**ÇANKAYA UNIVERSITY**Faculty of Engineering  
Department of Software Engineering  
  
**SENG 272 – Software Engineering Project***Developer Project Report*

Prepared by:

Oğuzhan Tarhan

Burak Aydoğmuş

Nedim Göktuğ Tabak   
  
Spring Semester 2024–2025

Developer Project Report

# 1. Requirements Analysis

1.1 Aim and Scope

The objective of this project is to develop a **console-based task and reward management system** for children. The system allows **Parents** and **Teachers** to assign tasks, and enables the **Child** to complete those tasks, receive ratings, collect points, and redeem rewards in the form of **wishes**. The system adopts a **gamification-based motivation model** to encourage task completion and responsible behavior in children.

**✳ Scope of the Project:**

* Role-based interactions for **Child**, **Parent**, and **Teacher**.
* A fully **text-command-driven interface** without graphical components.
* **Point and level system** based on task ratings.
* Management of two types of wishes: **Product (WISH1)** and **Activity (WISH2)**.
* Support for **bonus points** and **wish approval constraints** (level, points).
* **File-based input/output** for initialization and execution via Tasks.txt, Wishes.txt, and Commands.txt.
* **Error handling and validation** for incorrect commands and logic violations.

**🔒 Constraints & Assumptions:**

* Single-child operation (multi-user logic is not implemented).
* No persistent user authentication.
* All commands must follow strict input format.
* Application is run entirely through a text file (Commands.txt).

**TO – DO LİST**

|  | **Task** | **Status** |
| --- | --- | --- |
|  | Implement User, Child, Parent, and Teacher classes | ✅ Done |
|  | Create and manage Task class with deadline, rating, and point system | ✅ Done |
|  | Enable child to complete tasks and earn points | ✅ Done |
|  | Allow Parent and Teacher to approve tasks and assign star ratings | ✅ Done |
|  | Implement Wish system with types WISH1 (product) and WISH2 (activity) | ✅ Done |
|  | Enforce level and budget validation for wish approval | ✅ Done |
|  | Implement child level progression based on average task ratings | ✅ Done |
|  | Implement daily and weekly task filtering options | ✅ Done |
|  | Add file-based input/output for Tasks.txt, Wishes.txt, and Commands.txt | ✅ Done |
|  | Handle invalid commands and edge cases with error messages | ✅ Done |

## 1.2 User Roles

**Child:**  
The child is the primary user of the system. They can view tasks (daily/weekly), complete them, and submit for review. Upon approval, the child earns points based on ratings. They can also request wishes (products or activities) which are subject to approval based on level and budget. The child’s level and rating evolve dynamically as tasks are approved.

**Parent:**  
The parent assigns new tasks, evaluates completed ones, and approves or rejects wishes. They can grant bonus points and define required levels for certain rewards. Parents ensure balance between reward motivation and responsible behavior.

**Teacher:**  
The teacher focuses on educational tasks. They assign and approve academic tasks with ratings that impact the child’s progress. Teachers do not manage wishes but influence points and level progression

## 1.3 Typical Scenario

**Child Scenarios**

**1. Viewing Tasks and Planning**  
The child begins by checking their tasks, filtering by day or week. This helps them prioritize responsibilities and manage their time.

**2. Completing Tasks**  
After finishing a task, the child reports it for review. They know that approved tasks bring points and help increase their level.

**3. Submitting Wishes**  
The child can submit a product or activity wish. They learn to check their level and available points before requesting a reward.

**4. Tracking Points and Level**  
The child monitors their earned points and level progression. This motivates them to improve and unlock higher-level wishes.

**5. Reflecting on Results**  
If a wish is rejected due to low level or budget, the child is encouraged to complete more tasks and aim for better ratings.

**Parent Scenarios**

**1. Assigning Tasks**  
The parent adds tasks for the child, deciding the point value and timing. These can be daily chores or personal goals.

**2. Approving Task Completion**  
When a task is marked done, the parent reviews it and gives a star rating. Points are earned based on this rating.

**3. Adding Bonus Points**  
Parents can reward extra effort by adding points to the child’s budget, encouraging consistent responsibility.

**4. Observing Progress**  
Throughout, the parent keeps track of the child’s development, ensuring fair evaluation and healthy motivation.

**Teacher Scenarios**

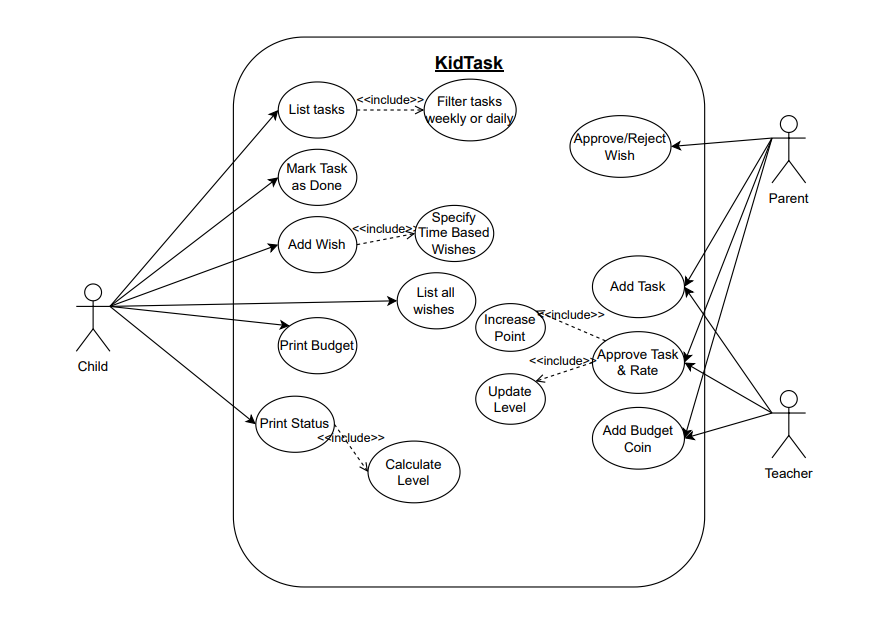
**1. Assigning Academic Tasks**  
The teacher assigns school-related tasks to promote learning and discipline alongside household duties.

**2. Reviewing Work**  
Completed tasks are reviewed and rated fairly. These ratings contribute to the child’s points and level.

**3. Supporting Structure**  
The teacher helps build academic habits and responsibility, balancing the role of the parent in the system.

**4. Encouraging Consistency**  
Even though the teacher cannot approve wishes, their evaluations help the child stay engaged and earn rewards.

USE CASE DİAGRAM



## 1.4 Functional I/O Highlights

**Based on the defined scope, the application is a console-based task and wish management system** designed to handle structured, text-based user input, process commands, and display real-time feedback. The system supports three distinct roles (Child, Parent, Teacher) and all actions are initiated via a command file (Commands.txt).

**🔡 Inputs:**

* All inputs are received from the Commands.txt file.
* Each line in the file represents a single command.
* Commands are structured with defined parameters such as:
  + User role (T for Teacher, F for Parent)
  + Task/wish IDs
  + Descriptions in quotes
  + Date and time in proper format
  + Star ratings and level numbers

**Format Standards:**

* Strings: Enclosed in double quotes "..."
* Date: yyyy-MM-dd
* Time: HH:mm

**💬 Outputs:**  
All outputs are printed to the console immediately after command execution. They include:

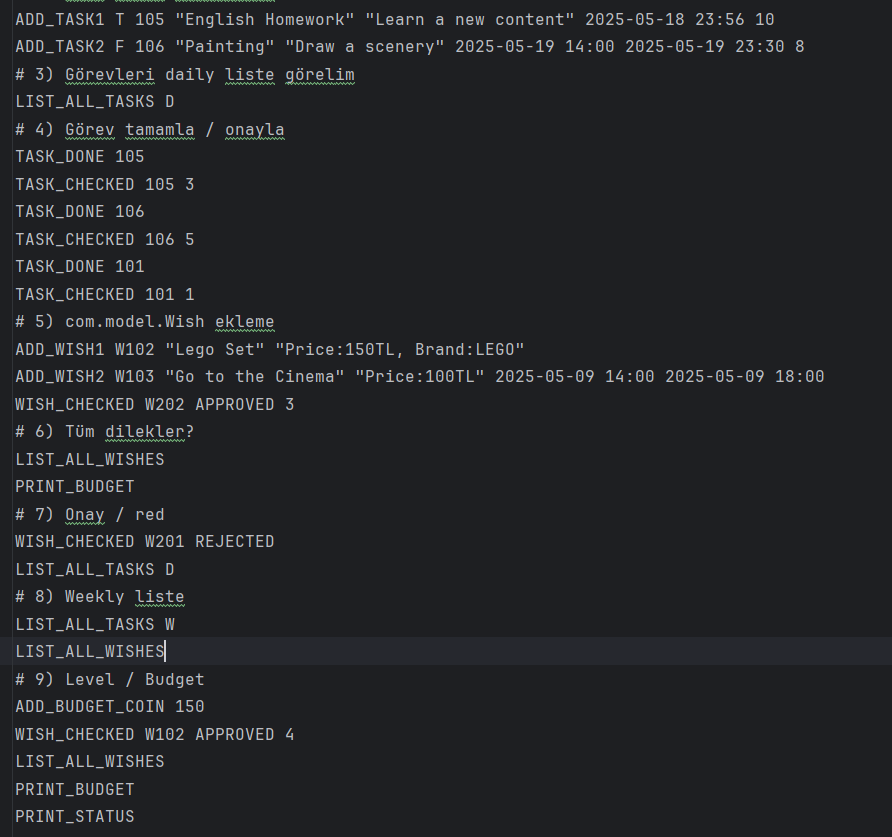
* Confirmation messages
* Error or rejection messages
* Filtered task or wish lists
* Child’s current status (level, points, rating)

Outputs are formatted to be user-readable, without GUI elements.

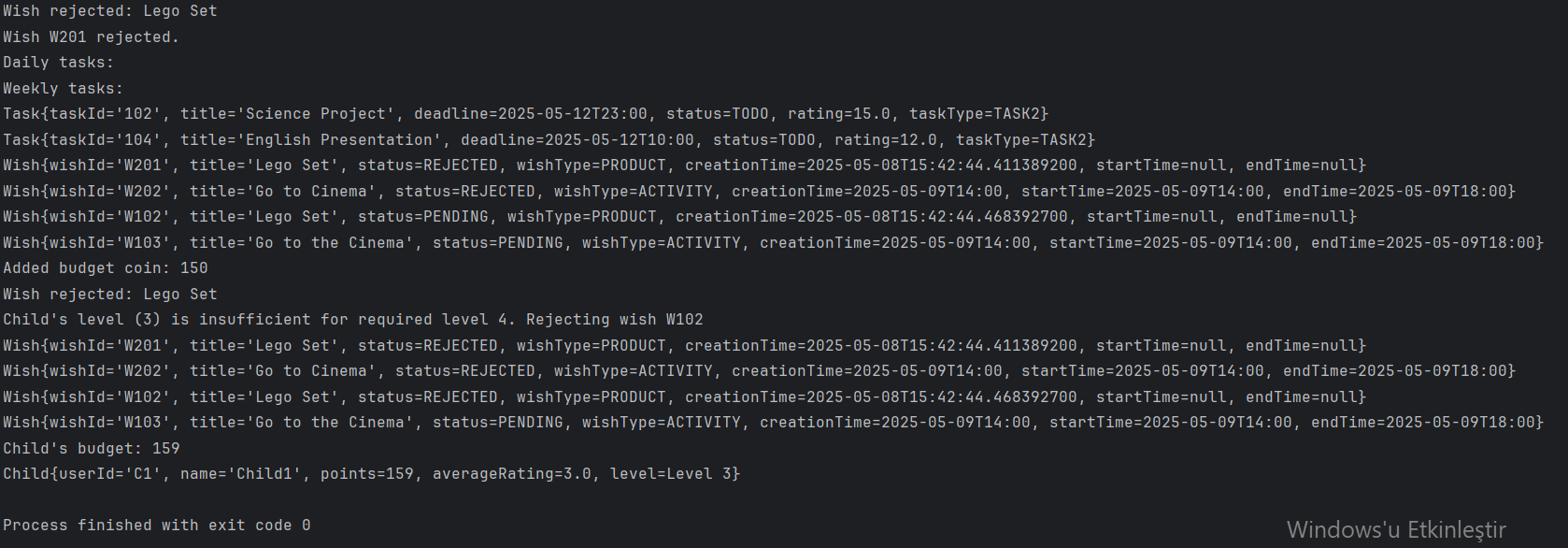
**🧩 Functional Requirements:**

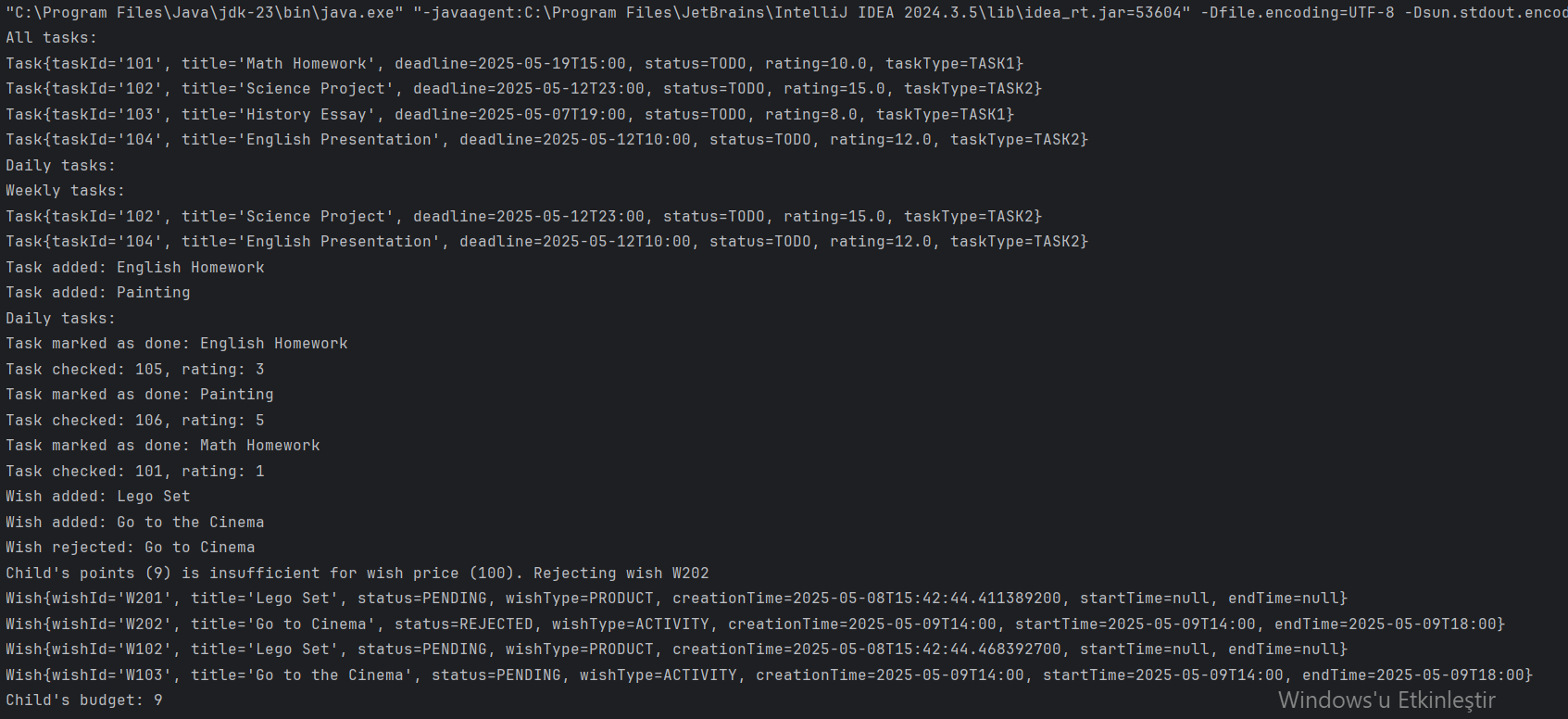
* **FR1 – Add Task**  
  The system allows parents or teachers to add a task with optional start/end time.
* **FR2 – List All Tasks**  
  Children can view all tasks, optionally filtered by D (daily) or W (weekly).
* **FR3 – List All Wishes**  
  Children can list all submitted wishes, regardless of status.
* **FR4 – Mark Task as Done**  
  A child can mark a task as completed.
* **FR5 – Check/Approve Completed Tasks**  
  Teachers or parents can approve a completed task and assign a rating from 1 to 5. Approved tasks grant points and impact the child’s level.
* **FR6 – Add Wish**  
  A child can submit product or activity wishes.
* **FR7 – Check/Approve Wishes**  
  A parent can approve or reject wishes based on level and point balance.
* **FR8 – Add Budget Coin**  
  A parent can grant extra points to a child.
* **FR9 – Print Budget**  
  Displays the child’s current total points (budget).
* **FR10 – Print Status**  
  Displays the child’s current level and average task rating.

**# Example Inputs (Commands.txt)**



# Example Outputs (Console)





# 2. Software Design

## 2.1 Data Structures

This project uses a combination of fundamental Java collections and object-oriented data models to efficiently manage tasks, wishes, and users. Below are the key data structures used and the rationale behind their selection:

**1. ArrayList<Task>**

* **Why chosen:** Allows dynamic storage and ordered access to task objects.
* **Use case:** All tasks created in the system—whether from Tasks.txt or via ADD\_TASK commands—are stored in an ArrayList inside the TaskManager class.
* **Supports:**
  + Easy iteration for task filtering (daily/weekly)
  + Fast addition of new tasks
  + Index-based access for operations like searching or updating by ID

**2. ArrayList<Wish>**

* **Why chosen:** Offers efficient sequential access to wishes and maintains insertion order.
* **Use case:** The WishManager class uses an ArrayList to store all wishes submitted by the child.
* **Supports:**
  + Listing all wishes regardless of status
  + Searching and updating a specific wish by ID
  + Sorting and filtering logic if needed later

**3. Enum Types – TaskType, WishType, TaskStatus, WishStatus**

* **Why chosen:** Enums ensure type safety and restrict values to predefined options.
* **Use case:** Used across the application to clearly represent the type and state of tasks and wishes.
* **Supports:**
  + Consistent validation during task/wish creation and updates
  + Simplifies conditional logic and improves code readability
  + Reduces input-related errors (e.g., invalid statuses)

**4. LocalDateTime / LocalDate / LocalTime**

* **Why chosen:** Java's built-in time API is immutable and robust for handling deadlines and activity durations.
* **Use case:** Tasks and activity-based wishes require deadline or time range fields that are managed using LocalDateTime.
* **Supports:**
  + Date-based filtering (e.g., daily/weekly tasks)
  + Time comparison for validation
  + Future expansion like scheduling

## 2.2 Object‑Oriented Design

This system is structured around core object-oriented design principles, using well-defined classes to represent users, tasks, wishes, and control mechanisms. Responsibilities are clearly distributed across objects to support modularity, readability, and maintainability.

**🧩 Core Classes and Their Responsibilities**

**1. User (Abstract Class)**

* Shared base for all users.
* Fields: id, name, role.
* Abstract behavior implemented by Child, Parent, Teacher.

**2. Child (Extends User)**

* Tracks own progress (points, level, rating).
* Holds:
  + ArrayList<Task> taskList
  + ArrayList<Wish> wishList
* Can mark tasks done, submit wishes.
* Computes average rating and updates level accordingly.

**3. Parent (Extends User)**

* Assigns and reviews tasks.
* Approves or rejects wishes.
* Can give bonus points manually.

**4. Teacher (Extends User)**

* Assigns educational tasks.
* Approves and rates only academic tasks.
* Cannot manage wishes.

**5. Task**

* Represents an individual task.
* Fields: taskId, title, description, deadline, pointValue, rating, status, assignedBy.
* Holds star rating and task lifecycle status.

**6. Wish**

* Represents a reward request.
* Fields: wishId, title, description, wishType, wishStatus, price, requiredLevel, startTime, endTime.
* Deducts points upon approval, tracks approval status.

**7. TaskService**

* Service layer to manage logic for task operations.
* Methods:
  + addTask(...)
  + markTaskAsDone(...)
  + approveTask(...)
  + filterTasksByDayOrWeek(...)
* Decouples task-related logic from UI and entity classes.
* Called from CommandProcessor.

**8. WishService**

* Handles wish-related processes.
* Methods:
  + addWish(...)
  + approveWish(...)
  + listWishes()
* Validates budget and level.
* Deducts points from child upon successful approval.

**9. CommandProcessor**

* Core dispatcher for all commands.
* Reads each parsed line from Commands.txt and routes:
  + Task-related → TaskService
  + Wish-related → WishService
  + Status/print commands → Child, Parent, or direct output.
* Responsible for:
  + Command syntax validation
  + Delegation to correct logic layer
  + Error reporting

**10. FileHandler**

* Reads initial input files:
  + Tasks.txt, Wishes.txt, Commands.txt
* Uses regex and format validation.
* Handles:
  + Parsing logic
  + Line-by-line interpretation
  + Potential write...() methods for saving changes (if implemented)

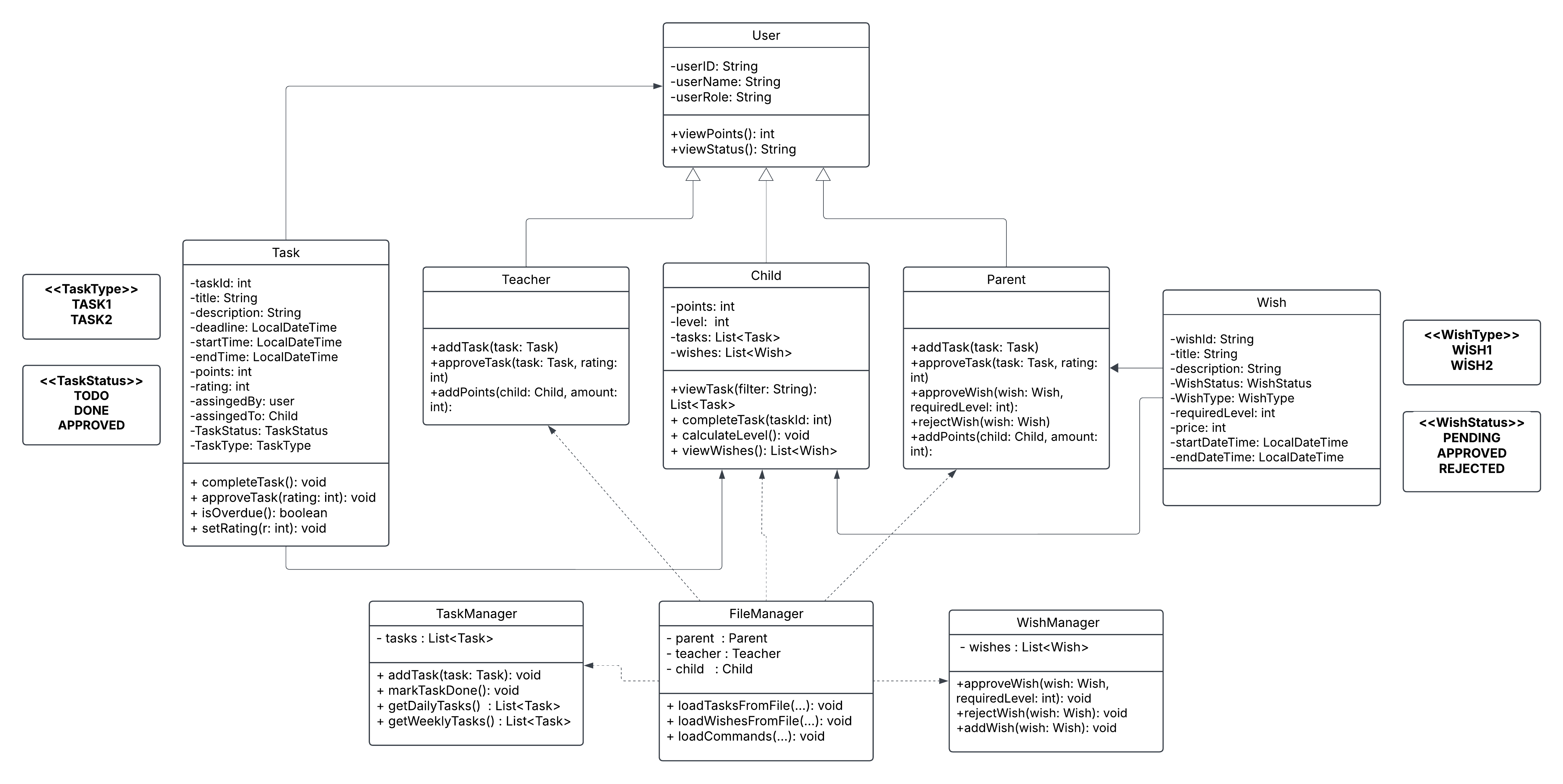
**11. Main**

* Entry point of the program.
* Creates users and initializes CommandProcessor.
* Kicks off the execution flow by reading and dispatching all commands.

**12. Enum Types**

* Used throughout for safety and clarity.
* Includes:
  + TaskStatus { TODO, DONE, APPROVED }
  + WishStatus { PENDING, APPROVED, REJECTED }
  + WishType { WISH1, WISH2 }

CLASS DİAGRAM



2.3 Key Algorithms & Logic Flows

This section outlines the main operational flows of the system, detailing how tasks and wishes are handled, how levels are calculated, and how command processing is structured. Each algorithm follows a modular, service-oriented design pattern, ensuring separation of concerns and scalability.

**🔄 Task Completion and Approval Flow**

**Overview**  
This flow governs how a task is completed by the child, reviewed by the parent or teacher, and rewarded through a point and level system.

**Steps:**

1. **Task Assignment**  
   A task is created by the parent or teacher via TaskService.addTask(...) and assigned to the child.
2. **Mark as Done**  
   The child marks the task as done using TaskService.markTaskAsDone(taskId). The task status is updated to DONE.
3. **Approval and Rating**  
   The parent or teacher approves the task via TaskService.approveTask(taskId, rating), assigning a star rating (1–5). The task status is updated to APPROVED.
4. **Rewarding and Progress Update**  
   Upon approval:
   * Points are awarded to the child based on the task’s point value.
   * The child’s average rating is recalculated.
   * The level is updated based on defined thresholds using updateLevelByRating().

**Level Thresholds:**

| **Average Rating** | **Level** |
| --- | --- |
| < 2.0 | 1 |
| 2.0–2.99 | 2 |
| 3.0–3.99 | 3 |
| ≥ 4.0 | 4 |

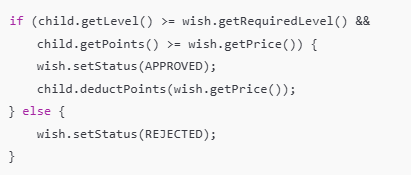
**🎁 Wish Submission and Approval Flow**

**Overview**  
This flow manages how the child submits wishes and how the parent validates them based on eligibility.

**Steps:**

1. **Wish Submission**  
   The child submits a wish using WishService.addWish(...). The wish is added to the list with PENDING status.
2. **Evaluation**  
   The parent checks the wish with WishService.approveWish(wishId, action), verifying:
   * If the child’s level meets the required level.
   * If the child has sufficient points to cover the wish price.
3. **Decision and Outcome**
   * If conditions are satisfied: The wish is approved and the price is deducted.
   * Otherwise: The wish is rejected.

**Example Logic:**



**📊 Level Calculation Algorithm**

**Overview**  
After each approved task, the child’s average rating is updated, and the level is recalculated accordingly.

**Steps:**

* All approved task ratings are collected.
* The average is computed.
* The child’s level is updated using predefined rating ranges (see table above).

**Importance:**  
This ensures that the child's progress is not only based on task quantity but also on the quality (rating), encouraging better performance.

**🧩 Command Processing Flow (CommandProcessor)**

**Overview**  
This flow handles parsing and execution of commands from Commands.txt.

**Steps:**

1. Commands are read using FileHandler.readCommands().
2. Each command is passed to CommandProcessor.processCommand(line).
3. The command type is identified (e.g., ADD\_TASK1, TASK\_DONE, ADD\_WISH1).
4. The command is routed to the appropriate method in:
   * TaskService
   * WishService
   * Child, Parent, or Teacher

**Purpose:**  
This architecture separates command parsing from business logic, improving maintainability and testability.

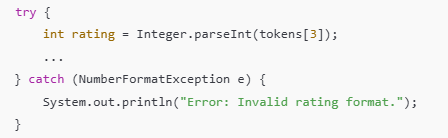
**⚠️ Error Handling & Validation Flow**

**Overview**  
To ensure robust execution, the system incorporates input validation and exception handling mechanisms.

**Checks Include:**

* Missing or malformed parameters (e.g., date, ID, quotes).
* Invalid enum values.
* Logical constraints (e.g., insufficient points or level).
* Exception types:
  + NumberFormatException
  + DateTimeParseException
  + IndexOutOfBoundsException

**Example Handling:**

****

**Outcome:**  
Instead of crashing, the system prints informative error messages and skips invalid lines gracefully.

**3. Implementation**

**🔨 Development Process**

The implementation of the project followed a modular, object-oriented approach, in which core classes were developed and integrated in stages to ensure maintainability and clarity. The process consisted of the following steps:

1. **User Hierarchy Construction**
   * The abstract User class was created to define shared fields and methods.
   * Concrete classes Child, Parent, and Teacher were derived to encapsulate role-specific behaviors.
2. **Domain Models: Task and Wish**
   * The Task class was designed with fields such as taskId, description, deadline, status, pointValue, and rating.
   * The Wish class included fields for wishId, description, price, type, status, and requiredLevel.
3. **Service Layer Implementation**
   * TaskService was implemented to manage task creation, status updates, and approval workflows.
   * WishService handled wish submission, approval logic, and budget validation.
   * These services were designed to decouple business logic from command parsing and user interaction.
4. **Command Processing Engine**
   * A dedicated CommandProcessor class was introduced to parse and dispatch commands from Commands.txt to the appropriate services or users.
   * Commands such as ADD\_TASK, TASK\_DONE, ADD\_WISH, and PRINT\_STATUS were recognized and processed.
5. **File Handling**
   * The FileHandler class managed the reading of initial data from Tasks.txt, Wishes.txt, and Commands.txt.
   * Price values embedded in wish descriptions were extracted using regular expressions.
   * Time and date strings were validated and parsed using Java’s DateTimeFormatter.
6. **Testing and Output**
   * All program output was printed to the console for immediate feedback.
   * A variety of test scenarios were executed to validate correctness, handle edge cases, and ensure stable behavior.

**⚠️ Challenges Faced**

|  |  |  |
| --- | --- | --- |
| **Issue** | **Description** | **Resolution** |
| Incorrect Wish Price Extraction | Price values embedded in wish descriptions (e.g., "Price:150TL") were not parsed correctly. | Converted the string to lowercase and used a regex pattern to extract numeric values. |
| Level Not Updating | After task approval, the child’s level did not reflect updated ratings. | Called updateLevelByRating() after every rating update. |
| Points Not Deducted After Wish Approval | Approved wishes did not subtract the cost from the child’s budget. | Added deductPoints() call within wish approval logic. |
| Malformed Commands Causing Crashes | Missing parameters or incorrect formats in Commands.txt caused runtime errors. | Wrapped command parsing logic in try-catch blocks and added meaningful error messages. |
| Time Parsing Failures | Invalid date/time strings were ignored silently. | Enforced strict DateTimeFormatter validation and printed errors for invalid formats. |
| Wish Approved at Wrong Level | Wishes were approved even if the child was below the required level. | Added level-check condition before changing wish status. |

**🧠 Support and Resources**

The following tools and resources were consulted and utilized during development:

* **Stack Overflow**  
  For resolving regex issues and exception handling patterns in Java.
* **ChatGPT**  
  Used for validating class design decisions, handling edge case logic, and improving error messages.
* **Official Java Documentation**  
  For reference on ArrayList, LocalDateTime, Enum, and file I/O utilities.
* **Manual Testing**  
  Numerous test cases were executed by crafting sample Commands.txt inputs, simulating real-world use.

# 4.Testing & Error Handling

**✅ Test Cases and Validation**

To ensure the reliability and correctness of the system, a series of manual test cases were executed. These tests focused on validating core functionalities such as task workflows, wish approvals, budget control, and error responses.

Below is a summary of representative test cases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case Description | Input (Green) | Expected Output | Actual Output | Status |
| Add a valid daily task | ADD\_TASK1 F 101 "Homework" "Math pages 1-10" 2025-05-01 14:00 10 | Task 101 added successfully. | Task 101 added successfully. | Passed |
| Mark task as done | TASK\_DONE 101 | Task marked done. | Task marked done. | Passed |
| Approve task with rating | TASK\_CHECKED 101 5 | Task approved. 5 stars. 10 points awarded. | Task approved. 10 points awarded. | Passed |
| Submit a valid wish | ADD\_WISH1 W301 "LEGO Set" "Price:150TL, Brand:LEGO" | Wish added successfully. | Wish added successfully. | Passed |
| Approve wish with valid level and points | WISH\_CHECKED W301 APPROVED 2 | Wish approved. Points deducted. | Wish approved. 150 points deducted. | Passed |
| Reject wish - insufficient points | Child points = 50. Wish price = 150. | Wish rejected (insufficient budget). | Wish rejected. | Passed |
| Reject wish - insufficient level | Child level = 1. Required level = 3. | Wish rejected (level too low). | Wish rejected. | Passed |
| Invalid command format | ADD\_TASK1 F 102 "Only Title" | Error: Invalid command format | Error: Invalid command format | Passed |
| Print child status | PRINT\_STATUS | Displays level, rating, and points. | Level: 2, Rating: 4.5, Points: 90 | Passed |
| Submit activity-based wish | ADD\_WISH2 W401 "Movie Night" "Price:100TL" 2025-05-10 18:00 2025-05-10 20:00 | Activity wish added successfully. | Wish W401 added successfully. | Passed |

**✅ Bugs Encountered and Solutions**

* **Wish price was always zero**  
  The price field in wishes always returned 0. This happened because the price value wasn’t correctly extracted from the description text. The issue was resolved by converting the text to lowercase and using regex to isolate the numeric value.
* **Child level didn’t update after task approval**  
  Even after tasks were approved and rated, the child’s level didn’t change. This was because the level update method was never triggered. It was fixed by calling updateLevelByRating() after task approval.
* **Approved wishes didn’t deduct points**  
  Approved wishes were not affecting the child’s point balance. The deduction logic (deductPoints()) was missing in the approval flow. It was added to ensure points are subtracted once a wish is approved.
* **Malformed commands caused system crashes**  
  Improperly formatted commands (e.g., missing parameters or quotation marks) led to runtime exceptions. The issue was solved by wrapping command parsing in try-catch blocks and showing clear error messages.
* **Date/time parsing failed silently**  
  Some inputs with invalid or misformatted dates/times didn’t trigger visible errors. The inputs were trimmed, and strict format validation (yyyy-MM-dd, HH:mm) was enforced to prevent silent failures.
* **Wishes were approved even at incorrect levels**  
  Wishes that required a certain level were being approved regardless of the child’s level. This logic flaw was corrected by adding a proper level check before changing the wish status to APPROVED.

**🧠 Support and Resources for Bug Fixes**

* **Stack Overflow**: Used for fixing regex expressions and date parsing errors with LocalDateTime.
* **ChatGPT**: Helped in designing clean class structures and fixing logic bugs in rating and level tracking.
* **Java Docs**: Referred for collection handling, exception classes, and enum usage.
* **Manual Testing**: Tested dozens of edge cases using crafted Commands.txt scenarios.
* **Own Debug Logs**: Used System.out.println extensively to trace logic failures and confirm fixes

### 🔍 Observations from Testing

* The system passed all functional and boundary tests when provided with valid input.
* All errors were caught and reported with clear, user-friendly messages.
* No command caused system termination, even when malformed or invalid.

# 5. Version Control

**GitHub Repository Link**: <https://github.com/OgzhnTarhn/Seng272>

We created a GitHub repository to store and collaborate on the best version of our project. Each team member was added as a collaborator, allowing us to work together efficiently. Throughout the development process, we regularly made commits to fix bugs, resolve errors, and improve the overall functionality of the code. By continuously reviewing and refining our contributions, we were able to enhance the quality of the project and ultimately achieve the most stable and optimized version

**Commit Messages:**  
Descriptive and consistent commit messages were used to clearly document each change:

* feat: implement ADD\_TASK1 and ADD\_TASK2 logic
* fix: correct wish price parsing issue
* refactor: restructure FileManager for cleaner input handling

**📁 How the Project Was Split and Managed**  
Work was divided logically between components:

* Task, Wish, and User classes were developed first to define the structure.
* Manager classes (TaskManager, WishManager) were added to handle collections.
* File operations and parsing were separated into a dedicated FileManager class.
* Commands and test data were created and modified in Commands.txt for testing purposes.
* Each new feature or bug fix was committed separately to allow easy rollback if necessary.

# 6. Discussion / Reflection

We held both face-to-face and online meetings throughout the development process. By discussing ideas and sharing feedback with one another, we were able to improve the quality of the project collaboratively. During this time, our team leader provided us with detailed guidance on what needed to be done, helping us stay on track and enhance the overall outcome. By combining the strongest and most effective parts of each team member’s code, we successfully created a highly refined and improved version of the project.