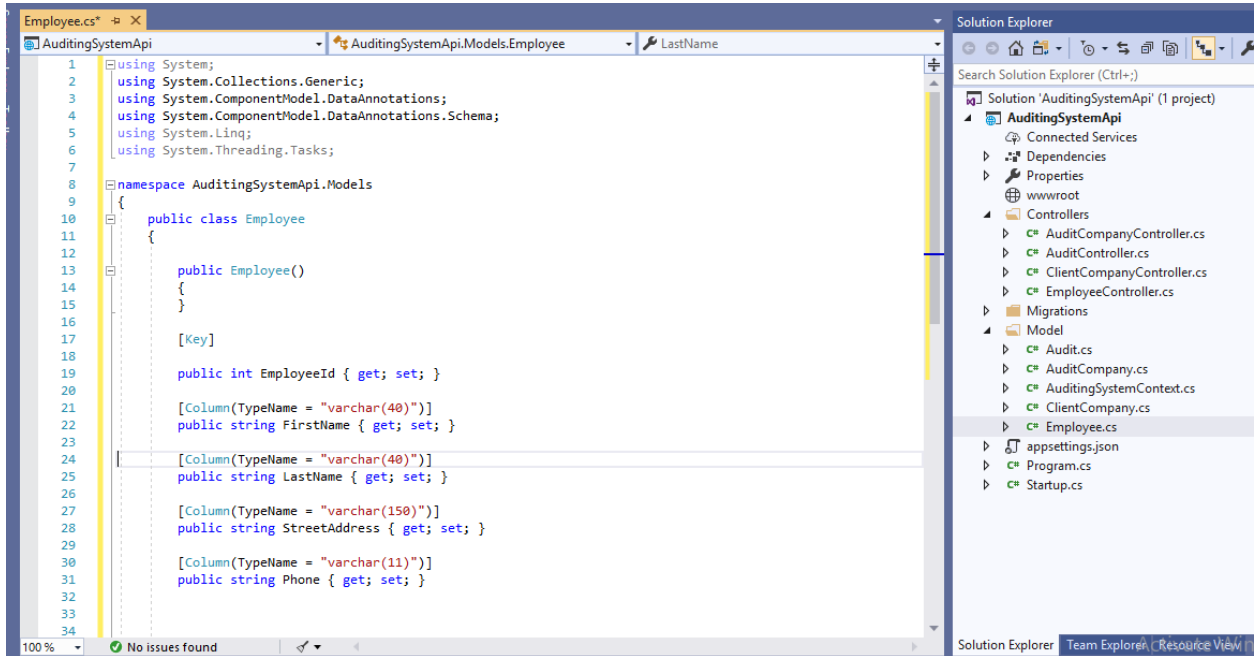


Figure - 5.21: Code Deployed for Auditing System's Web API (C#)

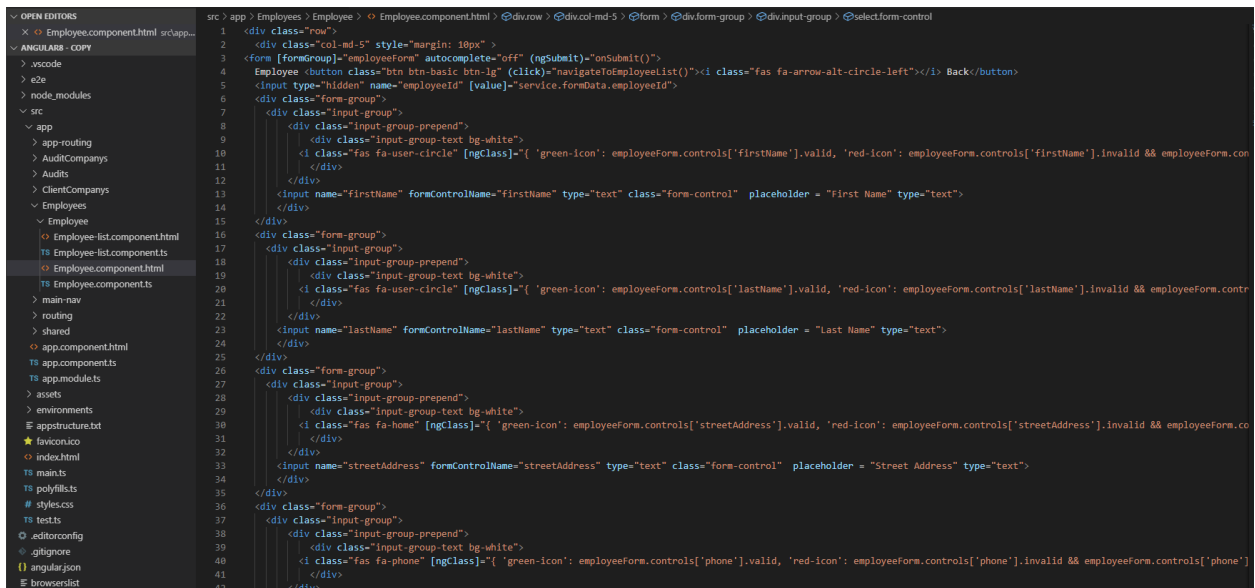


Figure - 5.22: Code Deployed for Auditing System's Angular code

```

EXPLORER
  employee.page.html X
  src > app > employee > employee.page.html > ion-content > form > ion-item.input-item
  1 <ion-header>
  2 <ion-toolbar>
  3 <ion-buttons slot="start">
  4 <ion-menu-button></ion-menu-button>
  5 </ion-buttons>
  6 <ion-title>
  7 Employee
  8 </ion-title>
  9 </ion-toolbar>
  10 </ion-header>
  11
  12 <ion-content>
  13 <form [formGroup]="employeeForm" autocomplete="off">
  14 <ion-button color="primary" (click)="navigateToEmployeeList()">Back</ion-button>
  15
  16 <ion-item class="input-item">
  17 <ion-label position="floating">First Name</ion-label>
  18 <ion-input type="text" formControlName="firstName" clearInput required</ion-input>
  19 </ion-item>
  20 <ion-item class="input-item">
  21 <ion-label position="floating">Last Name</ion-label>
  22 <ion-input type="text" formControlName="lastName" clearInput required</ion-input>
  23 </ion-item>
  24 <ion-item class="input-item">
  25 <ion-label position="floating">Street Address</ion-label>
  26 <ion-input type="text" formControlName="streetAddress" clearInput required</ion-input>
  27 </ion-item>
  28 <ion-item class="input-item">
  29 <ion-label position="floating">Phone</ion-label>
  30 <ion-input type="text" formControlName="phone" clearInput required</ion-input>
  31 </ion-item>
  32
  33 <ion-item>
  34 <ion-label position="floating"></ion-label>
  35 <ion-select formControlName="auditCompany">
  36 <ion-select-option value="undefined" checked="true">Choose AuditCompany</ion-select-option>
  37 <ion-select-option *ngFor="let comp of AuditCompanyService.list" [value]="comp"{{ comp.name }}</ion-select-option>
  38 </ion-select>
  39 </ion-item>
  40
  41
  
```

Figure - 5.23: Code Deployed for Auditing System’s Aionic code

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS E:\MS Research\Project\angular8 - Copy> ng serve

Date: 2019-09-03T19:40:56.119Z
Hash: 3e101ec9e63f1470c013
Time: 19680ms
chunk {main} main.js, main.js.map (main) 83.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 248 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 202 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 6.88 MB [initial] [rendered]
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
i [wdm]: Compiled successfully.
  
```

Figure - 5.24: Compiling Angular app

```

> ng.cmd run app:serve --host=localhost --port=8100
[ng] i [wds]: Project is running at http://localhost:8100/webpack-dev-server/
[ng] i [wds]: webpack output is served from /
[ng] i [wds]: 404s will fallback to //index.html
[ng] chunk {audit-audit-list-module} audit-audit-list-module.js, audit-audit-list-module.js.map (audit-audit-list-module) 6.83 kB [rendered]
[ng] chunk {audit-audit-module} audit-audit-module.js, audit-audit-module.js.map (audit-audit-module) 11 kB [rendered]
[ng] chunk {auditcompany-auditcompany-list-module} auditcompany-auditcompany-list-module.js, auditcompany-auditcompany-list-module.js.map (auditcompany-auditcompany-list-module) 7.29 kB [rendered]
[ng] chunk {auditcompany-auditcompany-module} auditcompany-auditcompany-module.js, auditcompany-auditcompany-module.js.map (auditcompany-auditcompany-module) 8.86 kB [rendered]
[ng] chunk {clientcompany-clientcompany-list-module} clientcompany-clientcompany-list-module.js, clientcompany-clientcompany-list-module.js.map (clientcompany-clientcompany-list-module) 7.42 kB [rendered]
[ng] chunk {clientcompany-clientcompany-module} clientcompany-clientcompany-module.js, clientcompany-clientcompany-module.js.map (clientcompany-clientcompany-module) 9.58 kB [rendered]
[ng] chunk {common} common.js, common.js.map (common) 35.8 kB [rendered]
[ng] chunk {core-js-js} core-js-js.js, core-js-js.js.map (core-js-js) 78.7 kB [rendered]
[ng] chunk {css-shim-3ea8955c-3ea8955c-js} css-shim-3ea8955c-3ea8955c-js.js, css-shim-3ea8955c-3ea8955c-js.js.map (css-shim-3ea8955c-3ea8955c-js) 20.7 kB [rendered]
[ng] chunk {css-shim-js} css-shim-js.js, css-shim-js.js.map (css-shim-js) 20.7 kB [rendered]
[ng] chunk {dom-59290340-59290340-js} dom-59290340-59290340-js.js, dom-59290340-59290340-js.js.map (dom-59290340-59290340-js) 19 kB [rendered]
[ng] chunk {dom-js} dom-js.js, dom-js.js.map (dom-js) 19.3 kB [rendered]
[ng] chunk {employee-employee-list-module} employee-employee-list-module.js, employee-employee-list-module.js.map (employee-employee-list-module) 7.08 kB [rendered]
[ng] chunk {employee-employee-module} employee-employee-module.js, employee-employee-module.js.map (employee-employee-module) 10.8 kB [rendered]
[ng] chunk {focus-visible-70161a50-js} focus-visible-70161a50-js.js, focus-visible-70161a50-js.js.map (focus-visible-70161a50-js) 2.15 kB [rendered]
[ng] chunk {hardware-back-button-08f20350-js} hardware-back-button-08f20350-js.js, hardware-back-button-08f20350-js.js.map (hardware-back-button-08f20350-js) 2.06 kB [rendered]
[ng] chunk {index-f98adfd1-js} index-f98adfd1-js.js, index-f98adfd1-js.js.map (index-f98adfd1-js) 37.7 kB [rendered]
[ng] chunk {input-shims-09d70723-js} input-shims-09d70723-js.js, input-shims-09d70723-js.js.map (input-shims-09d70723-js) 13.5 kB [rendered]
[ng] chunk {ios-transition-5c2f00df-js} ios-transition-5c2f00df-js.js, ios-transition-5c2f00df-js.js.map (ios-transition-5c2f00df-js) 15.2 kB [rendered]
[ng] chunk {main} main.js, main.js.map (main) 28 kB [initial] [rendered]
[ng] chunk {md-transition-e0e9d421-js} md-transition-e0e9d421-js.js, md-transition-e0e9d421-js.js.map (md-transition-e0e9d421-js) 3.26 kB [rendered]
[ng] chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 278 kB [initial] [rendered]
[ng] chunk {runtime} runtime.js, runtime.js.map (runtime) 10.2 kB [entry] [rendered]
[ng] chunk {shadow-css-9e778f69-c68d0961-js} shadow-css-9e778f69-c68d0961-js.js, shadow-css-9e778f69-c68d0961-js.js.map (shadow-css-9e778f69-c68d0961-js) 14.8 kB [rendered]
[ng] chunk {status-tap-2758df1d-js} status-tap-2758df1d-js.js, status-tap-2758df1d-js.js.map (status-tap-2758df1d-js) 1.79 kB [rendered]
[ng] chunk {styles} styles.js, styles.js.map (styles) 108 kB [initial] [rendered]
[ng] chunk {swipe-back-78383f2a-js} swipe-back-78383f2a-js.js, swipe-back-78383f2a-js.js.map (swipe-back-78383f2a-js) 2.38 kB [rendered]
[ng] chunk {swiper-bundle-8d61f7c5-js} swiper-bundle-8d61f7c5-js.js, swiper-bundle-8d61f7c5-js.js.map (swiper-bundle-8d61f7c5-js) 175 kB [rendered]
[ng] chunk {tap-click-b300ec79-js} tap-click-b300ec79-js.js, tap-click-b300ec79-js.js.map (tap-click-b300ec79-js) 6.37 kB [rendered]
[ng] chunk {vendor} vendor.js, vendor.js.map (vendor) 4.89 MB [initial] [rendered]
[ng] Date: 2019-09-03T19:51:50.145Z - Hash: 9fc0d432c681da74bf8 - Time: 18125ms

[INFO] Development server running!

Local: http://localhost:8100

Use Ctrl+C to quit this process

[INFO] Browser window opened to http://localhost:8100!

[INFO] ... and 78 additional chunks
[ng] i [wds]: Compiled successfully.

```

Figure - 5.25: Compiling Ionic app

Employee	ClientCompany	Audit	AuditCompany
Employee	⊕ Back		
	First Name		
	Last Name		
	Street Address		
	Phone		
	▼		
Submit			

Figure - 5.26: Component View of Angular's Web App




Employee ClientCompany Audit AuditCompany					
+ New					
EmployeeId	FirstName	LastName	StreetAddress	Phone	Action
1	as	as	as	12345678909	 
2	werw	wer	wer	12345678909	 

Figure - 5.27: Component List View of Angular’s Web App

Figure - 5.28: Page View of Ionic’s Hybrid App

5.1.2 Conventional VS Proposed methodology

The conventional development is taking more time than developing through model driven methodology that we have proposed. This reduce the initial development time, improve the level of abstraction and improve the quality of application. The development of same domain application on mobile and web need minimum three skills, experience in hybrid app development (ionic), experience in web app development (angular) and experience in web service development (ASP.NET Web API).

For comparison of the proposed methodology with conventional method we have chosen 12 developers and divide with four groups. Each group had three developer with different skills. On developer of Ionic, one of angular and one of asp.net Web API are chosen in each group. The developers have develop the hybrid app, web app and RESTful service by conventional and model driven methodology. The comparison of development time is shown in the table 1. We found that model driven methodology is more than 5 times more time saving.

5.2 LEXPERT (Lab Expert)

LEXPERT (Lab Expert) is a pathology Lab Management System that is developed in WPF with WCF service and already available in market. We select LEXPERT as our case study in our proposed methodology for generating scaffolding CRUD code for web and mobile applications. The case study is divided into four sections. Firstly, the requirements of LEXPERT application are discussed in **Section 5.2.1**. Secondly, **Section 5.2.2** contains the UML class diagram with applied profile to present the system architecture of the required system. Furthermore, **Section 5.2.3** shows the transformation results in the form of generated code.

5.2.1 Requirements

This part gives the detailed description of the functional and non-functional requirements proposed by the client. The purpose of this project is to provide a friendly environment to maintain the record of patients and their performed tests and to generate reports. The main purpose of this project is to observe cash inflow. Requirements that we have gather are divided in the following categories.

Create, Read, Update and Delete Patient

The user can apply CRUD operations on patient records in user friendly mode.

Create, Read, Update Lab

Lab consist of many tests. The user can Create, Read and update Lab operations on patient records in user friendly mode.

Create, Read, Update Lab

Lab consist of many tests. The user can Create, Read and update Lab operations on patient records in user friendly mode.

Create, Read, Update Test

Apply CRUD operation on Test

Create, Read, Update Doctor

Apply CRUD operation on Doctor

Create, Read, Update Panel

Apply CRUD operation on panel

Create, Read, Update Employee

Apply CRUD operation on Employee

Work flow can be best understand by the following diagram

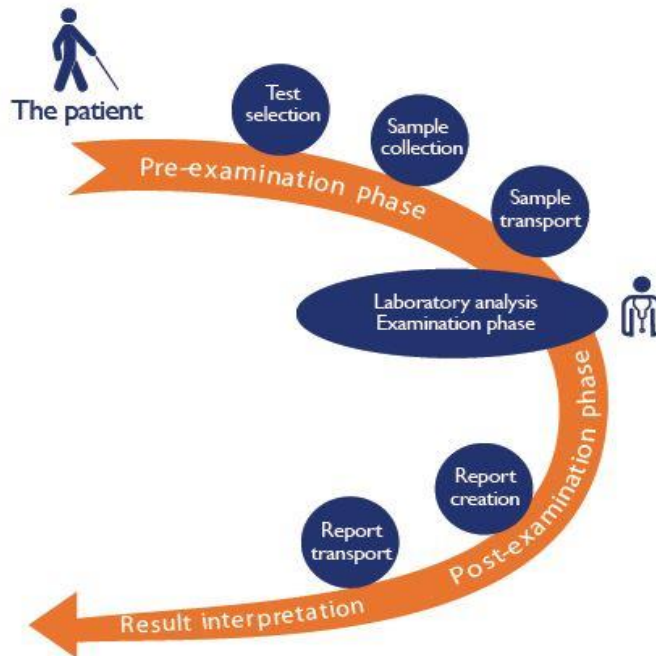


Figure - 5.29: Display of Angular's app

5.2.2 Modeling

The class diagram of the LEXPERT is designed and the developed stereotypes are applied with the required values. The class diagram of LEXPERT is shown in **Figure 5.30**

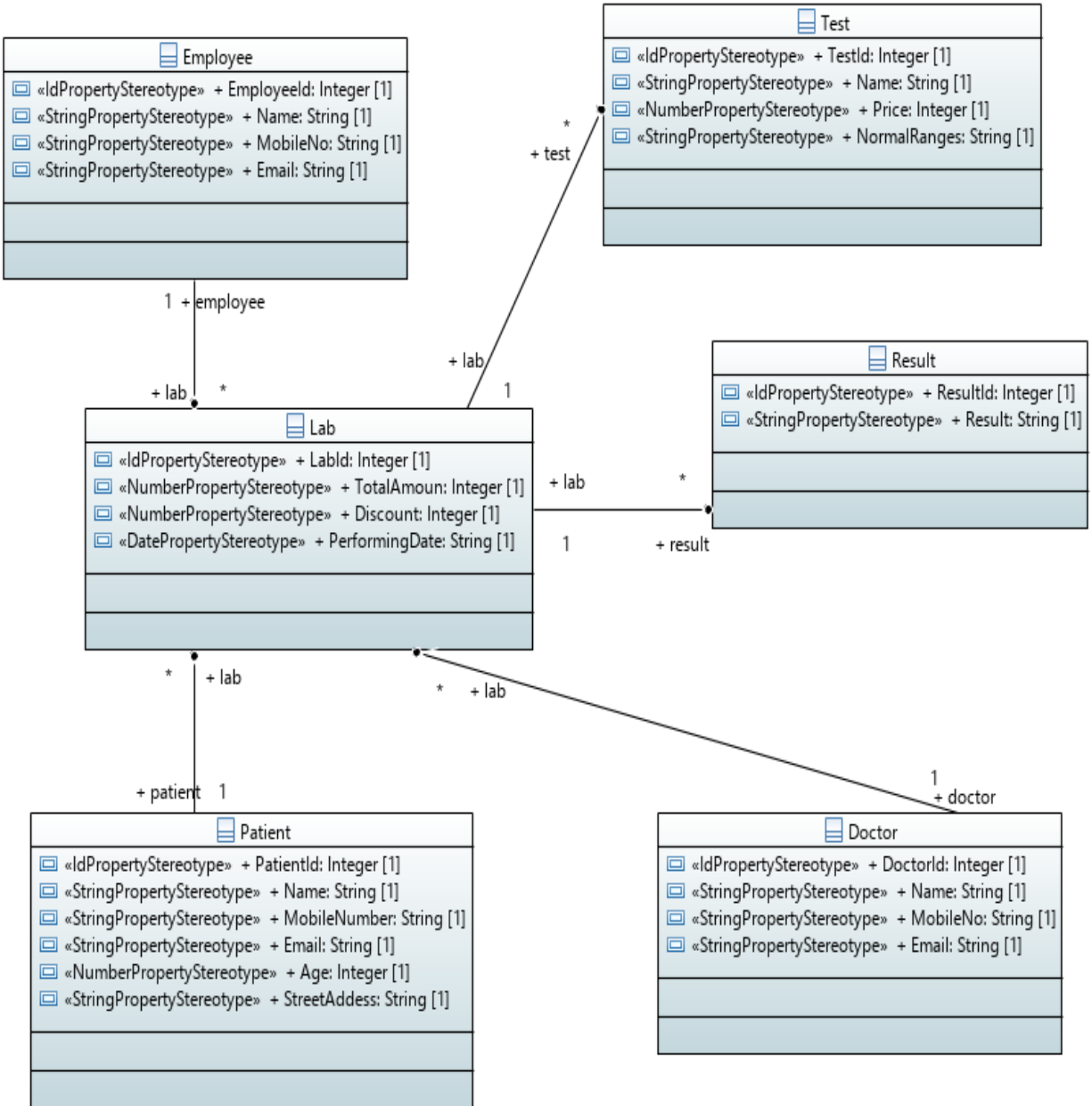


Figure - 5.30: Class diagram of LEXPERT

5.2.3 Code Generation

As the code generated for the Auditing system, in the same way the code is generated for RESTful server (ASP.NET Web API), Web app (Angular) and Hybrid app (Ionic). The code structures for web app (Angular), Hybrid app (Ionic) and Web API are shown in **Figures 5.31, 5.32 and 5.33.**

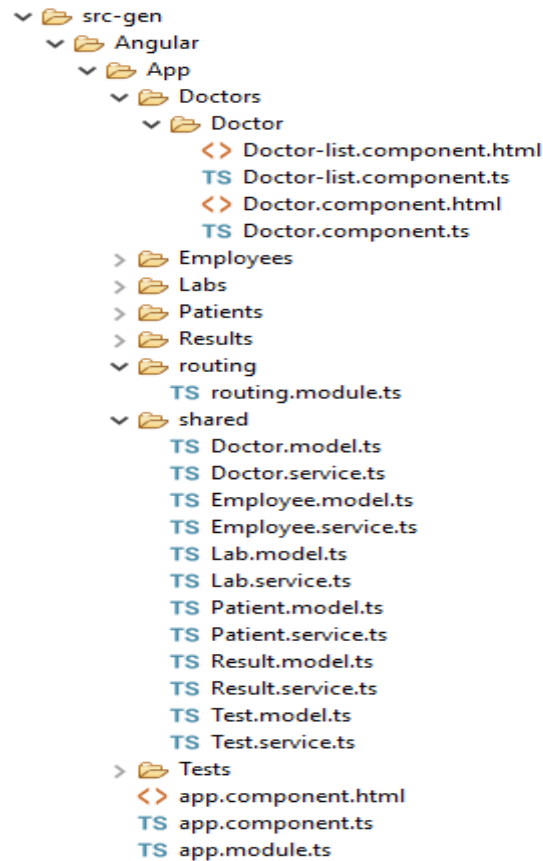


Figure - 5.31: Structure of Generated Code for Web App (Angular)

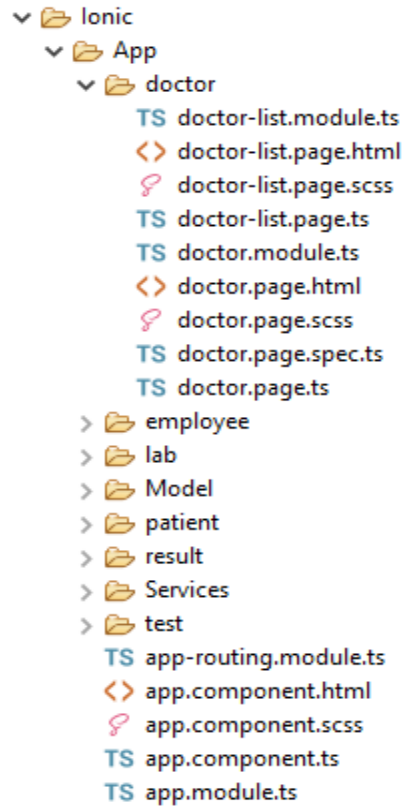


Figure - 5.32: Structure of Generated Code for Hybrid App (Ionic)

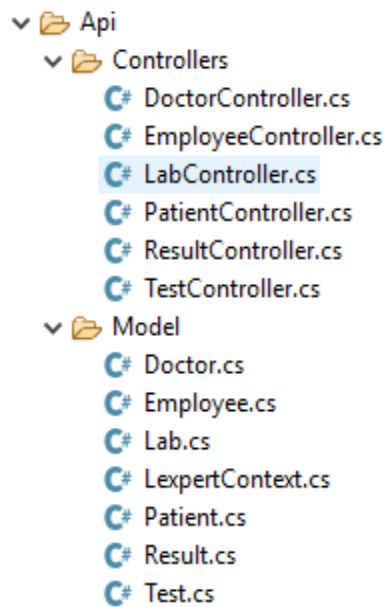


Figure - 5.33: Structure of Generated Code for RESTful Service (ASP.NET Web API)

Chapter 6

Comparative Analysis

CHAPTER 6: COMPARATIVE ANALYSIS

The previous chapter deals with the implementation and validation aspects of the proposed work. The proposed framework presents the modeling of software system into class diagram by applying the defined profile at higher abstraction level. To model the concepts of system Unified Modeling Language (UML) profile is proposed and transformation of model into code has been carried out using transformation engine for implementation purpose. Our proposed approach provided a major contribution in the field of model driven engineering. Our proposed methodology is more efficient than conventional one.

6.1 Comparison

For comparison we selected 12 developers and divide in four group, with three developer in each group. In each group the three developers selected with different skills (ASP.NET, Ionic, Angular). The time for conventional development and the time by our proposed methodology is noted for comparison. After comparison we found that our proposed methodology is five times more efficient than conventional one. This section discusses the comparison of conventional development with our proposed solution.

The Auditing system is selected for validation of our proposed methodology. The 12 developers have developed the auditing system in four groups conventionally and the time for each developer is noted. In each group one developer have skills of developing in .NET, Second developer have skills of developing web app in angular and third developer have skills of developing in mobile app in ionic. After noting the time of every single developer in each group the total time is calculated.

In the second phase the Auditing system is developed by four groups by our proposed model driven methodology. In this methodology the time calculation is different from that of conventional. As the developers had no skills in model driven development, so we have trained the developers on our proposed methodology. The time of training is also considered in calculation of time. The post processing time is also considered in calculation of time. The time of training, time of modeling, and time post processing is added and for each group and total time is calculated. In final the average time of conventional and model driven development is calculated and from that data we found that our proposed methodology is more than five time efficient than

conventional development. The detail of develops' work shown in **Annexures A, B, C and D**. The comparison performed is shown in **Table 5**.

Table 5.1 - Comparison of conventional time and model driven time

No.	Category	Experience	Conventional Method		Model Driven Method				Application		
			Time (hours)	Total	Training	Modeling	Post Processing	Total			
Group 1	1	Developer1	7	1.5	3	3	0.5	21	RESTful Service in ASP.NET Core		
	2	Developer2	9	40					3	3	Web App in Angular
	3	Developer3	4	80					3.5	5	Hybrid Mobile App in Ionic
Group 2	1	Developer1	17	2	3	2.5	0.5	17	RESTful Service in ASP.NET Core		
	2	Developer2	4	36					3	2.5	Web App in Angular
	3	Developer3	6	40					2.5	3	Hybrid Mobile App in Ionic
Group 3	1	Developer1	15	1.5	2	2	0.5	15.5	RESTful Service in ASP.NET Core		
	2	Developer2	3	54					3	3	Web App in Angular
	3	Developer3	8	50					2.5	2.5	Hybrid Mobile App in Ionic
Group 4	1	Developer1	9	1.5	2	2.5	0.5	17.35	RESTful Service in ASP.NET Core		
	2	Developer2	12	55					2.5	4	Web App in Angular
	3	Developer3	15	36					2.5	3.5	Hybrid Mobile App in Ionic
	Average			99.4				17.7			
The Model driven methodology is 5.6 times more efficient than conventional. $99.4/17.7 = 5.6$											

Chapter 7

Discussion along with Limitation

CHAPTER 7: DISCUSSION WITH LIMITATION

This section is divided into two sub sections. A detailed discussion on the proposed research work is described in **Section 7.1**. **Section 7.2** deals with the limitation of the research.

7.1 Discussion

Due to rapid growth in the use of mobile apps, the software owners demanding for the system of same domain to be on both mobile and web apps. The data and presentation layers are involved commonly in more applications. This is the reason that we have proposed and developed a methodology which can save the development time and can generate the scaffolding code for the data layer (RESTful Server) and presentation layer (web app and hybrid app). The profile is designed and the designed profile is to be applied to class diagram from which the scaffolding code is generated in C# for RESTful Server, Ionic for hybrid app and angular for web app. The key advantages of our proposed methodology are are:

- 1) It reduced the time of development.
- 2) It provides high level of abstraction
- 3) This is open source and reusable
- 4) It offers early stage evolutionary prototyping
- 5) It improve the code quality

The models proposed in our solution are at high abstraction level and are easily understandable by various stakeholders. These models neglect the implementation details, thus simplifying the low-level implementation complexity. Furthermore, it reduces time and cost of development. The generated code is deployed into its respective environment for verification.

Our proposed framework has been validated on two case studies: Auditing System and Lexpert (Lab expert which is pathology lab management system). Both case studies require data layer and presentation layer. The first case study is chosen for comparison of conventional development with that of model driven development.

7.2 Limitations

Our proposed solution provides automated code generation for data layer and presentation layer of applications, however, there exist some limitations. Firstly, the profile designed does not cover some scenarios like the selection of code architecture or selection of stylesheets. Secondly, it need to generate the code for react, react native and vue. Furthermore, transformation rules for only Root types and object types are provided as they are used across the most applications.

Chapter 8

Conclusion along with Future Work

CHAPTER 8: CONCLUSION ALONG WITH FUTURE WORK

In this research, we propose model driven scaffolding code generation for web and mobile application to model the requirements at high abstraction level. The proposed profile comprises several stereotypes related to frontend and backend concepts to model the data layer and presentation layer of applications. Providing a consistent user experience across different connection types is a difficult task. Low level implementation of such application types is a complex process. To reduce this implementation complexity, we model the requirements at high level of abstraction. These models generate code through the transformation engine. Verification of the generated code is done by deploying it to their respective implementation environments.

The research work is the first step to make the applications development process simpler and time saving. Particularly, model driven scaffolding code generation has been proposed which adapts the concept of UML Class Diagram, to model the data layer and presentation layer of applications. A complete transformation engine is developed to transform the source models into target low level implementation code. We have covered the transformation of profiled class diagram to RESTful service (C#), web app (angular) and hybrid app (ionic). The transformation engine implementation is carried out in Acceleo through Model to Text approach. We demonstrate the applicability of our proposed tool through two case studies.

In future, we may plan to give the option to choose the some extra feature like the code architecture and stylesheets on UI. Moreover, further frontend enhancement is required to fully provide all the features. We also plan to enrich our base application with other frameworks like vue, react etc.

REFERENCES

- [1] Entity Framework, https://en.wikipedia.org/wiki/Entity_Framework
- [2] Model First, <https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/model-first>
- [3] Code First to a New Database, <https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database>.
- [4] .NET Core, https://en.wikipedia.org/wiki/.NET_Core
- [5] W. Chansuwath, T. Senivongse; A model –driven development of web applications using AngularJS framework; IEEE 15th International Conference on Computer and Information Science (**ACIS**); (2016)
- [6] H.Benouda, M. Azizi, M. Moussau; Automatic code generation within MDA approach for Cross-platform mobile apps; First IEEE International Conference on Embedded & Distributed Systems (**EDiS**) (2017)
- [7] S. Roubi, M. Erramdani, S. Mbarki; A model driven approach to generate graphical user interfaces for Rich Internet Applications using Interaction Flow Modeling Language; 15th IEEE International Conference on Intelligent Systems Design and Applications (**ISDA**); (2015)
- [8] S. Roubi, M. Erramdani, S. Mbarki; Modeling and generating graphical user interface for MVC Rich Internet Application using a model driven approach; International Conference on Information Technology for Organizations Development (**IR4OD**); (2016)
- [9] R. Acerbis, A. Bongio, M. Brambilla, S. Butti; Model–Driven Development of Cross-Platform Mobile Applications with Web Ratio and IFML; 2nd ACM International Conference on Mobile Software Engineering and Systems (**MOBILESoft**); (2015)
- [10] K. Pongpanjanthra, Y. Limpiyakorn; Model-Based Approach to Generating Web Portals; 5th International Conference on IT Convergence and Security (**ICITCS**); (2015);

- [11] J. Luis Herrero Agustin, Model-Driven web applications; IEEE International Science and Information Conference (**SAI**); (2015)
- [12] S. Azreena Mubin, A. Hazri Jantan, A. Kamaruddin, Y. Limpiyakorn; UML Stereotypes for the Development of Process Interaction-Driven Web Applications; ACM Proceedings of the 2nd International Conference in HCI and UX Indonesia (**CHIuXiD**) (2016);
- [13] T. Cerny, E. Song; UML-based enhanced rich form generation; ACM Proceedings of the 2011 Model-Driven web applications; IEEE International Science and Information Conference (**SAI**); (2015)
- [14] N. Vittorazzi de Almeida, S. Louzada, V. E. Silva Souza; A Model-Driven Approach for Code Generation for Web-based information Systems Built with Frameworks; ACM Proceedings Symposium on Multimedia and the web. (**WebMedia**) (2017)
- [15] A. Cortinas, C. Bernaschina, M. R. Luaces, P. Fraternali; Enabling Agile Web Development through In-Browser Code Generation and Evaluation; International Conference on Model and Data Engineering; (2017)
- [16] Z. Morales, C. Magana, J. A. Aguilar, A. Zaldivar-Colado, C. Tripp-Barba, S. Misra, O. Garcia, E. Zurita; A Baseline Domain Specific Language Proposal for Model-Driven Web Engineering Code Generation. Springer International Conference on Computational Science and Its Applications (**ICCSA**); (2016)
- [17] P. Fraternali, S. Comai, A. Bozzon, G. T. Carughi; Engineering rich internet applications with a model-driven approach; ACM Transactions on the Web (**TWEB**); (2010)
- [18] R. Rodriguez-Echeverria, J. C. Preciado, J. Sierra, J. M. Conejero, F. Sahchez-Figueroa; AutoCRUD: Automatic generation of CRUD specifications in interaction flow modelling language; Elsevier (2018)
- [19] S. L. da Costa, V. V. G. Neto, J. L. de Oliveira; A User Interface Stereotype to Build Web Portals; 9th Latin American Web Congress (**LA-Web**); (2014)

Annexure - A
Verification – Group1
A Model-Driven Framework to generate data and presentation layers’
scaffolding code for Multiplatform applications
(Auditing System)

Developer 1

Name: Abdullah Saeed Email: asaeed@thepointofit.com Experience: 7 years
Conventional Coding Time: 1.5 hours

Model Driven Coding Time:

Training: 3 hours Modeling: 3 Hours Post Processing: 0.5 hours Total: 6.5 hours
Participation: Developing RESTful Service in ASP.NET Date: 11th July 2019

Developer 2

Name: Tanzeel ur Rahman Email: trehman@thepointofit.com Experience: 9 years
Conventional Coding Time: 40 hours

Model Driven Coding Time:

Training: 3 hours Modeling: 0 Hours Post Processing: 3 hours Total: 6 hours
Participation: Developing Web App in Angular Date: 20th July 2019

Developer 3

Name: Muhammad Aqib Email: maqib@thepointofit.com Experience: 4 years
Conventional Coding Time: 80 hours

Model Driven Coding Time:

Training: 3.5 hours Modeling: 0 Hours Post Processing: 5 hours Total: 8.5 hours
Participation: Developing Hybrid Mobile App in Ionic Date: 22nd July 2019

Total: Conventional Time: 121.5 hours Model Driven Time: 21 hours

Annexure - B
Verification – Group2
A Model-Driven Framework to generate data and presentation layers’
scaffolding code for Multiplatform applications
(Auditing System)

Developer 1

Name: Muhammad Imran Email: ibodla@lmkr.com Experience: 17 years
Conventional Coding Time: 2 hours

Model Driven Coding Time:

Training: 3 hours Modeling: 2.5 Hours Post Processing: 0.5 hours Total: 6 hours
Participation: Developing RESTful Service in ASP.NET Date: 24th July 2019

Developer 2

Name: Faisal Ishfaq Email: faisalishfaq81z@gmail.com Experience: 5 years
Conventional Coding Time: 36 hours

Model Driven Coding Time:

Training: 3 hours Modeling: 0 Hours Post Processing: 2.5 hours Total: 5.5hours
Participation: Developing Web App in Angular Date: 30th July 2019

Developer 3

Name: Tasaddaq Sultan Email: tassaddaq-sultan @live.com Experience: 6 years
Conventional Coding Time: 40 hours

Model Driven Coding Time:

Training: 2.5 hours Modeling: 0 Hours Post Processing: 3 hours Total: 5.5 hours
Participation: Developing Hybrid Mobile App in Ionic Date: 31th July 2019

Total: Conventional Time: 78 hours Model Driven Time: 17 hours

Annexure - C
Verification – Group3
A Model-Driven Framework to generate data and presentation layers’
scaffolding code for Multiplatform applications
(Auditing System)

Developer 1

Name: Haroon Waheed Email: hwaheed@thepointofit.com Experience: 15 years
Conventional Coding Time: 1.5 hours

Model Driven Coding Time:

Training: 2 hours Modeling: 2 Hours Post Processing: 0.5 hours Total: 4.5 hours
Participation: Developing RESTful Service in ASP.NET Date: 4th August 2019

Developer 2

Name: Sadiqullah Email: sadiq.ullah@teo-intl.com Experience: 3 years
Conventional Coding Time: 54 hours

Model Driven Coding Time:

Training: 3 hours Modeling: 0 Hours Post Processing: 3 hours Total: 6 hours
Participation: Developing Web App in Angular Date: 8th August 2019

Developer 3

Name: Asfandyar Nasim Email: asfandyar@emumba.com Experience: 8 years
Conventional Coding Time: 50 hours

Model Driven Coding Time:

Training: 2.5 hours Modeling: 0 Hours Post Processing: 2.5 hours Total: 5 hours
Participation: Developing Hybrid Mobile App in Ionic Date: 10th August 2019

Total: Conventional Time: 105.5 hours Model Driven Time: 15.5 hours

Annexure - D
Verification

**A Model-Driven Framework to generate data and presentation layers’
scaffolding code for Multiplatform applications
(Auditing System)**

Developer 1

Name: Tanzeel ur Rahman Email: trehman@thepointofit.com Experience: 9 years
Conventional Coding Time: 1.5 hours

Model Driven Coding Time:

Training: 2.5 hours Modeling: 2.5 Hours Post Processing: 0.5 hours Total: 5 hours
Participation: Developing RESTful Service in ASP.NET Date: 11th July 2019

Developer 2

Name: Farhan Khan Email: fsaif@thepointofit.com Experience: 12 years
Conventional Coding Time: 55 hours

Model Driven Coding Time:

Training: 2.5 hours Modeling: 0 Hours Post Processing: 4 hours Total: 6.5hours
Participation: Developing Web App in Angular Date: 29th July 2019

Developer 3

Name: Haroon Waheed Email: hwaheed@thepointofit.com Experience: 15 years
Conventional Coding Time: 36 hours

Model Driven Coding Time:

Training: 2.5 hours Modeling: 0 Hours Post Processing: 3.5 hours Total: 6 hours
Participation: Developing Hybrid Mobile App in Ionic Date: 31th July 2019

Total: Conventional Time: 92.5 hours Model Driven Time: 17.35 hours