Applying Test-driven Development in Evaluating Student Projects

Cuong Huy Tran

Student number: 41217

Master of Science Technology Thesis

Supervisors: Adj. Prof. Dragos Truscan

MSc. Tanwir Ahmad

Faculty of Science and Engineering

Department of Information Technologies

Åbo Akademi University

February 2020

ABSTRACT

Grading software projects submitted by students can become a heavy and time-consuming task, which for many students, can result in delayed feedback provided to them. Additionally, one would like to allow students to evaluate early their projects by themselves before submitting the final version for grading.

This thesis presents a solution that improves the grading process of student projects not only for lecturers but also for students. In our approach, we adopt a test-driven development methodology to provide a clear benchmark of the course project implementation. Our solution allows students to self-evaluate their progress at any moment, while lecturers can use it to automate the grading process. GitHub Classroom is used as a supporting tool to allow students to retrieve and implement their projects from the same initial skeleton project including the tests, and lecturers to retrieve the student projects and evaluate them automatically.

The results show that test-driven development is a viable solution to be applied in an academic environment to improve the grading process. This study also shows that courses in the Information Technology area could use our approach to increase learning and teaching efficiency.

Keywords: web application, unit testing, automated testing, extreme programming, test-driven development, and Django framework.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

TDD	Test-driven development
OS	Operating System
HTML	HyperText Markup Language
CSS	Cascading Style Sheet
JS	JavaScript
HTTP	HyperText Transfer Protocol
SQL	Structured Query Language
API	Application Programming Interface
MVC	Model-View-Controller
DRY	Do not Repeat Yourself
KISS	Keep It Short and Simple
URL	Uniform Resource Locator
ORM	Object Relational Mapping
XP	Extreme Programming
GUI	Graphic User Interface
TFP	Test-First Programming
VCS	Version control system
YAAS	Yet Another Auction Site
UC	Use Case
REST	Representational State Transfer
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
URI	Uniform Resource Identifier
ID	Identifier
AAA	Arrange, Act and Assert
GHz	Gigahertz
RAM	Random-access Memory
CI	Continuous Integration
CD	Continuous Deployment

1. Introduction

In the modern age of software development, technologies have been developed constantly and innovatively. People search for processes that help to program to become more efficient, more productive, and with a minimum number of errors. Test-driven development (TDD) [1] emerges as a solution to achieve the aforementioned goals. The main principle of TDD is simple: tests are written first, then implement the source code until all the tests pass.

The Development of Server-side Web Services is one of the courses given at Åbo Akademi University, in which students must implement from scratch a relatively large size project (about 1200-1700 lines of code) in Django [2], a Python framework for web applications. After students have submitted their projects for grading, lecturers grade the latter by individually checking each project, including activities such as inspecting the code and executing the project. On average, 40 to 70 students attend the course each year and each project can take 20-30 minutes to grade. Based on the number of students, it can potentially take several weeks for students to receive their final grades. Besides that, students would like to be able to evaluate the status of the project and the potential grade throughout the implementation of the project.

Our proposed solution is to build an initial skeleton project containing preliminary files and our acceptance test suite. We use test-driven development (TDD) to write acceptance tests for the project. Each test case corresponds to one requirement in the project specification. Since the project implementation is empty, these tests will fail in the beginning. In order to pass the tests, students are required to implement corresponding specifications. By continuously evaluating their implementation against the tests, students can track their current points while implementing. Additionally, lecturers can use our proposed solution to grade students' projects and ease the grading effort.

The project follows common Python and Django coding standards and it is appropriately documented, which makes it easy to read and navigate. The test coverage is expected to be reasonably high because it needs to cover as many features as possible. The skeleton project should avoid encoding as much design as possible that could limit the freedom in the implementation. A tutorial is also required to state how to set up the project, run the tests, and how the grade is automatically calculated.

To summarize, the thesis will provide a new approach to improve teaching and learning experiences, a tutorial that helps the students and the lecturers use the approach, and the necessary artifacts needed by students to implement and evaluate their projects.

Automatic grading of assignments is not a novel topic, many researchers have already addressed this topic in the past with similar approaches. Pilla [3] utilized GitHub and Travis CI, a continuous integration (CI) service that integrates with GitHub, to build an automatic testing environment for students. Although the work was conducted on some simple C-code assignments, the preliminary results showed great potential. Comparably, Cai and Tsai [4] applied a similar solution to an Android application development course with improved security.

However, neither of them used a starter repository in their solution. Our approach is also different from theirs because we follow TDD to create a starter repository. Students can download the repository and start working immediately. We do not use any continuous integration (CI) service; instead, we have implemented our approach to automatically download student projects, grade them and generate a detailed course-level report. From our experience, a continuous integration does not provide a global view of all students' repositories, and it requires students to commit code frequently to be relevant. With our approach, we can retrieve student projects any time we want and have all the information of those projects in the report. The approach also allows teachers to update the starter repository and even student repositories.

This thesis is divided into six chapters. The first one describes the motivation and solution to the main problem briefly. Chapter 2 is about software development processes such as Extreme Programming and Test-First Programming. They are essential parts of the Test-Driven Development that is discussed later. After that, we delve deeper into web applications, the concepts of the Django framework, and related technologies. In Chapter 3 and Chapter 4, we will have a closer look at the target of the thesis and the proposed solution, respectively. Finally, the evaluation and conclusion of the thesis are discussed in Chapter 5 and Chapter 6, respectively.

2. Background

This chapter will describe the theoretical background and practical concepts needed for the thesis' work. Firstly, Extreme Programming and Test-First Programming will be discussed thoroughly as a foundation for the Test-Driven Development methodology. Secondly, software version control systems will be explained and how they work. After that, the definition of web applications is introduced, along with their principles. Because the Django framework was used in this thesis' work, its main features will also be introduced.

2.1. Software development processes

Throughout the years, several methodologies for software development have been proposed. In the following, we are going to review those that are relevant to this work.

2.1.1. Extreme Programming

Extreme Programming (XP) [5] is a methodology for software development that emphasizes programming techniques, clear communication, and teamwork. All the members of the process, such as managers, developers, and clients, need to collaborate on an equal basis to achieve a simple yet productive working environment.

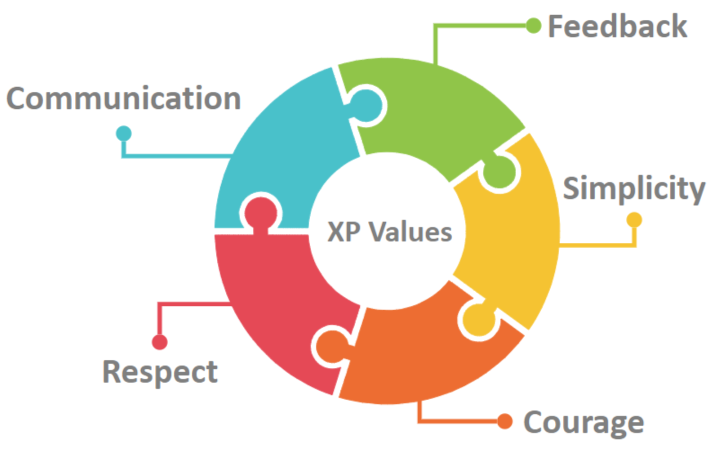


Figure 1. Five values of Extreme Programming.

Extreme Programming improves software development in five essential values: communication, feedback, simplicity, courage, and respect [6], as shown in Figure 1. The teams who practice XP communicate with each other by oral communication, source code, or system structure to solve problems as efficiently as possible. For this reason, they receive concrete and continuous feedback by testing their projects from the beginning. Programmers write their own tests, and the tests are verified again by testers. This approach allows the system to progress smoothly. Tests also keep the system design and development cycle well-structured and straightforward. In the beginning, the teams plan and break the software development into short phases in which there will be a deployable product at the end of each phase, also known as minimal viable product (MVP), therefore, they could deliver frequently the software to clients as frequently as possible and make changes if suggested.

XP encourages creativity in planning, design, and implementation of functionality to adapt to continually changing requirements and technology. However, human factors play a central role in software development and they are the key to accomplishing the above values. XP promotes team members and customers to build a close collaboration. XP can work with teams of any size; with every small success, it grows a relationship between the members of a team and strengthen the respect based on the unique contribution of each individual [6].

An XP software development process typically has several development phase iterations. Each iteration consists of five stages: planning, managing, designing, coding, and testing, as displayed in Figure 2.

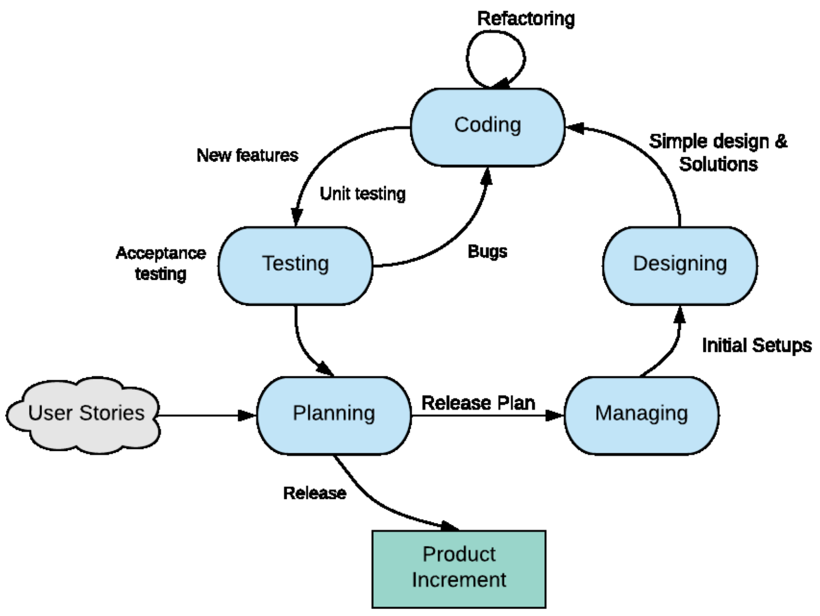


Figure 2. Extreme Programming Iteration Overview.

Planning

At the beginning of the stage, customers make a list of *user stories* that define the functional features along with the priority and difficulty level of each of those features [7]. Then, the team analyzes the user stories to create a *release plan* for the current iteration. The release plan contains a schedule, a total amount of work for this iteration, and the plan for the future. Due to the nature of XP, a development iteration is relatively short. Therefore, the release plan should have enough tasks so that they can be completed within the duration of the iteration.

Managing

At this stage, the project manager will create some necessary *initial setups* to establish the team according to XP rules. The workspace needs to be open and pleasant to encourage discussions among members. The team also agrees on a sustainable pace that is ideal for both implementation and maintenance. Even if XP does not work entirely for some reason, its rules can be changed midway due to its flexibility [7].

Designing

At this stage, the team should create a fitting system design that the whole team will follow in this development phase. The design should be simple, but easy to explain to new people and allow them to contribute immediately. Designing also helps developers understand the system, and find ideas or solutions for improvements, or address potential future problems. The outcome of this step is *designs and solutions* based on the release plan of the current iteration.

Coding

XP practices collective code ownership: Everyone reviews code and any developer can add functionality, refactor or fix bugs [7]. The team should pick an appropriate coding convention that all developers must follow. New features and their tests will be implemented at this stage. This stage can be repeated to ensure the new code works correctly and robustly. This activity is usually called *refactoring code*. After thoroughly reviewing the new code, the team integrates and commits it into the teams' software version control routinely.

Testing

As an essential rule, all code or functions must have unit tests with good code coverage. Before the project can be released, all unit tests must pass and all critical *bugs* in that iteration must be fixed. *Acceptance tests* are also run frequently to ensure the project satisfies the requirements. The Coding and Testing phases take usually most of the time in an iteration because the team goes back and forth between them to develop and fix bugs. After this stage, there is a deliverable product that is ready to be deployed, and the whole process will be repeated.

2.1.2. Test-First Programming

Test-First Programming (TFP) is a concept of Extreme Programming. It is rediscovered and documented by Kent Beck in [6]. Its core idea is very straightforward: *Write the tests before writing the code*. As explained in [8], there are seven key steps to apply TFP:

1. Think about what needs to be done.
2. Think about what it looks like from the customer perspective.
3. Plan how to test it.
4. Write simple test cases.
5. Write the production code for those tests.
6. Expand the tests to cover the complete behavior.
7. Write the production code to pass the tests.

It is unquestionable that for a large size and complicated software, thorough tests are necessary to be in place so that the software can be implemented correctly. Unit tests are not a magic wand that can make an application bug-free, because it is impossible. However, they are still relevant to increase confidence in the quality of the software. The ideal would be that every developer writes and performs tests for his or her code. Nevertheless, the tests could also be written after the code, and it would still be acceptable, so why then should one practice Test-First? There are many problems that Test-First Programming addresses in [8]:

- It prevents developers from being distracted and falling into the feature creep. The tests help as a guide that keeps the developers focus on the planned scope.

- If it is difficult to write a test, there might be a design problem. Writing tests help understand the specification.
- By writing clean and testable code, a developer can earn the trust and respect, which is a core value of Extreme Programming.

Besides the Test-First approach, there is another common development method, in which unit tests are written after the production code, thus, it is called *Test-Last* development. This approach, even if very popular today, has some limitations [9] that affect software development because the tests written this way are originated from the source code. Even if the tests pass, there could be some hidden errors, which create a false sense of security. In the worst case, tests are not even implemented or postponed to a later date as the code is already working.

2.1.3. Test-Driven Development

Test-Driven Development (TDD) is a process, which is similar to Test-First Programming that requires tests to be written before the implementation code. However, it has some differences that we will discuss later. The TDD process is a cycle and it will be repeated over and over until all the tests pass [1]. A standard TDD cycle is presented in Figure 3.

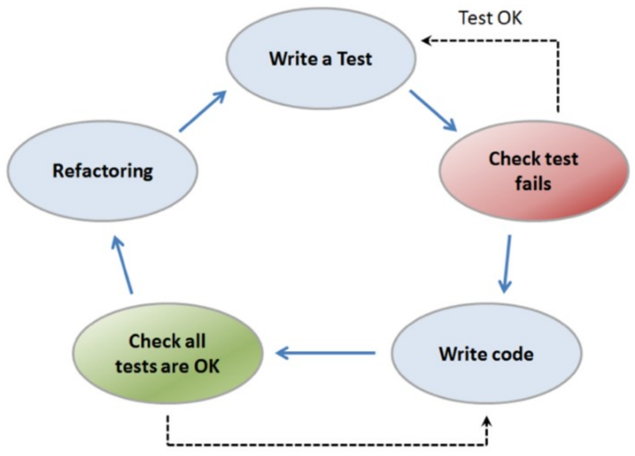


Figure 3. Test-driven development cycle [10].

1. **Write a test.** Every new feature begins with writing a test. The test should be clean and simple. By doing this before writing the code, it motivates the developers to think first about the requirements, the design of the system and the way it should work [11].

2. **Run and check if the test fails.** The test is expected to fail since the application code does not exist yet. This step emphasizes the target feature for the developers. If the test passes, it must be re-written to fail.
3. **Write code.** Write just enough production code to fulfill the test. Programmers need to be careful not to implement further than the functionality of the test.
4. **Run all tests.** If all tests pass, it means the new code does not break any existing features and the new test is satisfied. If they fail, the new code has to be modified until all tests pass.
5. **Refactor code.** In this step, any possible code duplications are removed. The purpose of this step is to preserve the readability and maintainability of both existing and new features. The test cases are re-run frequently to ensure the refactoring code does not alter unrelated features [11].

TDD can be applied to many software development models, but it will work better if a model is software testing-oriented. For instance, there is a model named V-model, in which testing is done at each phase. The V-model stands for the Verification and Validation model [12]. The phases of the model are depicted in Figure 4 below.

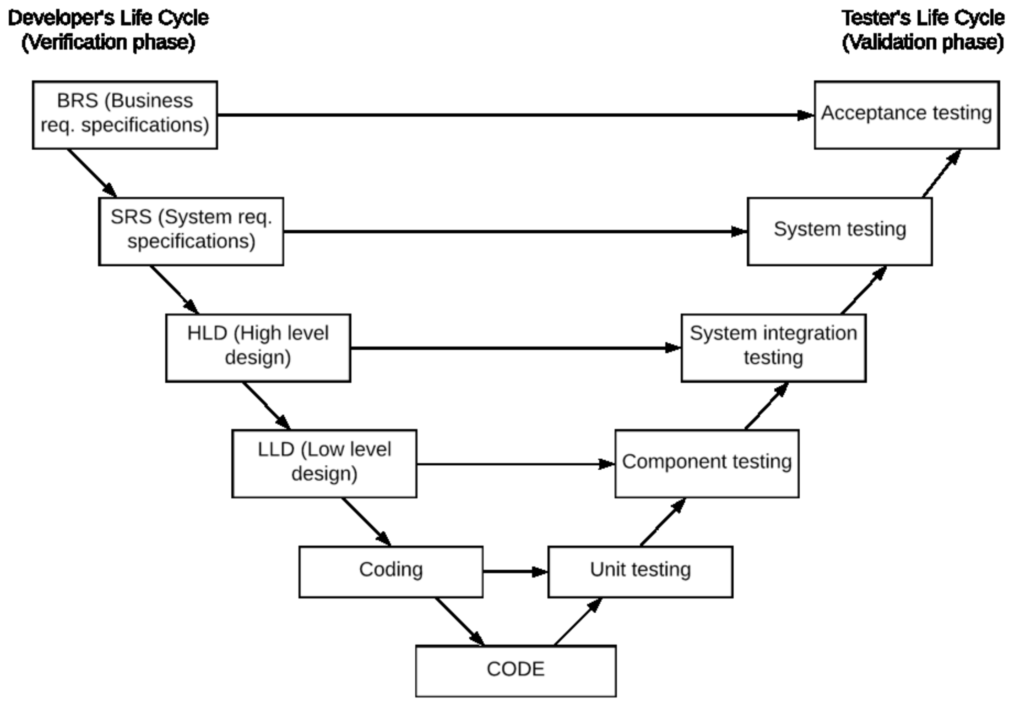


Figure 4. V-model Diagram [12].

Typically, there are five stages in a V-model. In the business requirements analysis phase, acceptance tests are created to check later on if the software meets the user's needs. Next, a general test plan is created according to the system specification. After that, on the high-level design phase, integration tests are performed to test the connections between the system, modules, and platforms. The next phase is low-level design, where the software logic and components are designed. Thus, component tests are created in this phase. And the last phase is the coding or implementation phase, where all the implementation takes place. All methods and functions should have unit tests associated. Once this phase is reached and completed, the path of execution moves up to use the earlier test plans.

Unit testing is the most basic and lowest. It focuses on the smallest units of an application, which in object-oriented-programming they are methods or functions and can be parent classes or children classes. To assist unit testing, frameworks, stubs, testing databases, mock objects are applied. Unittest code should be compact and quick to execute [13].

A standard test method features three steps. Firstly, it sets up a testing environment, then it invokes a function that needs to be tested, and finally, it evaluates the result. If the result is as expected, the unit test passes, otherwise, it fails, signaling that there are some defects in that function. These unit testing phases can be called AAA, which stands for Arrange, Act and Assert [14]. Every developer is responsible for writing and performing unit tests for their code.

The main benefit of TDD is that when writing new code, the test cases can act as a guideline, so the developers can conveniently follow resting assured that they are on track, and no feature's specification is missing [15]. Additionally, by running tests throughout the development, feedback is given regularly, and no code left unattended.

In addition, developers spend less time on debugging and fixing errors. Although TDD is not a miracle solution to eliminate all bugs, more tests mean better code coverage, and that will reduce the cost of maintenance and a large number of bugs [15]. Combined with a version control system, when a test fails, TDD helps to identify the error quickly and more productively.

TDD can also lead to more clean, modularized, and extensible code because of the constant refactoring. The code is tidier, well documented, which allows other team members to understand it. This makes the program is appealing for future enhancement or expansion.

TDD also has some limitations. If the project specifications and requirements are not studied and analyzed well enough, passing tests could cause a false sense of safety. Due to the nature of TDD, it has a long learning curve. Additionally, writing and maintaining an overwhelming number of tests costs time and resources, particularly for small teams. It takes approximately as much as 16% more development time than that of traditional Test-Last [16].

Compared to Test-First Programming, the tests in Test-Driven Development control the design of the program. TDD programmers already think broader about the design of the application before writing tests instead of directly implementing features. More importantly, after the code is written to fulfill the tests, TDD further requires refactoring the code to improve the performance, not only trying to meet the requirements as in TFP [17].

2.2. Software version control system

Version control system (VCS) is a tool that helps a developer or a team of developers to manage changes to source code over time so that they can recall them later if needed. VCS keeps track of every modification from add or edit to move or delete in a special kind of database. The types of file VCS can track are not only source code, but also images, audio files, movies, or any other type of digital asset.

Most software version control systems have some general operations such as adding new files, modifying current files, and committing changes. They also allow developers to compare differences between versions, revert some files or the entire project to a previous stable state, see who modified what, or who caused a problem, and many more. For almost all software projects, the source code is the most critical central part, and the teams are responsible for protecting it. A VCS, which is updated frequently during the development, can act as a backup storage. If some files are lost due to accidents or human error, the team can quickly recover them from VCS [18]. Currently, there are two popular types of version control systems: *centralized* and *distributed*.

Centralized version control systems (Figure 5) store everything in one shared server, including all the versioned files [18]. If a developer wants to work on a file, he or she needs to retrieve and check out that file from the shared server. It ensures that every developer always receives the latest version. Managers also have full control over who can do what and the status of a project. Some of the popular systems are Perforce, Team Foundation Server and Subversion. However, an issue occurs when two developers try to edit the same file at the same time. Centralized VCS solves this conflict by one of the following methods: file locking and version merging.

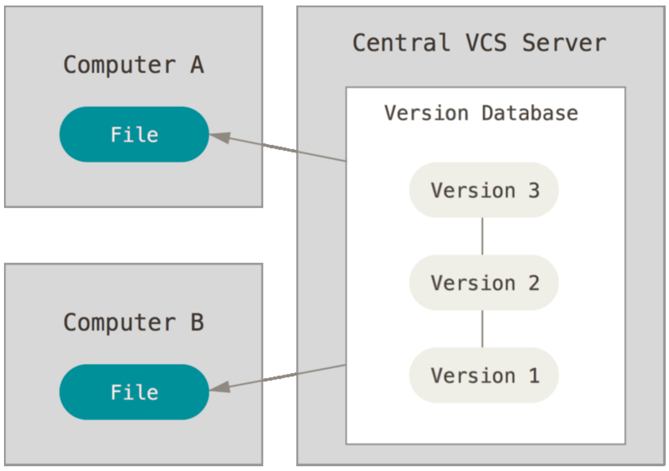


Figure 5. Centralized version control system [18].

File locking is a method of preventing concurrent access to a file with the aim that only one developer can change it at any time (called *check out*). Version merging is to allow multiple developers to edit the same file concurrently, but when they commit their code, they must merge the changes into the central repository. With the limitation in concurrent editing, centralized VCSs are not convenient to work with in large teams because file locking slows down the working pace, and the effort to merge versions is high. Additionally, if the shared server has any serious problem, it would result in an access suspension for the whole team or even a data loss.

In distributed version control systems (Figure 6), developers completely mirror the project or repository, including full version history, rather than checking out some of the files [18]. Then they are able to make changes locally and submit later to the centralized location. In this way, it makes common operations (such as commits, reverting changes, merges) faster since they do not need to communicate with a central server. For this reason, any of the local projects can effectively function as a full backup of the source code if the server that contains VCS cannot be accessed.

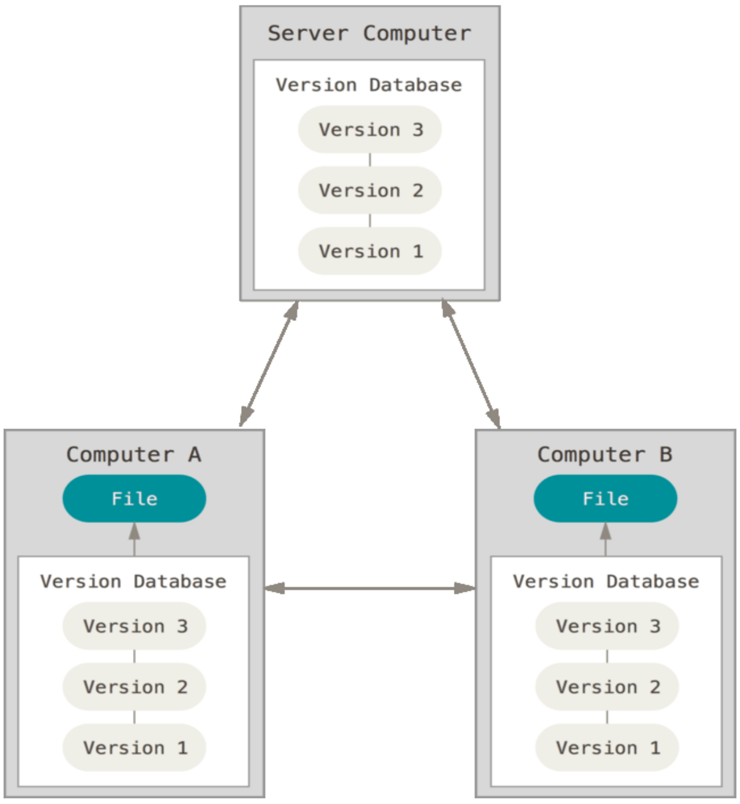


Figure 6. Distributed version control system [18].

Furthermore, the Distributed VCSs (such as Git, Mercurial, or Bazaar) enable the possibility to work on different workflows simultaneously within the same project. For example, a subgroup of a team can download the latest code and branch out to make some other features. Later, they can merge that branch to the main flow, or they can abandon it without affecting the main source [18]. However, the Distributed VCSs still have some drawbacks. Because developers are allowed to retrieve the online code base, it could be potentially a security issue. Another disadvantage of Distributed VCSs is storing binary files. If a project contains any significant number of binary files such as images or videos, it will increase the repository size and history considerably, hence downloading the project can take more time and space. Meanwhile, in centralized VCSs, it is not necessary for developers to save the entire project in their local computers so there is no requirement for additional space.

2.2.1. GitHub and GitHub Classroom

Github.com is one of the most popular open-source version control systems. It provides services for both public and private repositories. GitHub has many factors that are relevant for the goal of this thesis:

- GitHub is a distributed version control system. It provides many functionalities: repository, issue tracking, wiki, etc.
- Being open-source means there are many third-party applications that could be useful for the implementation.
- It is free to use by both teachers and students, for educational purposes.

In addition to the above features, GitHub Classroom, a GitHub's initiative, allows teachers to create assignments, track student progress, and integrate with useful third-party tools. It also works well for courses with a large number of students. With the help of GitHub Classroom, teachers spend less time on setting up the technical platform and spend more time interacting with students.

2.3. Web Applications

A Web Application is a computer program that provides dynamically created content to be displayed in a Web browser [19]. A web application is usually hosted on a server with a domain; for example, a car sales application could be located at *http://carsales.com*. Web applications can perform various tasks from simple to complex. For instance, a web application can include a simple user registration form, or it can be a massive e-commerce page or a multi-player online game.

In general, people tend to misuse web application and website terms or use them interchangeably because the boundary between them is uncertain with the advance of technologies. The definition from [19] not only summarizes what is a web application but also distinguishes it from websites:

> "What is a 'Web application?' By definition, it is something more than just a 'Web site.' It is a *client/server* application that uses a Web browser as its client program, and performs an interactive service by connecting with servers over the Internet (or Intranet). A Web site simply delivers content from static files. A Web application presents dynamically tailored content based on request parameters, tracked user behaviors, and security considerations."

2.3.1. Benefits of web applications

Web applications have various benefits which can ease software development:

- Anyone can use web applications regardless of the operating system or power of their computer, as long as their browser is compatible, and they have an internet connection.
- Since all users have the same version of the web application, the process of updating the application is quicker and more comfortable.
- Unlike traditional computer programs, it is not necessary to install a web application, so there is no space limitation.
- Web applications require fewer resources to maintain and also require lower computer configuration from users.

2.3.2. Structure of a web application

Generally, web applications are separated into levels called *tiers* [20], where each tier has a distinct role. Throughout the history of software development, there are many versions of this kind of structure, but for the scope of this thesis, we will focus on the *three-tier architecture*. The three tiers are called *presentation* tier, *business logic* tier, and *data access* (or *data management*) tier. This architecture is quite straightforward and easy to implement. Its structure breaks the application into three different parts with different roles, thus effectively improving the clarity while keeping the needed cohesion between them. Figure 7 below illustrates an overview of the architecture.

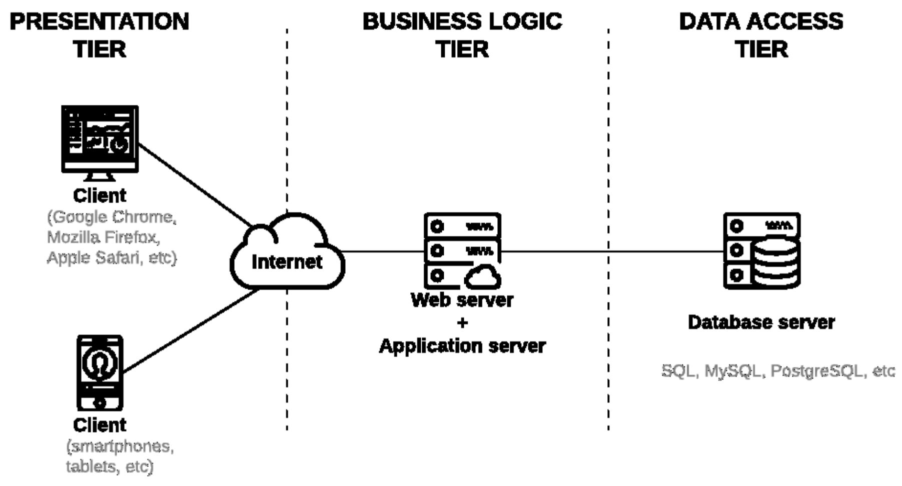

PRESENTATION TIER
BUSINESS LOGIC TIER
DATA ACCESS TIER
Client
(Google Chrome, Mozilla Firefox, Apple Safari, etc)
Internet
Web server + Application server
Database server
SQL, MySQL, PostgreSQL, etc
Client
(smartphones, tablets, etc)

Figure 7. Three-tier Architecture [20].

Presentation tier

This tier is the outer level of an application. The tier displays all the intelligible information to users such as sign-in form, text editor, or shopping cart. It is the place where end-users can interact directly with the application via browsers to send requests. The presentation tier is mostly designed with HTML [21], CSS [21], and JavaScript [22]. It transfers requests to the business logic tier and also receives responses from that tier.

Business Logic tier

This is a middle-tier that is responsible for all application logic and functions. It handles requests that come from the presentation tier and the data tier. This tier should not be involved in rendering any pages or accessing databases, however, it can generate HTML pages dynamically to be rendered by the presentation tier. This tier also contains unit tests and integration tests.

Data Access tier

The data access tier stores and retrieves information from a database or file system. The other two tiers are not allowed to access the database directly but must go through the data access tier. This tier hides the database and from other tiers because it provides an API to the Business logic tier, and then eventually back to the presentation tier.

2.3.3. Client-Server model

As mentioned in the definition, web applications function based on the interactions between the application and the users or, in other words, the server and the client. The Client-Server Architecture is a computer network in which many clients request and receive service from a server. The architecture separates a web application or software into two layers, front-end and back-end. The front-end represents the presentation layer, and the back-end is for the data access layer of a web application. The business logic connects the two layers by handling requests and data between them. Both front-end and back-end require developers to have different knowledge to be able to design and implement. For the scope of this thesis, only some selected technologies will be elaborated to explain what are their roles and how are they applied to this thesis' work.

2.3.3.1. Front-end

Front-end is the side of a web application that runs on the client (i.e., the browser). It is what users can see and interact. HTML, CSS, and JavaScript synergize with each other, powered by the client's browser, create almost everything on the web. Furthermore, the term *front-end developers* nowadays not only refers to those who can design with Photoshop or Fireworks, but also developers who can code using HTML, CSS, and JavaScript, or even some UI frameworks such as jQuery, AngularJS, ReactJS, etc [23].

Hypertext Markup Language (HTML) [21] is a markup language that is used to define the structure and meaning of web content. HTML consists of a series of elements that web browsers retrieve from a server, which are rendered and displayed in browsers. However, HTML mostly focuses on the content of web pages, so it needs other technologies such as CSS for a better design or JavaScript for providing interactive behaviors to users.

Cascading Style Sheets (CSS) [21] is a style sheet language used for describing how HTML elements are decorated on the web page. CSS controls many aspects of the presentation of a web, including fonts, colors, page layout, and design, to make it more appealing to users. A combination of HTML and CSS allows web pages to be presented responsively on different types of devices, such as computers, smartphones, and tablets.

JavaScript [22] is originally an open-source, client-side programming language. JavaScript can update the content of HTML and CSS. Scripts are written and executed as plain text, and it does not need a compiler to function fully. Today, it has become one of the most used programming languages and a widely-supported standard. JavaScript has been developed to the point it can even handle server tasks, but in this project, we only discuss the front-end part of it.

Modern JavaScript can handle activities related to user interactions and web page manipulation. As stated in [22], JavaScript allows to:

- Dynamically insert new content to a page or modify the style of existing content.
- React to events triggered by user actions such as mouse clicks, keypresses, etc.

- Send requests to remote servers to download or upload files.
- Use a large number of libraries, created by worldwide developers, to enhance the experience and expand its capabilities.

2.3.3.2. Back-end

A front-end side alone is not enough for a web application to run; it needs a server, or a back-end side, to work properly. The back-end typically has an application and a database that is stored on one or many servers. After users interact with the front-end, if there is any information that needs to be changed, the application will do it on the server. The back-end side is like the backbone of a project; it handles the information sent from the front-end side and stores it in a database.

Technologies required to develop back-end systems are reasonably different from front-end counterparts. There are PHP, Python, Java, C#, and many others as programming languages, or MSSQL, PostgreSQL, MongoDB as databases. To make them even more convenient to use, they are generally enhanced by many frameworks, for instance, Laravel, Django, Hibernate, and ASP.NET. Within the scope of this thesis, Python with the Django framework will be used.

2.3.4. HTTP – HyperText Transfer Protocol

The HyperText Transfer Protocol (HTTP) is a stateless request/response protocol for transporting messages over the network [24]. It is described as a core of communication between clients and servers. A typical HTTP request circle operates like this: a client submits a request to a server, then the server processes the request and returns a response to the client. HTTP is a fundamental aspect of the World Wide Web (WWW) architecture, and it supports not only hypertext documents but also images and videos.

Servers process each request or response independently without acknowledging its purpose [24]. Today, most of the web applications are using an extension of HTTP, called HyperText Transfer Protocol Secure (HTTPS), which is fundamentally the same but provides more secure communication over a computer network.

2.3.5. MVC Architecture

Model-View-Controller (MVC) is a design pattern that separates an application into three components – Model, View and Controller (Figure 8). Each of these has specific roles and abstract boundaries between them, which they use for communication [25].

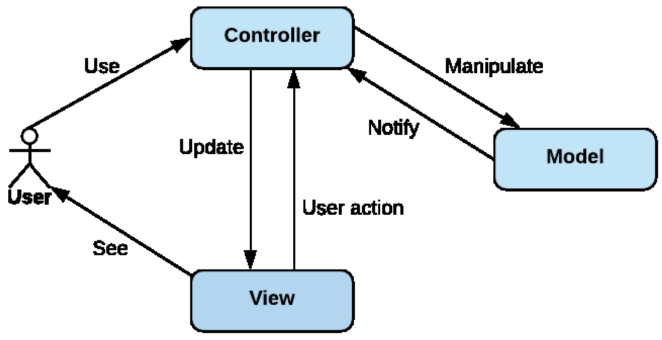


Figure 8. MVC design pattern.

- Model – it represents the shape, state, or structure of data. It does not contain business logic and does not depend on the controller or the view.
- View – it presents data that the model contains to the user and enables them to modify the data.
- Controller – it handles the user's requests and acts as an intermediary between the Model and the View components. It manages the data-flow from both ends and updates whenever data changes.

MVC is used nowadays as a UI-related pattern for web applications. It is usually integrated with other architectures for building a complete web application. It enhances code reusability and removes unnecessary complexity because the three components are separated with a minimum number of dependencies. The separation of components in the framework makes future development and modification easier to implement. To that extent, MVC is ideal for developing complex applications because multiple developers can work simultaneously on each component. Since components in the framework are loosely coupled, testing of different application parts is more comfortable. In fact, testing is a major design goal of MVC pattern.

Beside the advantages, MVC framework has some drawbacks. MVC can result in a system architecture from a moderate level to a very complicated one, so it might reduce the efficiency of the development, especially on small applications. Another disadvantage is when implementing a new feature, the code needs to be updated on three components. Thus, developers are expected to have knowledge of various technologies to work on both the back-end side and the front-end side.

2.3.6. Django framework

Django is a high-level and open-source web framework written in Python [2]. It is a package of several built-in libraries that are written in pure Python. These Django libraries are invisible when viewing a project, but they can be enabled easily in the settings of the project. Django is created based on MVC pattern but with some noticeable tweaks. It is referred to as a Model-Template-View (MTV) or Model-View-Template (MVT) architecture. Similar to MVC, the Model functionality is used to handle and access to data and databases. However, in Django, the View describes which data is presented to the user, not how the data looks. And finally, the presentation of data is shown in the Template.

The Django framework uses a file that contains URLs and mapping routes to functions, *urls.py*. When a client makes an HTTP request, the Django framework will use this file to navigate to the needed function in View, run that function with the data from Model, and transfer response to Template to show it to the user.

Django is known for its fast, secure, and mature aspects [2]. The framework follows the DRY (Do not Repeat Yourself) and KISS (Keep It Short and Simple) principles, which emphasize on reusability, code efficiency, and low coupling with many built-in components that assist the development. In addition, it can support the implementation of different types of web applications from wikis to client relationship management systems. The community that is using and supporting Django is robust and helpful, which means there are a lot of third-party packages and add-on support development.

The parts that make Django a popular framework to work with for developers are its open nature and on-going evolution. It has been in the industry for more than a decade and still keeps growing day by day, many new features have been added continuously in, and lots of bugs and errors have been patched. The following are some main features which are described in [2]:

- **Rapid development**. The lightweight and standalone attributes highly improve implementation and testing without affecting the quality**.**
- **Built-in unit test framework**. Django inherits Python's *unittest* library and enhances the whole process of writing and running tests.
- **High scalability**. Its modularity is suitable for small projects to complex applications.
- **Security**. Django provides many tools to effectively protect against cyber-attacks such as CSRF attacks, XSS attacks, SQL injections, etc. Moreover, it is updated regularly to keep up with the latest trends in website security.

Django has a default and recommended structure of how files and folders organized in a project. As displayed in Figure 9, a Django project consists of one or more children applications. A Django application is a self-contained package that is designated to handle one specific job, for instance, a user profile, a newsfeed, or a blog.

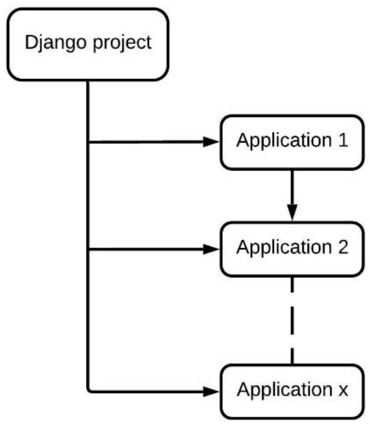
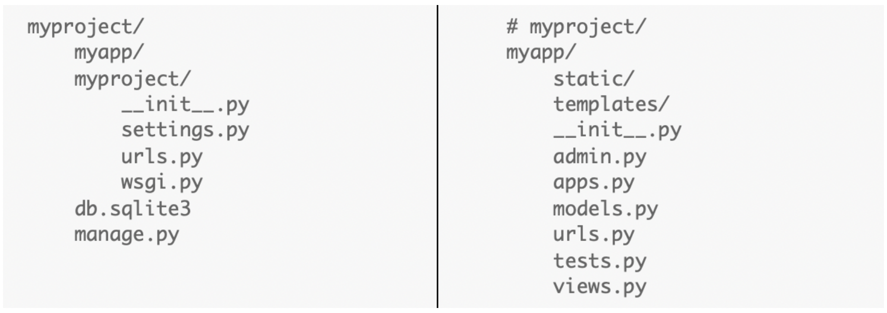


Figure 9. Overview of Django project structure.

A Django project can be created by running a specific Python command; it will automatically create a folder structure containing the files needed for the project. For visual and detailed purpose, let say *myproject* is a Django project created by the command, and some files come with it after its creation (described in the left part of Figure 10).



```
myproject/
    myapp/
    myproject/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    db.sqlite3
    manage.py
```

```
# myproject/
myapp/
    static/
    templates/
    __init__.py
    admin.py
    apps.py
    models.py
    urls.py
    tests.py
    views.py
```

Figure 10. Detailed view of a Django project structure.

The description of each file is presented below:

- The outer **myproject/** directory: a container of this project. Its name is not important and can be renamed for convenience.
- The inner **myproject/** directory: this is the actual Django web application. This directory is the top-most-level application that is created automatically.
- **__init__.py** file: an empty file that informs Django this directory is a Python package.
- **settings.py** file: settings and configuration for this project. Every Django project must have a setting file.
- **urls.py** file contains project-level URL declarations [26]. Django uses the file to map any function with its compatible URL.
- **wsgi.py** file: a gate point for the WSGI-compatible web server to serve this project.
- **manage.py** file: a command-line utility for executing Django command.

A Django project cannot function without its applications. They are designed to be modular and portable with the purpose that programmers can reuse them. For example, there is an application named *myapp* under *myproject*, and there are some standards in files and folders name so that a Django project can run (displayed in the right part of Figure 10):

- **static/** directory contains front-end files such as CSS, JavaScript, or images.
- **templates/** directory contains HTML files.
- **__init__.py** tells Python that myapp is a package.
- **admin.py** is where the app's models register with Django admin application.

- **apps.py** is a configuration file for the application.
- **models.py** defines the database schema of the application, using Object Relational Mapping (ORM).
- **urls.py** is the app-level URL configuration. It handles the mapping of functions from this app's views to their URLs.
- **tests.py** contains test suites and test cases for this application.
- **views.py** where the functions of the application are located.

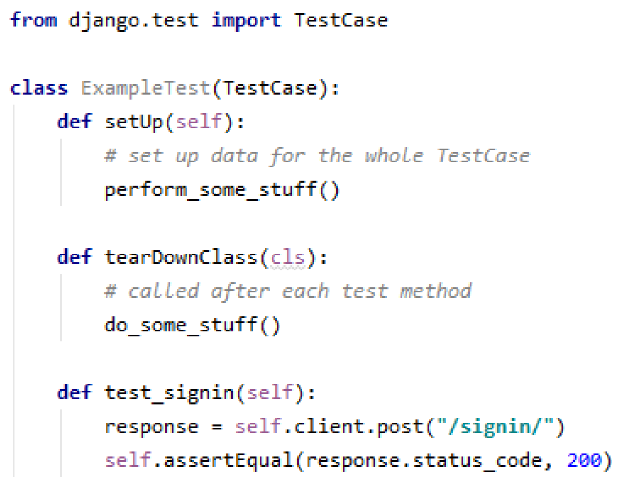
In 2017, Django 2 was released in order to succeed Django 1.11 as Django 1.11 will not be received patch fixes and mainstream support after April 2020. Django 2 introduced many new features, but it has a minor backward-incompatible, it only supports Python 3. This is not a significant concern to worry about because Python 2.7 also will meet its end of life in 2020. Therefore, it is not proposed for students to implement their projects in Python 2.7. These are some major features of Django 2:

- Simplified URL routing syntax that allows for an easier to write and read route patterns without regular expressions.
- More responsive, mobile-friendly admin page.
- Window expressions to provide a way to apply functions on partitions.

2.3.7. Django testing tools

The recommended approach to write tests in Django is using a built-in testing module, which is expanded from the *unittest* package of Python towards web applications development. Django test module can mock up a testing environment, allowing developers to test the views and interact with Django applications programmatically [27]. Developers can simulate GET and POST requests and observe the response. Many data such as URL, status code, template name or even HTML content can be extracted from the response. Additionally, Django testing tools support email functionality, in which sent emails will be redirected to a dummy outbox. From there, every information of messages can be inspected.

A typical Django test case three parts: *setUp* method, tests, and *tearDownClass* method. Figure 11 presents an example of how a test case looks like.



```python
from django.test import TestCase

class ExampleTest(TestCase):
    def setUp(self):
        # set up data for the whole TestCase
        perform_some_stuff()

    def tearDownClass(cls):
        # called after each test method
        do_some_stuff()

    def test_signin(self):
        response = self.client.post("/signin/")
        self.assertEqual(response.status_code, 200)
```

Figure 11. Example of a Django test case.

- *setUp* method: This is where the necessary data is mocked up preparing for the tests because this method is invoked before every test of the test case.
- Tests: requests are made by calling view functions to get a response. The response then can be checked by multiple built-in assertion syntaxes. The test in Figure 11 uses one of the assert functions.
- *tearDownClass* method: If there is a demand to do something after tests, this method is where to do it because it is called after every test.

3. The YAAS Application

This chapter describes the purpose of the course, the requirements of its final project YAAS, the required technologies, and the grading process.

3.1. Overview of the course

The target of the thesis work is a course taught at Åbo Akademi University named *Development of Server-side Web Services*. This is a master-level course that is relevant for students who have moderately advanced programming skills. These are the objectives of this course:

- Understand the necessary knowledge and the challenges involved in the development of web applications and services.
- Be able to handle web services, concurrency, persistence, authentication, session management, and apply them to develop a non-trivial web application.
- Learn how to test web applications and web services.

3.2. The YAAS project

In the course, students must develop individually a project named Yet Another Auction Site (YAAS). It is a web application and a web service to create and participate in auctions similar to other famous auction pages such as ebay.com or huuto.net. This kind of project is chosen as the course project because it is a good example of a service offered as a web application. Those companies owning auction sites do not buy or sell anything; instead, they provide services, i.e., platforms for their members to interact with each other in a convenient, fast, and easy way. The way it works fits the purpose of the course as it requires students to apply in practice different topics taught throughout the course.

The requirements of the YAAS project are provided as a list of use cases that need to be successfully implemented in order to earn points. Each use case has different levels of difficulty and gives a different number of points.

Each use case is broken down into several functional requirements. The main focus of the project is to implement a functional web application, without using too much attention to the graphic design of the web pages rendered in the browser.

3.2.1. Requirements

The following is a brief list of the main requirements for the project:

- **UC1 Create a user account**: any anonymous user can create a new account.
- **UC2 Edit account information**: a logged-in user can change his/her email address and password.
- **UC3 Create a new auction**: A logged-in user can create a new auction with its title, description, minimum price, and deadline.
- **UC4 Edit the description of an auction**: the seller of an active auction can change its description.
- **UC5 Browse and search auctions**: both anonymous and registered users are able to browse the list of active auctions as well as search auctions by title.
- **UC6 Bid**: any logged-in user can bid for an active auction, except its seller.
- **UC7 Ban an auction**: an administrator user can ban an active auction that does not comply with the terms of use of the site.
- **UC8 Resolve auction**: the system should automatically resolve any auction that meets its deadline date and pick a winner for it. The winner is the one with the highest bid on an auction.
- **UC9 Support for multiple languages**: the application must support multiple languages, which is freedom of choice by students, and allow users to change language effortlessly.
- **UC10 Support multiple Concurrent Sessions**: the application must be able to handle multiple concurrent users.
- **UC11 Support for currency exchange**: A user should be able to visualize the prices of the auctions and bids in a different currency based on the latest exchange rate. Exchange rates can be regularly fetched from third-party sites.
- **UC12 RESTful web service**: the application provides some APIs. This use case includes two smaller use cases, **WS1** for browsing, searching, **WS2** for bidding on an auction.

There are also some additional requirements:

- **TREQ4.1.1**: create unit tests for use case UC9.

- **TREQ4.1.2**: create unit tests for use case UC3.
- **TREQ4.1.3**: create unit tests for use case UC10.
- **TREQ4.2**: the application should come with a data generation program for testing purposes with at least 50 users, 50 auctions, and bids for some auctions.

Each use case can contain several requirements. For instance, UC2 has the following requirements:

- REQ2.2: A logged-in user can change his/her email address.
- REQ2.3: A logged-in user can change his/her password.

In total, the project has 12 use cases and 51 requirements. Due to space limitation, we will not provide the complete list of requirements.

3.2.2. Grading process

Each student builds his/her project from scratch and submits the project repository to a GitHub Classroom repository. Students will be graded depending on the number of use cases they implement successfully. Project submissions are evaluated individually by the lecturers.

Figure 12 below demonstrates the current flow of the grading process. Before the course starts, the teachers specify the project specifications that are directly related to the topics of the course. Then the teachers public the specifications so that students can have a brief view of the project. When it is time, the teachers create an assignment on GitHub Classroom and provide the invitation link to the students so they can start the implementation. When the students consider their projects are satisfied with the specifications, they submit their projects by uploading the code to GitHub Classroom repositories.

After the final deadline, teachers download the sources files. For each of the projects, teachers must open in an editor to inspect the code, install the necessary dependencies, run the projects and grade them separately.

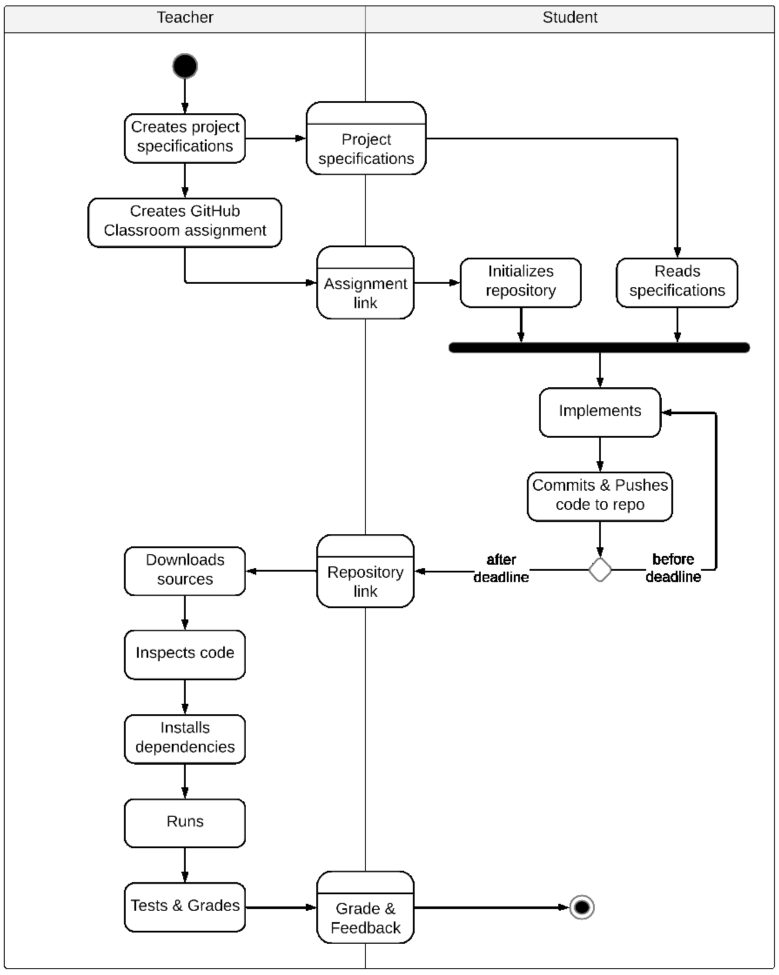


Figure 12. Current flow of grading process.

The number of students who enroll for the course can vary between 40 and 70 people; therefore, it will be a time-consuming task for lecturers to evaluate all projects. It takes considerable effort to check every requirement whether it is implemented or not. The grading process could take up to 6 weeks for students to receive their grades and even longer for those who submit late.

3.2.3. Required technologies

The Django framework is the primary framework to implement this project. During the development, students are able to use third-party Python packages to help them with the implementation. The latest version of Django will be used, which is Django 2.2.5.

For front-side design, students can use HTML and CSS. JavaScript libraries can also be used, but the presentation is not the focus of this project, so the front-end part should be simple and easy-to-use. To store the data, students can choose from a range of SQL databases supported by Django, such as SQLite, MySQL, PostgreSQL, etc. However, SQLite is recommended for practical grading reasons, because SQLite is the default database when creating a new Django project and it is not required to connect to an external server. The IDE used to implement is a personal preference, but PyCharm [28] is recommended because it supports the Django framework, as well as it has all Python tools in one place and provides all the necessities that are needed for productive Python development.

4. Proposed solution

This chapter describes the solution we propose to improve the implementation and the grading process of the YAAS project by applying test-driven development. We will investigate what design decisions need to be made, what portion of the project requirements are testable, and how much of the grading process can be automated.

Figure 13 below illustrates the general workflow of the proposed solution, in which teachers and students have activities that contribute to the whole process.

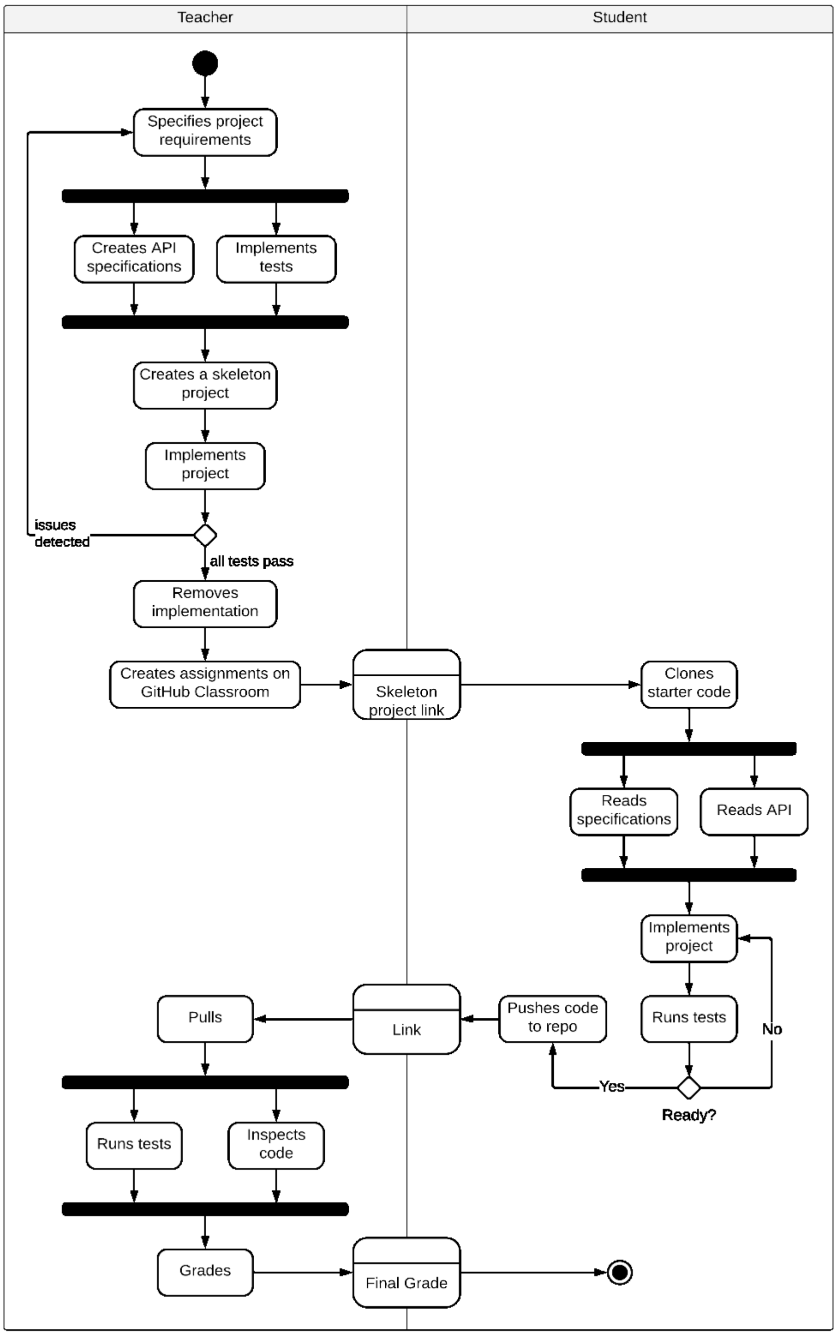


Figure 13. The process of submitting and grading

The proposed solution is to provide students a skeleton project with a set of tests. The same tests can be used both by teachers for grading and by students for evaluating their projects during development.

First of all, the teachers specify the project requirements from what has been taught in the course. Then, teachers create API specifications, containing the needed URLs, and what HTTP methods can be used. After that, acceptance tests are implemented against the defined API. In order to verify that the test are properly defined and the API specification is consistent, the project is also implemented by the lecturers until all the tests pass.

After the preparation is done, they can create an assignment on GitHub Classroom and use the project as a starter code. From a link to the skeleton project, which is provided by the teachers, students can clone it to their computer and start the implementation.

Students should run the tests periodically to check what features they have done and how many points they currently have. The tests can be used as a self-evaluation tool to check which use cases have passed, which have failed. Then they can submit their projects by pushing the code to GitHub Classroom repository.

On the grading side, after the final deadline of the project, teachers use the proposed approach to pull all repositories, run all the tests for each of them, inspect the code, and create an overview report. Finally, the teachers calculate the final grades from this report.

In summary, the results of this thesis will include items below:

1. A new approach to improve teaching and learning experiences.
2. Artifacts to be delivered to students:
    - **Requirements document**. It describes the goal of the YAAS application as well as the detailed objective of every use case.
    - **Interface specification**. It is written based on the detailed requirements that indicate which HTTP methods can be requested on different resources and which resources are expected. The automated tests are relying on it.

- **Project skeleton**, including the tests. The project includes all the files and settings that students need to run and implement.

3. Scripts that help lecturers utilize the approach optimally.

4.1. Design decisions

The challenge of this solution is how to write the tests as generic as possible without enforcing technical implementation decisions. The previous chapter has stated that there are twelve main use cases in the project. However, only eleven use cases will be covered by the tests. The UC10 is the one that will not have tests because the concurrency issue is not easy to test correctly. This use case requires the YAAS project supports concurrent sessions, meaning multiple users can bid a certain auction simultaneously. It is difficult to simulate the concurrent testing environment while ensuring there is room for creativity in the implementation. Thus, teachers will manually verify the UC10 use case.

The solution will follow an approach where the tests are independent of the models and templates so that students can freely design and develop the database and presentation of their projects. However, in order to execute the tests, the API of the web application must be fixed, which in a Django project the API is specified in the *url.py* file.

Besides, all tests are provided in a single file and we enforce that students are not allowed to edit the file during the implementation. If they want to create their own tests, they should put the tests in a different file.

The skeleton project provides a set of URLs and their functions; they will be described later in the following section. The initial stage of this project should only contain enough code and structure to run because students will have different levels of programming skills. If it is too detailed, it will restrain creativity and freedom in implementation. Its structure and interface must be clean, well-ordered with the purpose that a student can start to work immediately after he or she clones it from GitHub. There will be a tutorial to show how to set up and use the framework initially. These concerns will be described in more detail in the following sections.

4.2. Interface specification

An interface specification is created to reflect all the requirements of every use case in term of the interface. The interface specification details what are the URLs that can be accessed for accomplishing its functionality. Each URL is accompanied by the HTTP verbs that can be used. For each type of request, the specification also details the expected response from the server. Below is an example for the UC1 with its requirements. The detailed specifications for the entire API specification document are found in Appendix A.

UC1 - Create a user account

URI: */signup/.*

- GET – get a signup form, return code 200.
- POST – create a user with username and password
  - Sign up without a password, means invalid data, return status code 200.
  - Sign up with already taken username, return status code 400, and an error message is present in the response content (HTML).
  - Sign up with already taken email, return status code 400, and an error message is present in the response content (HTML).
  - Sign up with valid data, return status code 302 because the page would redirect to the index page after a successful signup.

An example request is shown below, describing a POST request to /signup URL and its necessary parameters. The data is written in JSON format.

POST /signup

```
{
    “username”: “user1”,
    “password”: “Password1”,
    “password1”: “Password1”,
    “password2”: “Password1”,
    “email”: “user1@mail.com”
}
```

If the data is valid, the response code should be 302, and if the data is invalid, the response code should be 200 and comes with an error message.

4.3. URL files

All URLs from the API specification file are collected in the *url.py* file and included in the skeleton project. Each URL has a corresponding function (view) that is invoked containing the actual business logic. Figure 14 below shows the URL configuration of the project. *path()* is the URL routing method that maps a URL to its matching view. For intricate URLs with strings and numbers, one can use *re_path()* method to write the patterns (examples can be found in Figure 15).

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('auction/', include('auction.urls', namespace='auction')),
    path('user/', include('user.urls', namespace='user')),
    path('', auction.views.index, name='index'),
    path('signup/', user.views.SignUp.as_view(), name='signup'),
    path('signin/', user.views.SignIn.as_view(), name='signin'),
    path('signout/', user.views.signout, name='signout'),
]
```

Figure 14. URL configuration of the project.

The interface specification file describes what are the URLs used by each user case, what HTTP requests can be sent to those URLs, what parameters they require, and what is the expected response.

Another example is the URL file of the *auction* app, which is an application of this skeleton project. It also contains the URL for the other use cases, as demonstrated in Figure 15. For instance, if the URL is *http://localhost/auction/ban/1*, the server will navigate to the function "ban" and invoke it with parameter "1".

```
app_name = 'auction'
urlpatterns = [
    path('', views.index, name='index'),
    path('search/', views.search, name='search'),
    path('create/', views.CreateAuction.as_view(), name='create'),
    re_path(r'^edit/(\d+)/$', views.EditAuction.as_view(), name='edit'),
    re_path(r'^bid/(\d+)/$', views.bid, name='bid'),
    re_path(r'^ban/(\d+)/$', views.ban, name='ban'),
    path('resolve/', views.resolve, name='resolve')
]
```

Figure 15. URL configuration of auction app.

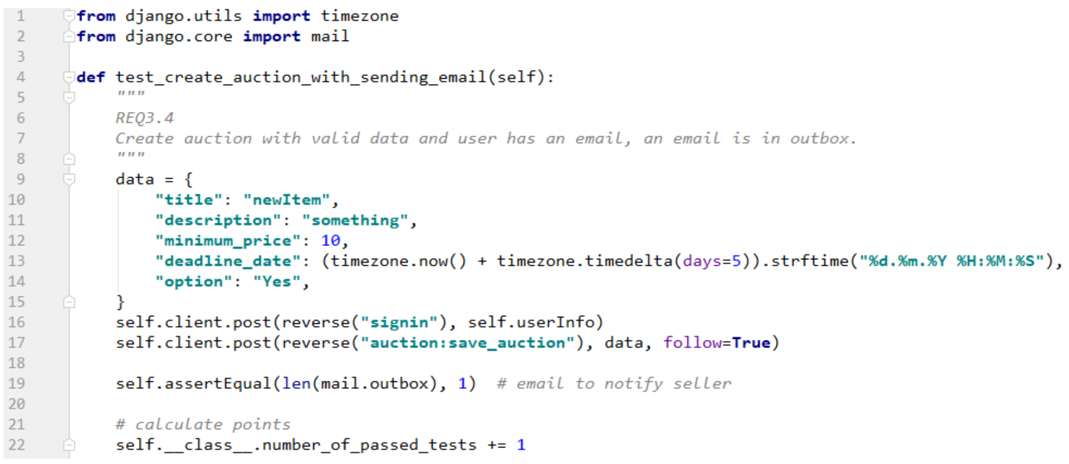
4.4. Implementation of the tests

Each use case has several smaller requirements and these requirements are turned into test methods for that use case. In order to achieve the points of a use case, students must implement to pass all of its test methods. If one of them fails, the use case is also considered failed.

The tests have been implemented based on the given interface specification and URLs above. Each use case is turned into a test case and will be put into the *testsTDD.py* file. A test case has multiple tests corresponding to its specification.

In the process of implementation of the tests, there was an issue in keeping the structure reasonably unbiased. Thus, a different approach in function invocation was used to allow many people with different designs to work seamlessly: the Django utility called *reverse()*, which calls the URLs in urls.py and then invokes the corresponding views (an example can be found in Figure 16 on line 19). The test assertions were also worked around like that.

Moreover, we did not use any direct reference in our tests to the database of the application. Instead, the tests check the responses' status code and message in HTML. That means the application implemented by students must have the same responses to pass the tests. Another way to check the result is to utilize other functions such as browsing or searching to retrieve a list of auctions. Due to the fact that API requirements contain many requirements of sending emails, we used the dummy outbox of the Django core package to collect and test email messages. Figure 16 demonstrates how to use the dummy outbox to test the email system (line 19), where after an email is sent by the system, the email message is expected to be present in the outbox.

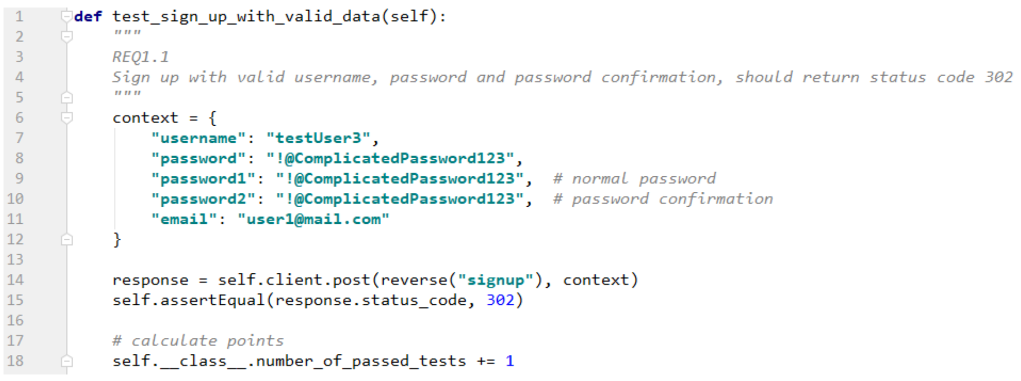


```
1   from django.utils import timezone
2   from django.core import mail
3
4   def test_create_auction_with_sending_email(self):
5       """
6       REQ3.4
7       Create auction with valid data and user has an email, an email is in outbox.
8       """
9       data = {
10          "title": "newItem",
11          "description": "something",
12          "minimum_price": 10,
13          "deadline_date": (timezone.now() + timezone.timedelta(days=5)).strftime("%d.%m.%Y %H:%M:%S"),
14          "option": "Yes",
15      }
16      self.client.post(reverse("signin"), self.userInfo)
17      self.client.post(reverse("auction:save_auction"), data, follow=True)
18
19      self.assertEqual(len(mail.outbox), 1)  # email to notify seller
20
21      # calculate points
22      self.__class__.number_of_passed_tests += 1
```

Figure 16. An example of testing email service.

As mentioned in the Design decisions section, the tests must be independent of the models so students can deliberately design them. The *reverse()* method allows interacting with URLs and avoiding explicitly using specific models. Due to the fact that the tests cannot access any model directly, test assertion will be used to check the status code or text in the response messages. The test results are expected to show successful and failed tests, how many points they provide, and the total points achieved.

For instance, the use case UC1 has five requirements with it, so its test case has five test methods. Each test method checks the exact response status code or message in the response messages. Figure 17 illustrates one of the methods of UC1. In this method, the function signup is invoked with some data, and it needs to return a response with status code 302 to pass the test. If the method passes, it will increment the *number_of_passed_tests* variable by one for the point calculation of this test case later (line 18). For more detail, Appendix B shows the complete code of the tests corresponding to UC1.

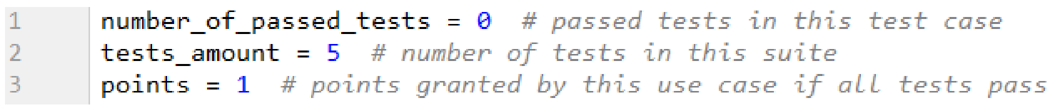


```
1   def test_sign_up_with_valid_data(self):
2       """
3       REQ1.1
4       Sign up with valid username, password and password confirmation, should return status code 302
5       """
6       context = {
7           "username": "testUser3",
8           "password": "!@ComplicatedPassword123",
9           "password1": "!@ComplicatedPassword123",  # normal password
10          "password2": "!@ComplicatedPassword123",  # password confirmation
11          "email": "user1@mail.com"
12      }
13
14      response = self.client.post(reverse("signup"), context)
15      self.assertEqual(response.status_code, 302)
16
17      # calculate points
18      self.__class__.number_of_passed_tests += 1
```

Figure 17. A test method of UC1

4.5. Support for grading

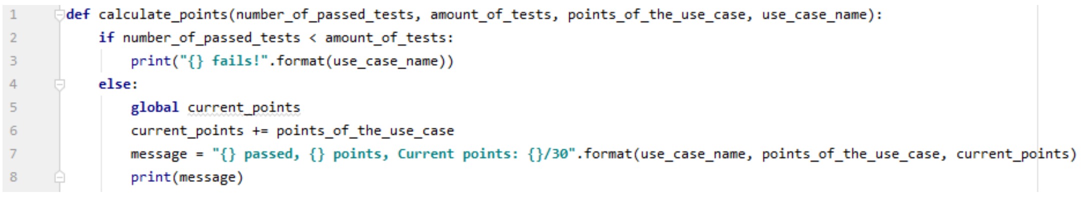
Every test case has some class-level variables to track and show the number of tests, passed tests, and points of the test case (Figure 18). Whenever the test runs successfully (i.e., the assertion does not raise an exception) the last code statement in Figure 17 is executed to mark that test has passed.



```
number_of_passed_tests = 0  # passed tests in this test case
tests_amount = 5  # number of tests in this suite
points = 1  # points granted by this use case if all tests pass
```

Figure 18. Setup code for calculating points.

When a test case completes its execution, its *tearDownClass()* method is invoked, which is only called at the end of the test case, to calculate points (Figure 19). We use this method to check if all tests of the test case are passed (line 2 in Figure 19). If there is a failed test, the system will prompt a failure message (line 3). If all the tests pass, the award points of this use case will be aggregated to the final points of a student (line 5-6), and the system will print a success message (line 7).



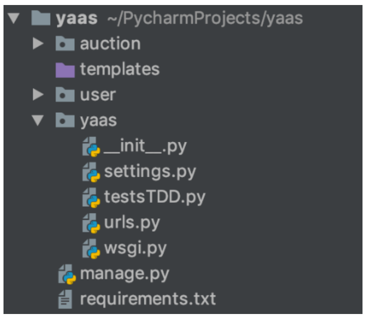
```
1  def calculate_points(number_of_passed_tests, amount_of_tests, points_of_the_use_case, use_case_name):
2      if number_of_passed_tests < amount_of_tests:
3          print("{} fails!".format(use_case_name))
4      else:
5          global current_points
6          current_points += points_of_the_use_case
7          message = "{} passed, {} points, Current points: {}/30".format(use_case_name, points_of_the_use_case, current_points)
8          print(message)
```

Figure 19. Points calculating function.

4.6. The structure of the skeleton project

The YAAS skeleton project follows closely to the Django files structure. As illustrated in Figure 20, the structure is simple and stays true to Django nature with some additions. The *requirements.txt* contains required packages which are Django 2.2.5, freezegun 0.3.12, and Django rest framework 3.10.2. The freezegun package is for manipulating time when writing tests, and the Django rest framework is required to implement REST web services. The *templates* folder is the place for HTML and global front-end files.

Inside the *yaas* directory, there is *settings.py* file containing the project settings and configuration. There is *testsTDD.py* that contains all the test cases that students need to implement against. The project-level url.py file is also located here that contains URL configuration.



```
▼ yaas ~/PycharmProjects/yaas
    ▶ auction
      templates
    ▶ user
    ▼ yaas
        __init__.py
        settings.py
        testsTDD.py
        urls.py
        wsgi.py
      manage.py
      requirements.txt
```

Figure 20. Structure of YAAS project.

In the *settings.py* file, most of the configuration is the provided by Django, and students are free to modify, but *INSTALL_APPS* is split into *PREREQ_APPS*, which is for Django apps and packages, and *PROJECT_APPS*, which is for user-defined apps (Figure 21). Students can make their applications and add them here if needed.

```
PREREQ_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
]
PROJECT_APPS = [
    'auction.apps.AuctionConfig',
    'user.apps.UserConfig'
]
INSTALLED_APPS = PREREQ_APPS + PROJECT_APPS
```

Figure 21. INSTALLED_APPS setting of settings.py.

By default, Django uses *db.sqlite3* which is recommended during the project, since it stores the data locally in a file without the need to connect to an external server. This is set in the skeleton project *DATABASES* in the *settings.py* file (Figure 22) and students are not allowed to change it.

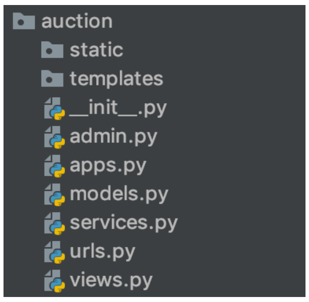
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Figure 22. DATABASES of settings.py.

There are two apps in the project, *auction* and *user*, represented by two directories with the same names. Students could change the functions but not the name and namespace.

4.6.1. Auction application

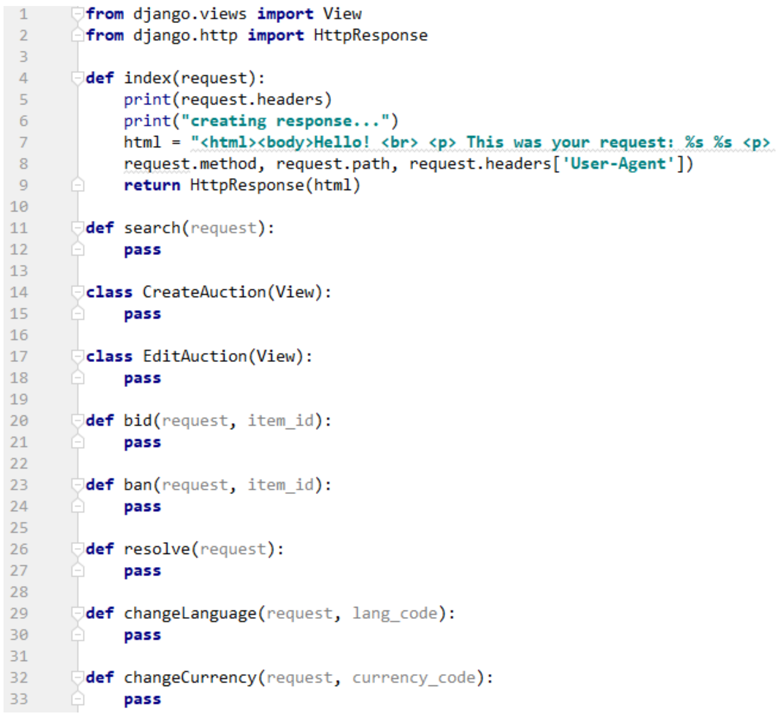
The auction application folder contains all the files necessary for a working app (Figure 23). *__init__.py* is to inform this is a Python package and *apps.py* is the configuration file for the application. Additionally, static and templates folders are pre-created so students can store their HTML, CSS or JavaScript files.



```
auction
    static
    templates
    __init__.py
    admin.py
    apps.py
    models.py
    services.py
    urls.py
    views.py
```

Figure 23. Files structure of Auction application

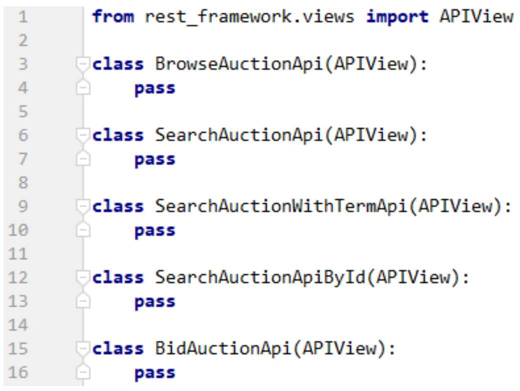
The *urls.py* file contains URL configuration for the auction application (refer to Figure 15 for more details). The *views.py* file (Figure 24) contains functions for the auction application. In the beginning, all of these functions are empty, and students must implement them. These functions are corresponding to the URLs in the aforementioned *urls.py* file. There is also no models provided, students need to create by themselves.



```python
from django.views import View
from django.http import HttpResponse

def index(request):
    print(request.headers)
    print("creating response...")
    html = "<html><body>Hello! <br> <p> This was your request: %s %s <p>
    request.method, request.path, request.headers['User-Agent'])
    return HttpResponse(html)

def search(request):
    pass

class CreateAuction(View):
    pass

class EditAuction(View):
    pass

def bid(request, item_id):
    pass

def ban(request, item_id):
    pass

def resolve(request):
    pass

def changeLanguage(request, lang_code):
    pass

def changeCurrency(request, currency_code):
    pass
```

Figure 24. Initial code for the views of auction application.

Figure 25 demonstrates the initial code for the *services.py* file. Similar to the views, these services are empty and ready for students to implement. The functions from both *views.py* and *services.py* will be tested using the testsTDD.py file.

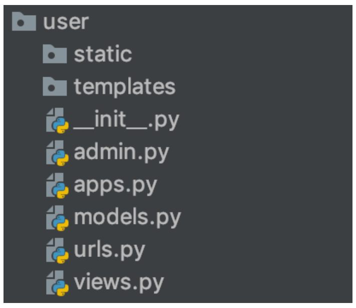


```python
from rest_framework.views import APIView

class BrowseAuctionApi(APIView):
    pass

class SearchAuctionApi(APIView):
    pass

class SearchAuctionWithTermApi(APIView):
    pass

class SearchAuctionApiById(APIView):
    pass

class BidAuctionApi(APIView):
    pass
```

Figure 25. Initial code for the services of auction application.

4.6.2. User application

Figure 26 depicts the details of the user application. It has *__init__.py* to indicate it is a Python package. It also has static and templates folders like auction application to store HTML, CSS, and JavaScript files. *apps.py* is used to store the configuration for registration in settings.py.



```
user
    static
    templates
    __init__.py
    admin.py
    apps.py
    models.py
    urls.py
    views.py
```

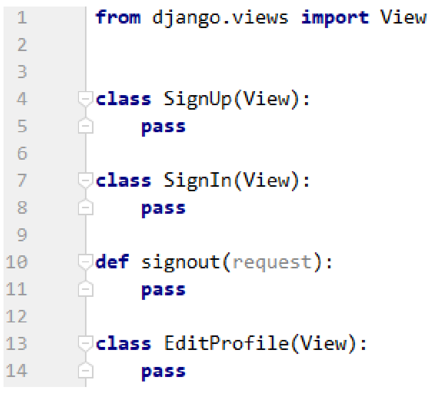
Figure 26. Files structure of User application.

The *urls.py* file contains URL configuration for the user application. Its structure is similar to the auction URLs. Figure 27 displays the URLs from the file.

```
app_name = 'user'
urlpatterns = [
    path('profile/', views.EditProfile.as_view(), name='editprofile'),
    path('', views.EditProfile.as_view(), name='user')
]
```

Figure 27. URL configuration for the user application.

The *views.py* contains empty functions for user application. The detailed code can be seen in the Figure 28. Later, students must implement them to make the project functional. These functions then will be tested using the *testsTDD.py* file. The code snippet below is an example of how signup view is implemented as an empty function.



```
from django.views import View


class SignUp(View):
    pass

class SignIn(View):
    pass

def signout(request):
    pass

class EditProfile(View):
    pass
```

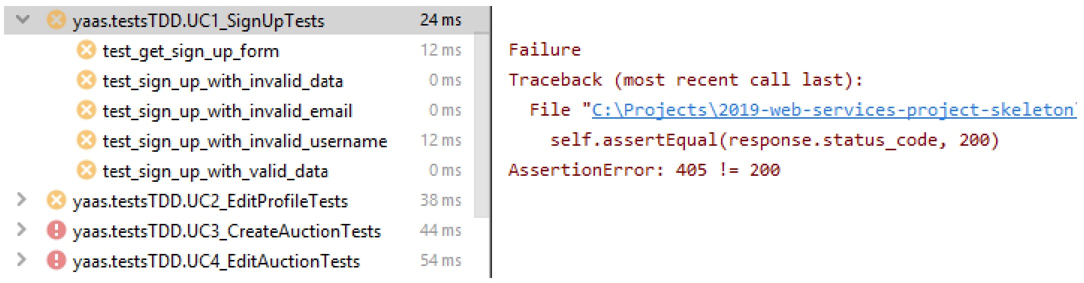
Figure 28. Initial code for the views of user application

4.7. Results

Despite the project specification has 51 requirements, some of them can be checked together and some does not have tests for them. Therefore, in total, there are 40 tests provided in the skeleton project, which are located in the testsTDD.py file. Only the requirements of UC10 are not tested because the concurrency issue is not easy to test correctly.

The time it takes to execute the tests will depend on how complete the implementation code is, the operating system, and the power of the system it runs on. It can vary from 15 to 60 seconds to execute all tests. For example, with a 3.30GHz Intel Core i-5, 8GB of RAM, and onboard Intel graphics card on a Windows computer, it takes roughly a second to run all the failed tests initially and about 20 seconds for the implemented project to pass all the tests. On the contrary, a MacBook Pro 2016 with a 2.9GHz Intel Core i-5 and 8GB of RAM will take up to 50 seconds to run all the tests.

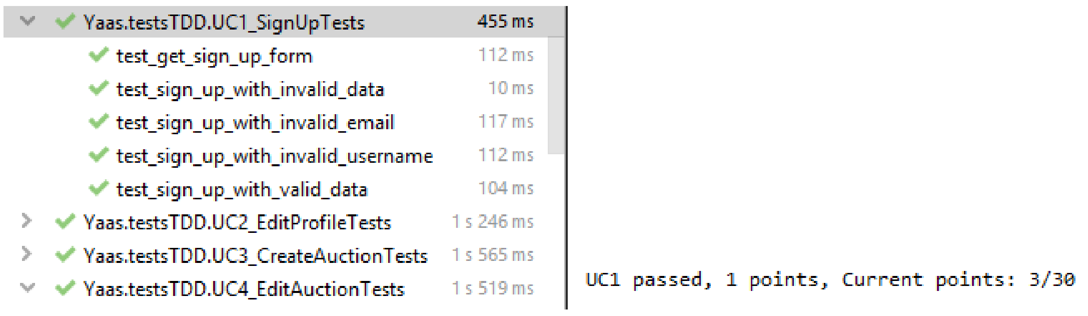
The test report, provided by the Django unit testing tools, will show what tests have failed or passed and how many points a project is currently worth. Figure 29 and Figure 30 demonstrate how the test report looks like initially and after the tests pass when using the PyCharm IDE.



```
yaas.testsTDD.UC1_SignUpTests                 24 ms
    test_get_sign_up_form                     12 ms
    test_sign_up_with_invalid_data             0 ms
    test_sign_up_with_invalid_email            0 ms
    test_sign_up_with_invalid_username        12 ms
    test_sign_up_with_valid_data               0 ms
yaas.testsTDD.UC2_EditProfileTests            38 ms
yaas.testsTDD.UC3_CreateAuctionTests          44 ms
yaas.testsTDD.UC4_EditAuctionTests            54 ms
```

```
Failure
Traceback (most recent call last):
  File "C:\Projects\2019-web-services-project-skeleton
    self.assertEqual(response.status_code, 200)
AssertionError: 405 != 200
```

Figure 29. Failed tests.



```
Yaas.testsTDD.UC1_SignUpTests                455 ms
    test_get_sign_up_form                    112 ms
    test_sign_up_with_invalid_data            10 ms
    test_sign_up_with_invalid_email          117 ms
    test_sign_up_with_invalid_username       112 ms
    test_sign_up_with_valid_data             104 ms
Yaas.testsTDD.UC2_EditProfileTests        1 s 246 ms
Yaas.testsTDD.UC3_CreateAuctionTests      1 s 565 ms
Yaas.testsTDD.UC4_EditAuctionTests        1 s 519 ms
```

```
UC1 passed, 1 points, Current points: 3/30
```

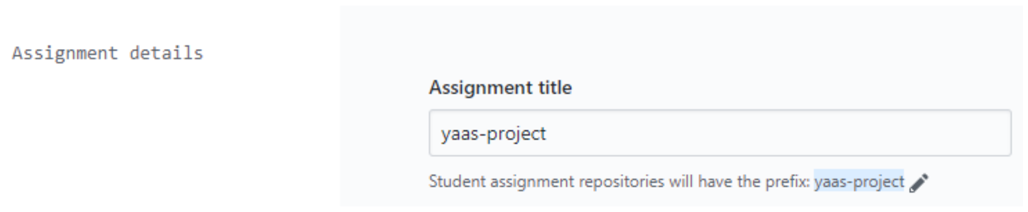
Figure 30. Passed tests.

If a student uses PyCharm as IDE for implementing, he or she can select each test case or test method to see the cause of the failure. Moreover, the overall result shows a summary that includes which tests have passed, how many points were granted, and the total current points (Figure 23).

When all the tests and preparations are done, teachers creates an assignment on GitHub Classroom that using the skeleton project as a starting project for students. The teachers then provide the link to the project on GitHub so students can download and implement from there.

4.8. Configuring GitHub Classroom

This section is to describe how GitHub Classroom should be configured for this approach by a teacher. Firstly, the teacher creates an assignment with title reflecting the YAAS project name (Figure 31). The title should be detailed and compact.


Assignment details

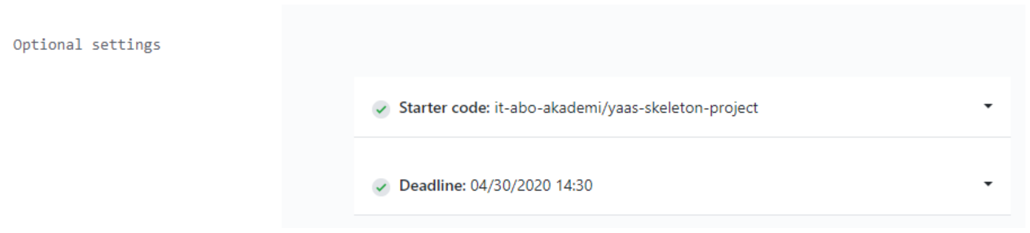
Assignment title

yaas-project

Student assignment repositories will have the prefix: yaas-project

Figure 31. Creating an assignment.

Then, the teacher chooses a starter code for the assignment, which is the skeleton project with empty functions that has been presented in this chapter (Figure 32). After that, the assignment can be created correctly and generate an invitation link.

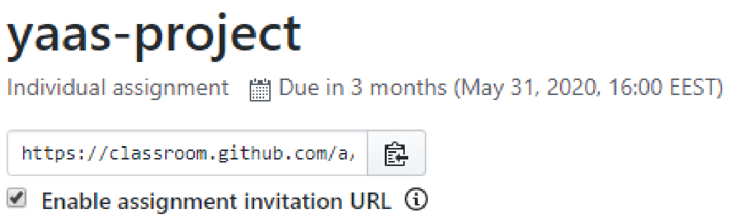

Optional settings

Starter code: it-abo-akademi/yaas-skeleton-project

Deadline: 04/30/2020 14:30

Figure 32. Choosing a starter code for the assignment.

Finally, the teacher can deliver the invitation link (Figure 33) to students. When the students click the link for the assignment, both the students and the teacher become maintainers/admin of the repository with full rights to access the code and make changes.



yaas-project

Individual assignment Due in 3 months (May 31, 2020, 16:00 EEST)

https://classroom.github.com/a/

[✓] Enable assignment invitation URL

Figure 33. The assignment invitation link.

4.9. How to use the framework by students

A tutorial is suggested and created in order to make it easier and more comfortable for students to use this framework in their final project. Firstly, students need to carefully read the YAAS project specification provided by the lecturers to understand the actual scope and target of the project. After that, go to the GitHub Classroom link of the project, this should be found in the specification, and clone or download the project to local computers. Before running the project, students must perform some initial steps to set up all the prerequisites.

4.9.1. Configuring settings

When students open the project the first time, they are required to set the Python interpreter for it. They can change it in the settings of PyCharm, but the virtual environment (venv) is the lightest and most optimal one. At that time, the system will read the *requirements.txt* file and suggest installing the packages inside it. Then, students need to initialize the database by running the following command in their project directory: *$ ./manage.py migrate*.

After all the setup completed, one should move to the settings.py file. It is the settings file for a Django project, and there are some noteworthy attributes:

- **SECRET_KEY**: This key is for cryptographic purposes that the Django installation requires to function correctly. It should be a unique and unpredictable value. To create this key, students can use an online generator [29], and paste it over there. Django will not start if the SECRET_KEY is empty.
- **INSTALLED_APPS**: A list of strings of used packages and apps in this project. It is split into PREREQ_APPS and PROJECT_APPS. The former

consists of Django packages and the latter contains project apps defined by students. One can add more packages or apps in their appropriate locations.

- **TEMPLATES** and **STATIC_URL**: They have default values as in Django convention. So, the folder that contains HTML files and the one that contains JS and CSS files name ‘templates’ and ‘static’ respectively.
- **AUTH_PASSWORD_VALIDATORS**: A list of strings for types of password validators. At first, it is empty for the sake of testing.

4.9.2. Running tests

Students are not allowed to make any changes to the *testsTDD.py* file. If they want to create additional tests, they should create a file named “test*.py”. There are several ways to run the tests from PyCharm or commands.

- PyCharm:
  1. Choose a test file and click the Run button on the Toolbar (Figure 34).

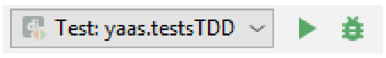


Figure 34. Option 1 for running tests.

  2. Right-click on a file and then click Run tests (Figure 35).

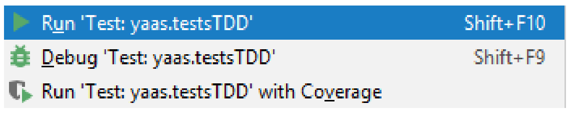
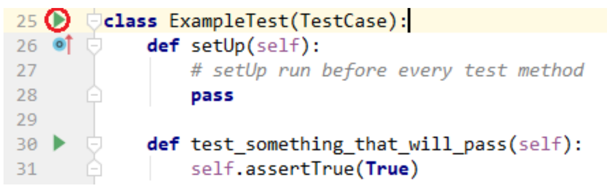

Run 'Test: yaas.testsTDD' Shift+F10
Debug 'Test: yaas.testsTDD' Shift+F9
Run 'Test: yaas.testsTDD' with Coverage

Figure 35. Option 2 for running tests.

  3. Click a run button at the beginning of a test case or a test to run them particularly (Figure 36).

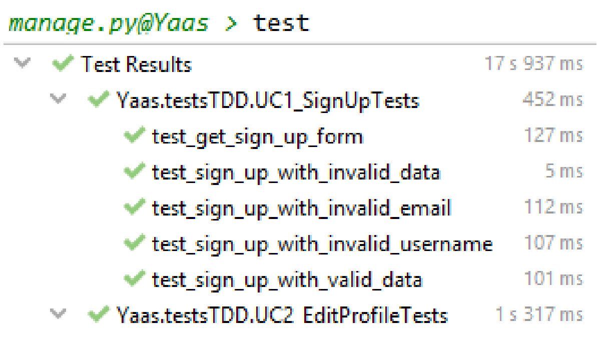


```
class ExampleTest(TestCase):
    def setUp(self):
        # setUp run before every test method
        pass

    def test_something_that_will_pass(self):
        self.assertTrue(True)
```

Figure 36. Option 3 for running tests.

- Using Manage.py utility provided by Django. These are several options:

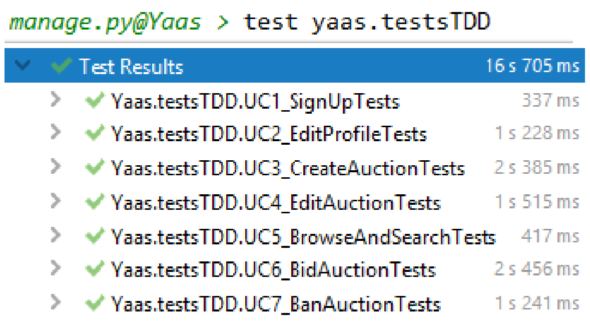
1. Run all the tests (Figure 37).



```
manage.py@Yaas > test
✓ Test Results                                 17 s 937 ms
  ✓ Yaas.testsTDD.UC1_SignUpTests                  452 ms
      ✓ test_get_sign_up_form                      127 ms
      ✓ test_sign_up_with_invalid_data               5 ms
      ✓ test_sign_up_with_invalid_email            112 ms
      ✓ test_sign_up_with_invalid_username         107 ms
      ✓ test_sign_up_with_valid_data               101 ms
  ✓ Yaas.testsTDD.UC2 EditProfileTests          1 s 317 ms
```

Figure 37. Option 1 for running test using manage.py utility.

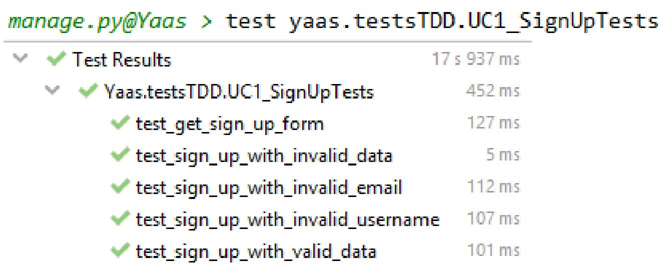
2. Run all the tests in testsTDD.py file (Figure 38).



```
manage.py@Yaas > test yaas.testsTDD
✓ Test Results                                   16 s 705 ms
  ✓ Yaas.testsTDD.UC1_SignUpTests                    337 ms
  ✓ Yaas.testsTDD.UC2_EditProfileTests            1 s 228 ms
  ✓ Yaas.testsTDD.UC3_CreateAuctionTests          2 s 385 ms
  ✓ Yaas.testsTDD.UC4_EditAuctionTests            1 s 515 ms
  ✓ Yaas.testsTDD.UC5_BrowseAndSearchTests           417 ms
  ✓ Yaas.testsTDD.UC6_BidAuctionTests             2 s 456 ms
  ✓ Yaas.testsTDD.UC7_BanAuctionTests             1 s 241 ms
```

Figure 38. Option 2 for running test using manage.py utility.

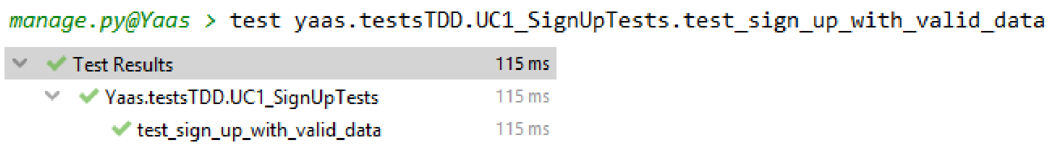
3. Run just one test case (Figure 39).



```
manage.py@Yaas > test yaas.testsTDD.UC1_SignUpTests
✓ Test Results                                 17 s 937 ms
  ✓ Yaas.testsTDD.UC1_SignUpTests                  452 ms
      ✓ test_get_sign_up_form                      127 ms
      ✓ test_sign_up_with_invalid_data               5 ms
      ✓ test_sign_up_with_invalid_email            112 ms
      ✓ test_sign_up_with_invalid_username         107 ms
      ✓ test_sign_up_with_valid_data               101 ms
```

Figure 39. Option 3 for running test using manage.py utility.

4. Run just one test method (Figure 40).



```
manage.py@Yaas > test yaas.testsTDD.UC1_SignUpTests.test_sign_up_with_valid_data
✓ Test Results                              115 ms
  ✓ Yaas.testsTDD.UC1_SignUpTests           115 ms
      ✓ test_sign_up_with_valid_data        115 ms
```

Figure 40. Option 4 for running test using manage.py utility.

The tests are designed to give a wide point of view about a project automatically. Each time a student runs the tests, he or she will know which test cases have passed, which have failed, as well as the results of test methods from those cases.

4.10. Using the framework for grading

To support the grading process, a set of scripts has been implemented to automate different steps performed by the lecturers. The scripts, written in Python, use the GitHub API to download all student projects from GitHub Classroom and store them in a local folder.

Then, one of the scripts will start executing the tests on each project using the command "python grade.py", and save the results in a grading report file with the structure presented in Table 1. For each student, the report includes:

- The name of the student.
- The date of running the script.
- The points received by each use case.
- The total number of points the project is worth.
- A link to the repository of the project.

The scripts can be run not only at the end of the course after the deadline for project submission has passed, but also regularly (e.g., weekly) to check the progress of the students during the course.

Table 1. An example of grade report

Student	**Date**	**UC1**	**UC2**	**…**	**Total**	**Repo link**
Student A	14/11/2019	1	1	…	16	https://github.com/…
Student B	14/11/2019	1	0	…	18	https://github.com/…
Student C	14/11/2019	0	1	…	15	https://github.com/…

5. Evaluation

The result of this work has been applied on a batch of 60 students. These are the preliminary conclusions.

5.1. Benefits

With a project that already has a set of tests, students can self-evaluate their work during the implementation. They will know which parts give points, which functions introduce errors, and what needs to be improved. If the students commit their code regularly, teachers can check their status and give helpful feedbacks rather than only acknowledge the final product.

The tests are another information hub besides technical requirements and API specifications. They can be used as a guide for students, reducing some of the ambiguous issues from the requirements, thereby it would increase their productivity.

Some effort was spent in the beginning by the lecturers to implement the tests and clearly specify in the interface, however, the benefits in terms of efficiency the new approach decreased dramatically the time needed to grade the assignments. As such, we were able to run the automated tests on all 60 projects submitted by students in around 110 minutes on a Windows 10 laptop featuring an Intel i7-7500U CPU with two cores at 2.90GHz and 16GB of RAM. This means less than two minutes per project. Roughly 5 minutes of additional time was allocated on average to manual code inspection of the requirements that had no associated tests. Overall, we have observed a reduction of time of more than 65%.

During the experiment, some errors and too strict assumptions have been identified in the specification files and in the acceptance tests. In that case, another script was implemented to update the test suite and distribute the changes to all the student repositories in GitHub. Using the GitHub API and having admin access to student repositories allowed us to easily and programmatically distribute the updates to students.

Student feedback has been collected to evaluate and improve the framework. Some of them said the new approach was good because they could follow the project specifications more accurately and it was a useful tool to evaluate the project themselves. However, some students had a hard time using the tests. They said the tests were too restrictive and sometimes confusing in term of implementation.

5.2. Limitations and Challenges

This is the first time an automatic grading project is applied to the course, so it prone to have some issues. There were some challenges to implement tests for some use cases and some limitations on how the framework works. The students must follow the naming rules and model types to run the project. The tests might not cover every requirement and might limit the creativity. Then since the assertion of tests only checks the response status code and the HTML, some tests could be passed without implementing the requirements correctly. However, the lecturers can always manually inspect the code to eliminate this situation.

The UC3 - create auctions - has had acceptance tests in the beginning, but they were removed after release. The reason was that there is a requirement for confirmation of an auction when creating. The requirement does not explicitly state that a confirmation form can be verified on the client-side or the server-side. This ambiguous problem caused the tests to fail for students who followed the server-side approach. Therefore, tests for UC3 were omitted for automated grading, and teachers manually verified this requirement. For the next version of the course, when the requirements are updated to be more detailed, the tests will be returned with a clearer and unique approach.

There were some other challenging use cases in the requirements. The UC8 - resolve auctions - requires some auctions to have deadlines in the past to be resolved. The initial idea was to create an auction with a deadline in five seconds, so when the resolve function ran, it would meet the deadline and resolve that auction. However, to create a new auction, its deadline date must be three days after the moment. A Django library named *freezegun* [30] is applied to allow Python tests to have more control with the *datetime* module. With this library, it was possible to manipulate the current *datetime* and create artifacts in the past.

The UC9 - change language - and UC11 - change currency - are both relating to managing cookies or sessions. They are difficult to be precisely tested since they would restrict students from implementing freely their cookie handler. The optimal way is to check text present on HTML. These use cases will be manually verified in the implementation code and GUI when grading.

Another use case that does not have tests is UC10 – support multiple concurrent sessions. Theoretically, there are two methods to encounter the multiple concurrent sessions, pessimistic and optimistic. They may result in various ways to implement the problem, and it was challenging to cover them all without forcing students onto a limited path. Therefore, there were tests for UC10 in an earlier phase to cover the optimistic approach, but they were removed since it would be too generic to be meaningful, and the tests are turned into a new requirement TREQ4.1.3. Now Students must write tests for UC10 by themselves. With this bonus requirement, the total points can potentially be over the max 30-point, making it more achievable to reach high grade.

The last requirement is TREQ4.2, which asks for a data generation program. It requires students to do some configuration and command lines to provide an application with initial data. Therefore, this requirement does not have tests.

There are some rules for data type and function parameters that students need to follow to run the tests strictly so that the project can run properly. All the ID fields must be Integer. Students should not explicitly specify a custom primary key; Django will automatically give every model an **id** column in Integer type. When implementing the views for auction and user, some parameters are expected to match what they are in the tests. Model fields are still personal preference. The parameters are below in the exact lowercase:

- Auction: title, description, minimum_price, deadline_date.
- User: username, password, email.

Finally, we could have a CI/CD solution on GitHub Classroom to regularly check the progress of students, but it provides some unnecessary services and does not offer a comprehensive view. CI/CD is useful if students commit code frequently, but that is not mandatory for the project. Teachers only need the final version which passes the tests and hence the results on previous commits are not so relevant. Although the idea of CI/CD is excellent on paper, it is not beneficial in reality, and the Python scripts to automatically pull and push code are more valuable than a CI/CD process.

6. Conclusion

The primary purpose of this thesis was to apply Test-Driven Development to create a framework that not only facilitates the grading of student projects but also provides early feedback for their implementation. The skeleton project can act as a broad guideline helping students to track the implemented requirements. Everything from initial setup to grading is revealed, so students should only care about coding their projects. Additionally, the testing tools help the lecturers when grading students' projects since features need to pass their test cases to be considered complete.

Throughout the thesis, many necessary details, which support the practical part, were explained. Web applications and their structure, concepts, and benefits should be described clearly since this project is a web application. Following this, Extreme Programming was introduced and how it led to TDD. Due to its benefits and application and how relevant it is, TDD was decided as a principal method to use to implement the thesis' project. Furthermore, by this approach, students will have a closer look into a robust method and they can apply in their projects in the future.

The practical part began with an overview of the YAAS project, its requirements, and objectives that the framework needs to fulfill. A brief manual was produced to illustrate how to set up and run the project.

Afterward, to make sure it works properly, the framework was evaluated. The evaluation chapter described what was achieved and what needed improvements in the future. Some changes have appeared after the framework was released to students but we provided an approach to address these challenges.

Web development is a broad topic, and in the internet age, everything is continuously developing. To keep up the pace of development while, at the same time, preventing everything from collapsing is the key to success. Although TDD fits undoubtedly in software development, it has its limitations, such as long learning curve and high initial investment. However, in our opinion, the benefits in terms of automation and early feedback surpassed the limitations, and thus it is planned that we will follow a similar approach in other courses.

REFERENCES

[1] Beck, Kent. *Test Driven Development: By Example*. 1st ed., Addison-Wesley Professional, 2002.

[2] “Django” *Django*, https://www.djangoproject.com/. Accessed 29 October 2019.

[3] Pilla, Mauricio Lima. "Teaching Computer Architectures through Automatically Corrected Projects: Preliminary Results." (2017).

[4] Cai, Yun-Zhan, and Meng-Hsun Tsai. “Improving Programming Education Quality with Automatic Grading System.” Lecture Notes in Computer Science Innovative Technologies and Learning, 2019, pp. 207–215., doi:10.1007/978-3-030-35343-8_22.

[5] “Extreme Programming: A gentle introduction.” *Extreme Programming*, http://www.extremeprogramming.org/. Access 27 July 2019.

[6] Beck, Kent, and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. 2nd ed., Addison-Wesley, 2005.

[7] “What is Extreme Programming? An overview of XP Rules and Values.” *Lucidchart*, 30 May 2018, https://www.lucidchart.com/blog/what-is-extreme-programming. Accessed 27 July 2019.

[8] Okken, Brian. “Test First Programming / Test First Development.” *Python Testing*, 3 March 2015, https://pythontesting.net/agile/test-first-programming/. Accessed 29 July 2019.

[9] Shah, Santosh. “Test Driven Development (TDD) vs Test Last Development (TLD): A comparative study.” *compilehorrors.com*, 3 June 2014, http://compilehorrors.com/test-driven-development-tdd-vs-test-last-development-tld-a-comparative-study/ /. Accessed 29 October 2019.

[10] Saurel, Sylvain. “Introduction to Test Driven Development (TDD).” *Hackernoon*, 28 March 2019, https://hackernoon.com/introduction-to-test-driven-development-tdd-61a13bc92d92. Accessed 2 August 2019.

[11] Sale, David. “Beginning Test-Driven Development in Python.” *Envato Tuts+*, 9 February 2015, https://code.tutsplus.com/tutorials/beginning-test-driven-development-in-python--net-30137. Accessed 2 August 2019.

[12] “What is V-model- advantages, disadvantages and when to use it?” *Try QA*, http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/. Accessed 29 October 2019.

[13] “Unit testing.” *Software Testing Fundamentals*, http://softwaretestingfundamentals.com/unit-testing/. Accessed 9 September 2019.

[14] Mike Jones. “Unit test basics.” *Microsoft Docs*, https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019. Accessed 12 February 2020.

[15] Acosta, Jan and Gajda, Kim. “Test-driven development.” *IBM*, https://ibm.com/garage/method/practices/code/practice_test_driven_development/. Accessed 30 October 2019.

[16] George, Boby, and Laurie Williams. “A Structured Experiment of Test-Driven Development.” *Information and Software Technology*, vol. 46, no. 5, 2004, pp. 337–342., doi:10.1016/j.infsof.2003.09.011.

[17] Sinnema, Remon. “The Differences Between Test-First Programming and Test-Driven Development.” *Java Code Geeks*, 31 December 2012, https://www.javacodegeeks.com/2012/12/the-differences-between-test-first-programming-and-test-driven-development.html. Accessed 2 August 2019

[18] “Getting Started – About Version Control.” *Git*, https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control. Accessed 10 October 2019.

[19] Shklar, Leon and Rosen, Richard. *Web Application Architecture: Principles, Protocols and Practices*. Wiley, 2003. pp. 3–5.

[20] Caballe, Santi, et al. “Building a Software Service for Mobile Devices to Enhance Awareness in Web Collaboration.” *2014 International Conference on Intelligent Networking and Collaborative Systems*, 2014, pp. 369–376., doi:10.1109/incos.2014.11.

[21] “HTML & CSS.” *W3C*, https://www.w3.org/standards/webdesign/htmlcss, Accessed 28 October 2019.

[22] “What is JavaScript?” *w3schools*, https://www.w3schools.com/whatis/whatis_js.asp. Accessed 28 October 2019.

[23] "What is a Front-End Developer?" *Frontend Masters*, https://frontendmasters.com/books/front-end-handbook/2018/what-is-a-FD.html. Accessed 28 October 2019.

[24] Fielding, Roy, and Julian Reschke. "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing." *Internet Engineering Task Force (IETF)*, 2014, pp. 4–7., doi:10.17487/rfc7230.

[25] "Overview of ASP.NET Core MVC." *Microsoft .NET*, 8 January 2019, https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.0. Accessed 28 October 2019.

[26] Nigel George. *Mastering Django: Covers Django 2 & 3*. E-book, 2019. https://djangobook.com/mastering-django-2-book/. Accessed 3 August 2019.

[27] "Testing tools" *Django documentation*, https://docs.djangoproject.com/en/3.0/topics/testing/tools/. Accessed 12 February 2020.

[28] "PyCharm: The Python IDE for Professional Developers", JetBrains, https://www.jetbrains.com/pycharm/. Accessed 20 November 2019.

[29] "Django Secret Key Generator", *Djecrety*, https://djecrety.ir/. Accessed 5 November 2019.

[30] Pulec, Steve. "FreezeGun: Let your Python tests travel through time", *GitHub*, https://github.com/spulec/freezegun. Accessed 9 September 2019.

APPENDIX

Appendix A

Interface specification of use cases from UC2 to WS2

UC2 – Edit account information

The specification of this use case:

URI: */user/* or */user/profile/.*

- GET – get profile of the current user
    - Get profile of unauthenticated user, return status code 302 (redirect).
    - Get profile of authenticated user, return status code 200.
- POST – change email/password of a user
    - Change email to the one that already taken. The email stays the same, return code 200.
    - Change email and password. The user should have a new password and email, return code 302.

<u>Example requests:</u>

POST /user/profile

```
{
    “email”: “newemail@yaas.com”,
    “password”: “newpassword1”
}
```

UC3 – Create a new auction

URI: *auction/create/.*

- GET – get a create auction form.
    - Return code 200.
- POST – create an auction
    - Create auction with authenticated user, return status code 302 (redirect).
    - Create an auction with an invalid deadline date. The response content should contain an error message. Status code 200

   - Create an auction with an invalid deadline date format. The response content should have an error message in it. Status code 200.
   - Create an auction with a minimum price lower than 0.01. The response content should contain an error message. Status code 200.
   - Create an auction with valid data, return code 200 (after 302), and success message is in the response content.
   - A user with email creates an auction, return code 200 (after 302), an email is sent to the seller, and a success message is in the response content.

Example request:

POST /auction/create

```
{
    “title”: “item1”,
    “description”: “something”,
    “minimum_price”: 10,
    “deadline_date”: “30/12/2019 20:00:00”
}
```

UC4 – Edit the description of an auction

URI: *auction/edit/<auction_id>/.*

- GET – get an auction to edit
   - Get an auction of other users. The error message is in the response HTML. Status code 200.
   - Get an auction to edit, return code 200.
- POST – edit the description of an auction
   - Edit other user’s auction, error message is in the response HTML, status code 200.
   - Edit valid auction, response HTML contains success message, status code 200.

Example requests:

GET /auction/edit/1

```
{
    “title”: “item1”,
}
```

POST /auction/edit/1

```
{
    "title": "item1",
    "description": "new content"
}
```

UC5 – Browse and Search auctions

Generally, the index page of the application should show the list of active auctions, so URI of Browse should be the main page. And Search's URI: *search/?term=<title>*. In the setup of this test suite, there is a user and an active auction of that user.

- GET
    - Browse for active auctions, return a list of active auctions. Status 200.
    - Search auctions by title, return list of active auctions that match the term. Status code 200.

Example request:

GET /search/?term=bike

UC6 – Bid

URI: *auction/bid/<auction_id>/*. In the setup of this test suite, there are a normal user and an admin user. Two auctions also are created by that user, one is kept active, and the admin user bans one.

- POST – bid on an active auction
    - An unauthenticated user bids, return status code 302 (redirect).
    - A seller bid on an own auction; error message is in response content, status code 200.
    - Bid on an inactive auction, error message is in response content, status code 200.
    - Bid on an auction that has a due deadline date. Error message is in the content of response and status code 200.
    - Bid with amount that less than current bid, response content contains an error message and status code 200.
    - Bid with valid data, return status code 200 (after 302), response contains success message, mails are sent to seller and all the bidders.

Example request:

POST /auction/bid/2

```
{
    “new_price”: 15,
}
```

UC7 – Ban an auction

URI: *auction/ban/<auction_id>/.* In the setup of this test case, there are a normal user, an admin user, and an auction of the normal user.

- POST – ban an auction
    - Normal user bans an auction, return status code 302, and the auction still appears in the list of active auctions.
    - Admin user bans an auction, return code 200 (after 302), auction is not in the list of active auctions, and mails are sent to the seller and bidders.

Example request:

POST /auction/ban/3

UC8 – Resolve auctions

URI: *auction/resolve/.* In the setup of this case, a Python package named *freezegun* is used to create auctions in the past so that the resolve function can work.

- POST – resolve auctions
    - Resolve auctions, all the auctions that have a due deadline date will be resolved, status code 200, response is in JSON format showing resolved auctions.

Example response:

HTTP 200 OK

```
{
    “resolved_auctions”: [“auction1”, “auction2”]
}
```

UC9 – Support for multiple languages

URI: *changeLanguage/<lang_code>/.* Students need to support English and Swedish.

- POST – change language
    - Change language to Swedish, response HTML contains a success message. Status code 200.
    - Change language to English, response HTML contains a success message. Status code 200.

<u>Example request:</u>

POST /changeLanguage/en

UC11 – Support for currency exchange

URI: *changeCurrency/<currency_code>/*. Students need to support USD and EUR.

- POST – change currency
    - Change currency to US Dollar, success message is in response content. Status code 200.
    - Change currency to Euro, success message is in response content. Status code 200.

<u>Example request:</u>

POST /changeCurrency/usd

WS1 – Browse and Search API

This case includes four different URIs that need to be implemented. They are:

api/v1/browse/: browse for a list of active auctions.
api/v1/search/<title>/: search auctions by title without a term.
api/v1/search/?term=<title>/: search auctions with /?term
api/v1/searchid/<auction_id>/: search auctions by id.

This use case specification:

- GET
    - Browse for active auctions, return code 200, and a list of active auctions.
    - Search for auctions by title, return status code 200, and a list of active auctions that contain the title.

- Search for auctions by title with *term*, return status code 200, and a list of active auctions that contain the title.
- Search for an auction by id, return status code 200, and an auction with that id.

<u>Example requests:</u>

/api/v1/search/car

/api/v1/search/?term=bike

<u>Example response:</u>

HTTP 200 OK

```
[
    {
        “title”: “old car”,
        “description”: “very old”,
        “minimum_price”: 500,
        “deadline_date”: “31.12.2019 20:00:00”
    },
    {
        “title”: “new car”,
        “description”: “almost new”,
        “minimum_price”: 1000,
        “deadline_date”: “31.11.2019 20:00:00”
    }
]
```

WS2 – Bid API

URI: *api/v1/bid/(\d+)/.*

- POST – Bid auctions API
    - Bid on own auction. Response has status code 400 and an error message.
    - Bid on banned auction. Response has code 400 and an error message.
    - Bid with invalid amount. Response has code 400 and an error message.
    - Bid with invalid data. Response has code 400 and an error message.
    - Bid with valid data. Response has code 200, a success message, and emails are sent to the seller and all the bidders.

<u>Example request:</u> for user “user1” and password “123”

Authorization: Basic dXNlcjE6MTIz

POST /api/v1/bid/5

```
{
    “new_price”: 12
}
```

<u>Example response:</u>

HTTP 200 OK

```
{
    “message”: “Bid successfully”,
    “title”: “auction1”,
    “current_price”: 12
}
```

Appendix B

Test code for UC1

```
class UC1_SignUpTests(TestCase):
    """UC1: create user"""
    number_of_passed_tests = 0  # passed tests in this test case
    tests_amount = 5  # number of tests in this suite
    points = 1  # points granted by this use case if all tests pass

    @classmethod
    def tearDownClass(cls):
        # check if test case passed or failed
        calculate_points(cls.number_of_passed_tests, cls.tests_amount,
cls.points, "UC1")

    def test_get_sign_up_form(self):
        """
        Get signup form, return status code 200
        """
        response = self.client.get(reverse("signup"))
        self.assertEqual(response.status_code, 200)

        # calculate points
        self.__class__.number_of_passed_tests += 1

    def test_sign_up_with_invalid_data(self):
        """
        REQ1.1
        Sign up without a password and email, return status code 200
        """
        context = {
            "username": "testUser3",
        }
        response = self.client.post(reverse("signup"), context)
        self.assertEqual(response.status_code, 200)

        # calculate points
        self.__class__.number_of_passed_tests += 1

    def test_sign_up_with_valid_data(self):
        """
        REQ1.1
        Sign up with valid username, password and password confirmation,
should return status code 302
        """
        context = {
            "username": "testUser3",
            "password": "!@ComplicatedPassword123",
            "password1": "!@ComplicatedPassword123",  # normal password
            "password2": "!@ComplicatedPassword123",  # confirm password
            "email": "user1@mail.com"
        }

        response = self.client.post(reverse("signup"), context)
        self.assertEqual(response.status_code, 302)

        # calculate points
        self.__class__.number_of_passed_tests += 1

    def test_sign_up_with_invalid_username(self):
        """
        REQ1.1
        Sign up with already taken username, return status code 200.
        """
        context = {
            "username": "testUser1",
            "password": "!@ComplicatedPassword333",
            "password1": "!@ComplicatedPassword333",  # normal password
            "password2": "!@ComplicatedPassword333",  # confirm password
            "email": "userb@mail.com"
        }

        response1 = self.client.post(reverse("signup"), context)
        # create another user with the same username
        response2 = self.client.post(reverse("signup"), context)
        self.assertEqual(response2.status_code, 200)
        self.assertIn(b"A user with that username already exists",
response2.content)

        # calculate points
        self.__class__.number_of_passed_tests += 1

    def test_sign_up_with_invalid_email(self):
        """
        REQ1.1
        Sign up with already taken email, should return status code 200.
        """
        user1Info = {
            "username": "testUser1",
            "password": "!@ComplicatedPassword333",
            "password1": "!@ComplicatedPassword333",
            "password2": "!@ComplicatedPassword333",
            "email": "user1@mail.com"
        }
        user2Info = {
            "username": "testUser2",
            "password": "!@ComplicatedPassword321",
            "password1": "!@ComplicatedPassword321",
            "password2": "!@ComplicatedPassword321",
            "email": "user1@mail.com"
        }

        response1 = self.client.post(reverse("signup"), user1Info)
        # create another user with the same username
        response2 = self.client.post(reverse("signup"), user2Info)
        self.assertEqual(response2.status_code, 200)
        self.assertIn(b"A user with that email already exists",
response2.content)

        # calculate points
        self.__class__.number_of_passed_tests += 1
```