<Project Name>

Object Design

<Version>

<Date>

<Your Name>

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc436772639)

[1.1. Object Design Trade-offs 1](#_Toc436772640)

[1.2. Interface Documentation Guidelines 1](#_Toc436772641)

[1.3. Definitions, Acronyms, and Abbreviations 1](#_Toc436772642)

[1.4. References 1](#_Toc436772643)

[2. Packages 1](#_Toc436772644)

[3. Class Interfaces 1](#_Toc436772645)

OBJECT DESIGN DOCUMENT

Object Design Document (ODD) describes object design trade-offs made by developers, guidelines they followed for subsystem interfaces, the decomposition of subsystems into packages and classes, and the class interfaces. The ODD is **used** to exchange interface information among teams and **as a reference during testing**. The audience for the ODD includes system architects (i.e., the developers who participate in the system design), developers who implement each subsystem, and testers.

Among three approaches to generate ODD, we follow “**ODD embedded into source code**” approach in SE301, since the other methods create many redundancies, inconsistencies.

The initial version of the ODD can be written soon after the subsystem decomposition is stable. Both packages and class interfaces can be generated from source code (comments!) by using a tool, which is named Javadoc. Keeping material for the ODD with the source code enables the developers to maintain consistency more easily and rapidly.

# Introduction

Describes the general trade-offs made by developers (e.g., buy vs. build, memory space vs. response time), guidelines and conventions (e.g., naming conventions, boundary cases, exception handling mechanisms), and an overview of the document. Interface documentation guidelines and coding conventions are the single most important factor that can improve communication between developers during object design. These include a list of rules that developers should use when designing and naming interfaces.

**Object Design Trade-Offs**

The Quiz Maker project consists of modules such as exam management, the instructor portal, the student portal, the administration portal, and so on. Therefore, our content management module has to interact with the other modules. Inter-company relationships are an important topic to consider and we have increased our workload to facilitate other modules for the storage and retrieval of data from the database. By providing these interfaces we take control of the database and obey the rules of object-oriented programming. So when you make a modification to a table in the DB, other modules will not get into trouble.

As the user enters data, the module will force the user to follow the predefined pattern designed for this purpose. This will surely prevent a faulty entry of the data into the database. Therefore, the consistency and integration of the data are provided with these patterns. Of course this procedure increases the complexity of our source code.

**Reliability**

We tested all the steps of the software, we developed it with different inputs and tried to improve the experience of the users by fixing all the mistakes. We tested the system after every new feature we added. When we encountered any problem, we figured out where we made a mistake.

**Ex**pandability

While designing our system, we paid attention to extensibility. Thanks to its modular structure, our system is ready for new technologies that can be added later.

**Programmability**

In our project, we tried to create a simple, functional product using the RxSwift 3.2 Swift Framework. The reason we use the Framework is that simple operations such as logging in and signing up are readily available and accordingly we can spend more time on other important parts of the software.

**Compatibility**

Another reason why we use Django in our project(web service) is that we know that it will not hurt us about compatibility. The RxSwift 3.2 framework is currently the most used framework on swift and has a lot of resources. Of course new features can be added to the system later, so we chose to use a software language that is known to survive the problem and make the software long-lived.

**Adaptability**

Our project can easily adapt to the software change. There are no restrictions in our project. It is easily understood and easy to change on the side of the code or on the database side.

**Availability**

Availability requires a system to be functional. Our system must have access to our system 24/7, as the times specified by the user are valid.

**Maintainability**

After the project was completed and started to operate, we gave importance to technical support against the errors that the users would face. We need to explain the system to the users clearly and simply. However, we created the system in a very simple and understandable way**.**

**References**

- There are many quiz maker applications. For example; https://www.quiz-maker.com/ https://www.easypromosapp.com/quiz/. But our system some different from others.

References to existing systems, etc.

# Packages

# a. Models

|  |  |
| --- | --- |
| Quizmaker/Models/Course; | Course.swift |
| Quizmaker/Models/ErrorResponse; | APIErrorResponse.swift  ErrorMessage.swift  GradeErrorResponse.swift |
| Quizmaker/Models/ErrorResponse/Quiz; | QuizCreateErrorResponse.swift |
| Quizmaker/Models/ErrorResponse/User; | ChangePasswordErrorResponse.swift  EditProfileErrorResponse.swift  FieldErrorResponse.swift  RegisterErrorResponse.swift |
| Quizmaker/Models/Question/Enums; | QuestionType.swift |
| Quizmaker/Models/Question; | Answer.swift  ParticipantAnswer.swift  Question.swift |
| Quizmaker/Models/Quiz; | Quiz.swift  QuizParticipant.swift |
| Quizmaker/Models/Section; | QuestionDetailSectionModel.swift  QuizDetailSectionModel.swift  QuizSectionModel.swift |
| Quizmaker/Models/User; | ChangePassword.swift  EditProfile.swift  Participant.swift  SignUp.swift  User.swift |
| Quizmaker/Models/User/Enums; | Gender.swift  UserType.swift |

# b. View Controllers

|  |  |
| --- | --- |
| quizmaker/ViewControllers/Course; | CourseAddStudentViewController.swift  CourseQuizListViewController.swift  CourseRemoveStudentViewController.swift  CourseViewController.swift |
| quizmaker/ViewControllers/Home; | AdminViewController.swift  HomeViewController.swift  InstructorViewController.swift  StudentViewController.swift  UserViewController.swift |
| quizmaker/ViewControllers/Question; | AnswerQuestionsViewController.swift  QuestionCreateViewController.swift |
| quizmaker/ViewControllers/Quiz; | QuizCreateViewController.swift  QuizParticipantsAnswerViewController.swift  QuizUpdateViewController.swift |
| quizmaker/ViewControllers/Quiz/Detail; | QuizDetailViewController.swift |
| quizmaker/ViewControllers/Quiz/JoinedQuiz; | JoinedQuizzesChoiceViewController.swift  JoinedQuizzesViewController.swift |
| quizmaker/ViewControllers/Quiz/Public; | PublicQuizListViewController.swift |
| quizmaker/ViewControllers/User; | ChangePasswordViewController.swift  EditProfileViewController.swift  MyAnswersViewController.swift  MyLecturesViewController.swift  MyQuizListViewController.swift |
| quizmaker/ViewControllers/User/Authentication; | LoginViewController.swift  RegisterViewController.swift |
| quizmaker/ViewControllers; | KeyboardHandler.swift |

# c. View Models

|  |  |
| --- | --- |
| quizmaker/ViewModels; | ParticipantAnswerViewModel.swift |
| quizmaker/ViewModels/Course; | CourseAddStudentViewModel.swift  CourseListViewModel.swift  CourseRemoveStudentViewModel.swift |
| quizmaker/ViewModels/Question; | AnswerQuestionViewModel.swift |
| quizmaker/ViewModels/Quiz; | JoinedQuizListViewModel.swift  PublicQuizListViewModel.swift  QuizCreateViewModel.swift  QuizDetailViewModel.swift  QuizListViewModel.swift  QuizUpdateViewModel.swift |
| quizmaker/ViewModels/User/Authentication; | LoginViewModel.swift  RegisterViewModel.swift |
| quizmaker/ViewModels/User; | ChangePasswordViewModel.swift  EditProfileViewModel.swift  MyAnswerListViewModel.swift  MyLecturesViewModel.swift |

# d. Views

|  |  |
| --- | --- |
| quizmaker/Views/Course; | CourseTableCell.swift |
| quizmaker/Views/Helpers; | Checkbox.swift  IndicatorButton.swift |
| quizmaker/Views/Question; | MultichoiceTableCell.swift  TextTableCell.swift  TruefalseTableCell.swift |
| quizmaker/Views/Quiz; | MyAnswersTableCell.swift  QuizCreateQuestionTableCell.swift  QuizListTableCell.swift  QuizParticipantAnswerTableCell.swift |
| quizmaker/Views/Quiz/Detail; | QuizDetailTableCell.swift |
| quizmaker/Views/Quiz/Detail/Participant; | QuizParticipantDetailTableCell.swift  QuizParticipantTableCell.swift |
| quizmaker/Views/Quiz/Detail/Question; | QuizQuestionDetailTableCell.swift  QuizQuestionTableCell.swift |
| quizmaker/Views/Student; | StudentTableCell.swift |

# Class Interfaces

Describes the classes and their public interfaces. This includes an overview of each class, its dependencies with other classes and packages, its public attributes, operations, and the exceptions they can raise.