AUGUST 2025

HELPDESK

IMPROVING THE DEVELOPER EXPERIENCE

Department of Computer Science
CIS*4910
Professor Stefan C. Kremer

Created By
Sulakshan Sivakumaran

TABLE OF CONTENTS

INTRODUCTION

WHAT IS HELPDESK?

Created by Stefan C. Kremer, Helpdesk is an application designed to enable students to interact with professors and teaching staff for their respective courses. Using their University of Guelph credentials, students can sign into Helpdesk to gain access to their enrolled courses. Helpdesk grants students the ability to ask questions via ticket submissions, sign up for demo appointments, and request regrades. Professors and teaching staff can use Helpdesk to answer student inquiries, view class lists, and more. This web application seeks to improve the student experience by allowing for quick access to resources and tools to aid their educational needs.

GOAL OF DEVELOPMENT

Over the past 4 months, Helpdesk has been under development to improve the developer experience. In order to improve the developer experience, Helpdesk is currently migrating from its old client-server relationship to a new client-server relationship that is currently undergoing development. This migration aims to make our codebase more modular, less repetitive, and highly cohesive. Overall, we are looking to improve our codebase by writing high-quality, clean, readable code.

TECHNOLOGIES USED

Helpdesk is a web-based application, built using JavaScript, Python, HTML/CSS, Apache, and MySQL. Development occurred on the Helpdesk development server hosted by the School of Computer Science at the University of Guelph.

MIGRATION - IMPROVING THE DEVELOPER EXPERIENCE

Building the client-server relationship

The Helpdesk is a web-based application built on a client-server relationship. Client-server architecture follows a protocol known as request-response, where the client requests information or data, and the server then provides the requested data (Coursera, 2024). Helpdesk follows this architecture with the client requesting data from the server and the server taking in the request and returning a response. For example, in the ***home.html.py*** file in the ***/Resources*** folder, we return a head tag with a reference to a ***.js*** file and a ***.css*** file. When requesting to load the home page, our client also requests the server to retrieve the ***.js*** and ***.css*** files. Our server then looks for these files and returns them as a response. Our server has access to all relevant pages through RESOURCES_NEW.

RESOURCES_NEW is a dictionary of all of the modules in the ***/Resources*** folder. This dictionary stores a key of the modules by file name and a value of a running instance of that module. All of the files in the ***/Resources*** folder are saved to this dictionary. When a file is modified and updated, that respective module will be reloaded, with the newest version of that instance being saved to the dictionary. When a request is made, we first retrieve the running instance from the dictionary based on the given path. We then call the serve function using that instance to return a response, which will be an HTML page. This is how the client-server relationship is built within the Helpdesk application.

Writing Modular Code

Modular programming is a technique that simplifies complex software development by breaking it down into smaller, manageable pieces called modules (Daily.Dev, 2024). Each individual module serves a single purpose within the larger system (Daily.Dev, 2024). By writing modular code, we are able to create highly cohesive units with low coupling between them. In regard to the Helpdesk application, writing modular code improves the developer experience in many ways. Writing modular code allows developers to create maintainable, reusable and scalable code.

Maintainable code is code that can be debugged, tested and updated without affecting other parts of the system. For example, the ***sql.html.py*** file in ***/Resources*** is a module that was created for the SQL feature. Any changes that need to be made to this feature will take place in this file, reducing the number of places a developer will need to touch. This highly decreases the amount of work needed to be done by a developer whilst keeping the possibility for bugs and errors to a minimum.

Reusable code simply refers to code that can be reused. Reusing code decreases the number of lines in a codebase, improving readability. By separating code into modules, we can simply call our modules when needed, avoiding the need to duplicate the code that is set up within the modules. For instance, the ***home.html.py*** file in ***/Resources*** calls various modules in the def menu(self, info) function. This way, we are able to call our endpoints that are set up within their respective modules, without needing to recreate these endpoints within the home module. This removes any need for a developer to manually duplicate any work done.

Scalable code is code that can be extended and updated over time without reformatting existing logic. Writing modular code allows us to build modules that can be improved when the need arises. For example, if we wanted to rewrite our upload feature, a developer would only be required to locate the ***upload.html.py*** file in ***/Resources*** and update the def serve(self, info) function. As our endpoints are set up within their respective modules, any changes made to this file would require no other changes to any other files within the system, due to its low coupling that is enabled by its modularity.

DRY - Don't repeat yourself

DRY, standing for "Don't repeat yourself", is a standard software development principle that was created to remove duplicated logic (Aibin, 2023). By adhering to DRY in our codebase, we are making the codebase more maintainable, as if we wanted to update our logic, we are only required to make this change in one place rather than in every area where our logic is repeated (Aibin, 2023). An instance where the DRY principle is applied in the Helpdesk codebase is with the function def add_file_to_resources(full_path, file_name) in ***helpdesk.py*** (located in ***/modules***). This function was created because there was repeated functionality used to add a file to our RESOURCES_NEW dictionary in both the def load_all_resources() and def load_resource_new(url). We reduce the redundancy by making a helper function that performs this operation for us. We then call this function in all required places. This also means that in the future, if we decide to update this functionality, we only need to update the helper function rather than updating the code in every place it is called. Additionally, the DRY principle improves the readability of our code as there will be less redundancy (Aibin, 2023).

Cohesion & Coupling

Cohesion and Coupling are design concepts that are commonly used when designing modular systems (Pagade, 2025).

Cohesion is the degree to which the elements inside a module belong together (Pagade, 2025). High cohesion occurs when a module contains elements that are tightly related to each other (Pagade, 2025). Highly cohesive modules will have a single responsibility. On the other hand, a module with low cohesion will have elements that are unrelated to one another (Pagade, 2025). Throughout the development of the Helpdesk application, modules were created to achieve high cohesion. This was achieved by creating modules that had a single, clear responsibility and contained only properties and methods relevant to that responsibility. For example, modules were made for each feature that is available to the user. For instance, the SQL feature has its own unique module. This module is solely responsible for displaying the database table structures and only contains the methods and properties that are required to make this functionality work. By following a highly cohesive design, developers can easily understand what each module does through proper naming structure, and can extend and update functionality in the future with minimal changes across the codebase.

Coupling is the degree of interdependence between software modules (Pagade, 2025). It deals with the concept of how changing one thing may require changes in another (Pagade, 2025). Two modules have high coupling if they are closely connected (Pagade, 2025). Low coupling occurs when two modules are not connected or are very loosely connected. Throughout the development of the Helpdesk application, modules were created to have low coupling. This was achieved by creating modules that were unrelated to each other and did not rely on one another. As mentioned before, each feature available to the user had its own module. All code related to that feature would be stored in its respective module, requiring no assistance from other modules, creating a low-coupled relationship. Low coupling improves the developer experience as making changes in the future would only require changes in a single or minimal number of modules. It is very difficult for a developer to make changes in a highly coupled system, as making changes in one module would likely cause errors and bugs to occur in other modules.

FUTURE IMPROVEMENTS

Throughout the migration development, Helpdesk has significantly improved its code quality, with further improvements still needing to be made. Helpdesk code quality can be improved by applying key object-oriented programming concepts, specifically Inheritance and Polymorphism.

Inheritance is a mechanism where you can derive a class from another class for a hierarchy of classes that share a set of attributes and methods (Janssen, 2025). A class can be derived from another class where one class is known as the parent class and the other class derives from the parent, known as the child class (Janssen, 2025). Inheritance is beneficial when we want to replicate properties and methods, essentially allowing a developer to reuse code without duplicating it. Taking a look at the Helpdesk codebase, we are able to identify that the modules in the ***/Resources*** folder all follow the same structure. For example, they all contain a `def isauth( self, user_roles )` and a `self.modifiedDate`. In this scenario, we can use inheritance by creating a base page class with an `isauth()` function definition and a `modifiedDate` property. Through inheritance, all of the child classes will inherit these methods and properties, removing the need for this code to be rewritten in these modules. This increases code readability and helps our codebase follow the DRY principle.

Polymorphism describes the concept that we can access objects of different types through the same interface (Janssen, 2025). Polymorphism can be achieved through method overriding and/or method overloading. With our inheritance improvements, we can use method overriding/overloading to achieve polymorphism. By using inheritance, we can have our modules in the ***/Resources*** folder follow the same structure. However, if we decide that we would like to have a different implementation for a property or method, we can override that method and have a new implementation while still having the same structure. For instance, let's say we have a base Page class with an `isauth` function that our modules inherit from. We can have a child class that can have the same exact isauth function but performs a different action within its body. This is known as method overriding, where we are overriding the base definition. This enables a developer to keep a structure within all modules, but also enables these modules to have flexibility if needed.

CONCLUSION

The Helpdesk application was created as a means to improve the student experience. It was built to improve communication between students and the teaching staff of various classes. Over the past 4 months, Helpdesk has been under development to improve the developer experience, with improvements in the codebase being made to improve future development processes. Helpdesk has become more modular, with the codebase improving in maintainability and scalability, and has a greater focus on writing good quality code.

REFERENCES

Aibin, M. (2023, June 17). Dry software design principle | baeldung on computer science. DRY Software Design Principle. https://www.baeldung.com/cs/dry-software-design-principle

Coursera. (2024, October 3). *What is client-server architecture?*. Coursera. https://www.coursera.org/articles/client-server-architecture

Daily.Dev. (2024, April 10). What is modular programming? https://daily.dev/blog/what-is-modular-programming

Janssen, T. (2025, February 6). *OOP concept for beginners: What is inheritance?*. Stackify. https://stackify.com/oop-concept-inheritance/

Janssen, T. (2025b, February 10). *OOP concepts for beginners: What is polymorphism*. Stackify. https://stackify.com/oop-concept-polymorphism/