Exam Management System

Application Design Document

Ofek Nov

Mor Abo

Roi Tiefenbrunn

Idan Aharoni

# Contents

1. [Use Cases](#_bookmark0) 3
2. [System Architecture](#_bookmark1) 4
3. [Data Model](#_bookmark2) 5
   1. [Description of Data Objects](#_bookmark3) 5
   2. [Data Objects Relationships](#_bookmark4) 5
   3. [Databases](#_bookmark5) 6
4. [Behavioral Analysis](#_bookmark6) 7
   1. [Sequence Diagrams](#_bookmark7) 7
   2. [Events](#_bookmark8) 7
   3. [States](#_bookmark9) 7
5. [Object-Oriented Analysis](#_bookmark10) 9
   1. [Class Diagrams](#_bookmark11) 9
   2. [Class Description](#_bookmark12) 9
   3. [Packages](#_bookmark13) 9
   4. [Unit Testing](#_bookmark14) 10
6. [User Interface Draft](#_bookmark15) 11

**Chapter 1**

# Use Cases

1. **Manage Courses**

* **Description:** This use case allows administrators to create courses within the system.
* **Actor:** Administrator
* **Preconditions:** User must be logged in as an administrator.
* **Basic Flow:**

1. Administrator accesses the course management section of the system.

2. Administrator views a list of existing courses.

3. Administrator can add a new course, and ask a specific user to be the course

admin.

4. The course admin that was chosen gets a request to become the course admin, he

must accept or deny the request.

* **Alternate Flows:**

If the course id already exists in the system or the user that the admin chose to be the course admin doesn’t exist, the course is not created, and an appropriate error message appears to the admin.

1. **Create and Assign Tasks**

* **Description:** This use case enables users to create tasks and assign them to specific individuals or roles.
* **Actor:** Course admin, TA, Grader
* **Preconditions:** User must be logged in with appropriate permissions.
* **Basic Flow:**

1. User accesses the task creation section of the system.

2. User creates a new task by providing a description, priority, and

optionally assigning it to a user or role.

3. User saves the task, and it becomes available for assignment or completion.

* **Alternate Flows:**

If the task is assigned to a role, the system distributes it among users with that role based on predefined algorithms.

1. **Manage Questions**

* **Description:** This use case allows users to manage questions within the system.
* **Actor:** Course admin, TA
* **Preconditions:** User must be logged in as a Course admin or TA.
* **Basic Flow:**

1. User accesses the questions management section of a course.

2. User creates new content, such as questions, stems, meta-questions,

and appendices, organized by subjects and keywords.

3. User edits, deletes, or validates existing content as needed.

* **Alternate Flows:**

Other users on the course need to review and provide feedback on proposed content changes before finalizing them.

1. **Generate Exams**

* **Description:** This use case allows users to generate exams based on the available content.
* **Actor:** Course admin
* **Preconditions:** User must be logged in as a course admin and the course must have defined exam properties such as subjects, number of questions and grading system.
* **Basic Flow:**

1. User accesses the exam creation section of the system.

2. User selects to create an exam and provides cause.

3. System generates the exam using LaTeX-based templates and exports it to a PDF

file and a Word file.

* **Alternate Flows:**

If the user wants to preview the exam before finalizing, he can review and make some adjustments as needed.

1. **Finishing a Task**

* **Description:** This use case allows users to finish a task that is assigned to them.
* **Actor:** Course admin, TA, Grader
* **Preconditions:** User must be logged in and a task is assigned to him.
* **Basic Flow:**

1. User accesses the tasks section of the system.

2. User selects a task to view.

3. The user answers the task and submits his answer.

* **Alternate Flows:**

If the user doesn’t have any assigned tasks, he can view all tasks that are suggested to his type of role and assigns himself one.

**Chapter 2**

# System Architecture

1. **Client-Side Components:**

* **Web Browser:**
  + **Location:** Installed on user devices (client machines).
  + **Functionality:** Renders the user interface provided by the frontend framework and interacts with the backend server via rest-API requests.

1. **Frontend Framework:**

* **React Framework:**
  + **Location:** Bundled as JavaScript files deployed to a web server.
  + **Functionality:** Provides the structure and components for building the user interface of the WMS, including role-based dashboards and content creation forms.

1. **Backend Server:**

* **Node.js HTTP Server\\Restify:**
  + **Location:** Deployed on a dedicated server or cloud platform.
  + **Functionality:** Hosts the server-side application logic, handling authentication, authorization, task distribution, content management, and communication with the database.

1. **Database:**

* **PostgreSQL Database:**
  + **Location:** Hosted on a separate database server.
  + **Functionality:** Stores and manages system data, including user information, tasks, content, and version control.

1. **Authentication and Authorization:**

* **Authentication Service:**
  + **Location:** Part of the backend server.
  + **Functionality:** Verifies user identities, issues authentication tokens (e.g., JWT), and manages user sessions.
* **Authorization Middleware:**
  + **Location:** Integrated into the backend server.
  + **Functionality:** Enforces access control based on user roles and permissions defined in the system, ensuring that users can only access authorized resources.

1. **Task Distribution and Management:**

* **Task Distribution Module:**
  + **Location:** Implemented within the backend server.
  + **Functionality:** Implements algorithms for task assignment and distribution among users based on workload, priority, and expertise.
* **Task Management Service:**
  + **Location:** part of the backend server.
  + **Functionality:** Handles CRUD operations for tasks, including creation, assignment, update, and deletion.

1. **Content Management:**

* **Content Storage:**
  + **Location:** Integrated with the backend server.
  + **Functionality:** Stores various types of content, including questions, stems, meta-questions, and solutions.
* **Content Management Service:**
  + **Location:** Deployed within the backend server.
  + **Functionality:** Implements functionalities for creating, editing, and deleting content items, as well as version control.

1. **LaTeX Processing:**

* **LaTeX Engine:**
  + **Location:** Part of the backend server.
  + **Functionality:** Converts LaTeX input into printable documents such as exams, keys, and solutions.
* **LaTeX Processing Service:**
  + **Location:** Integrated with the backend server.
  + **Functionality:** Generates LaTeX output based on user inputs, ensuring flexibility and customization for exam creation.

**Chapter 3**

# Data Model

This chapter describes the main information data domain of the application. Objects are real-world entities that have counterparts within the system. Associated with each object is a set of attributes and functions. Associating the functions will be handled in Chapter [5.](#_bookmark10) The remainder of this chapter is devoted to analyzing the attributes.

## Description of Data Objects

1. **User**

* **Attributes:**
  + **UserId** (int): Unique identifier for the user.
  + **Username** (string): User's username for authentication.
  + **Role** (enum): User's role in the system (e.g., instructor, TA, grader).

1. **Task:**

* **Attributes:**
  + **TaskId** (int): Unique identifier for the task.
  + **Description** (string): Description of the task.
  + **Priority** (int): Priority level of the task.
  + **AssignedTo** (User): Reference to the user to whom the task is assigned.

1. **Content:**

* **Attributes:**
  + **ContentId** (int): Unique identifier for the content.
  + **Subject** (string): Subject or topic of the content.
  + **keywords** (List<string>): List of keywords associated with the content.

## Data Objects RelationshipsA screenshot of a computer Description automatically generated

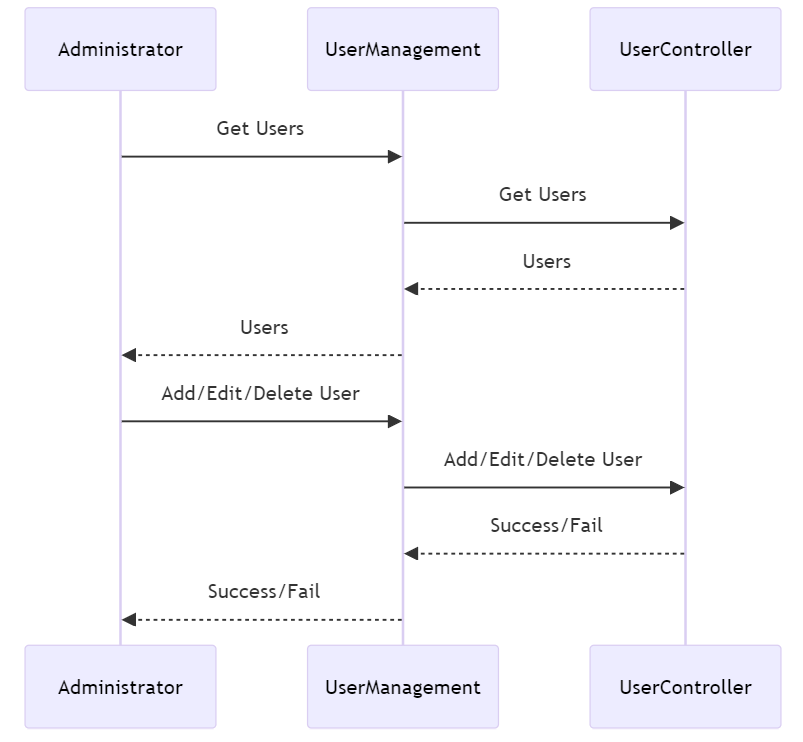
**Chapter 4**

# Behavioral Analysis

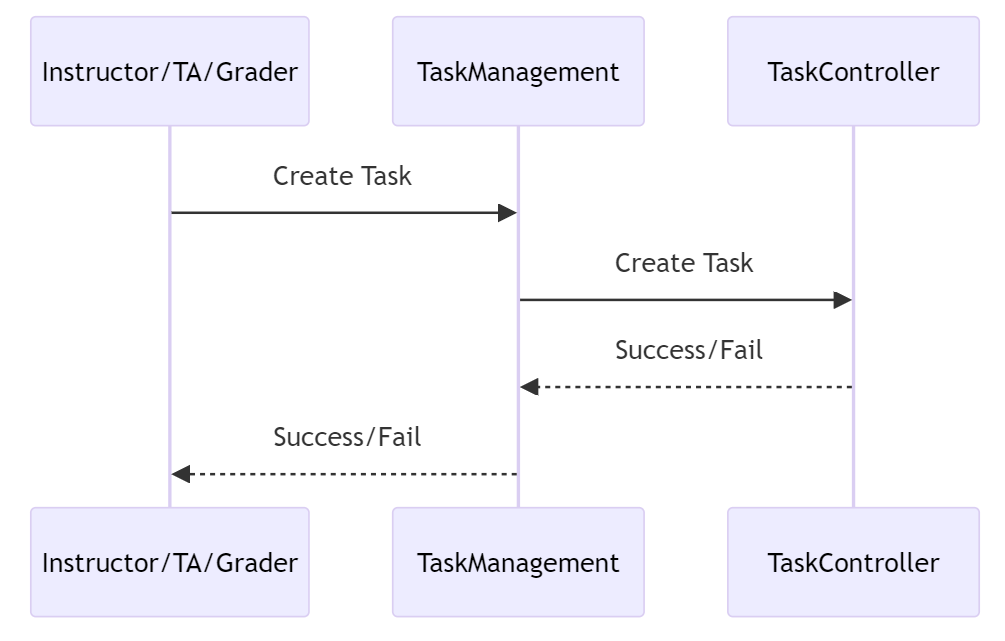
This chapter describes the control flow of our system.

## Sequence Diagrams

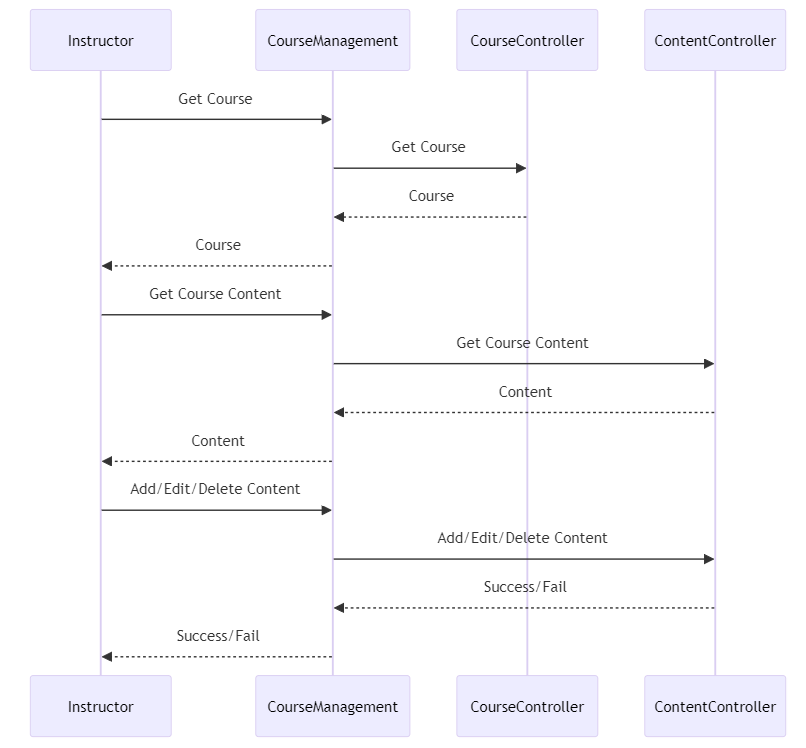
1. **Manage User**



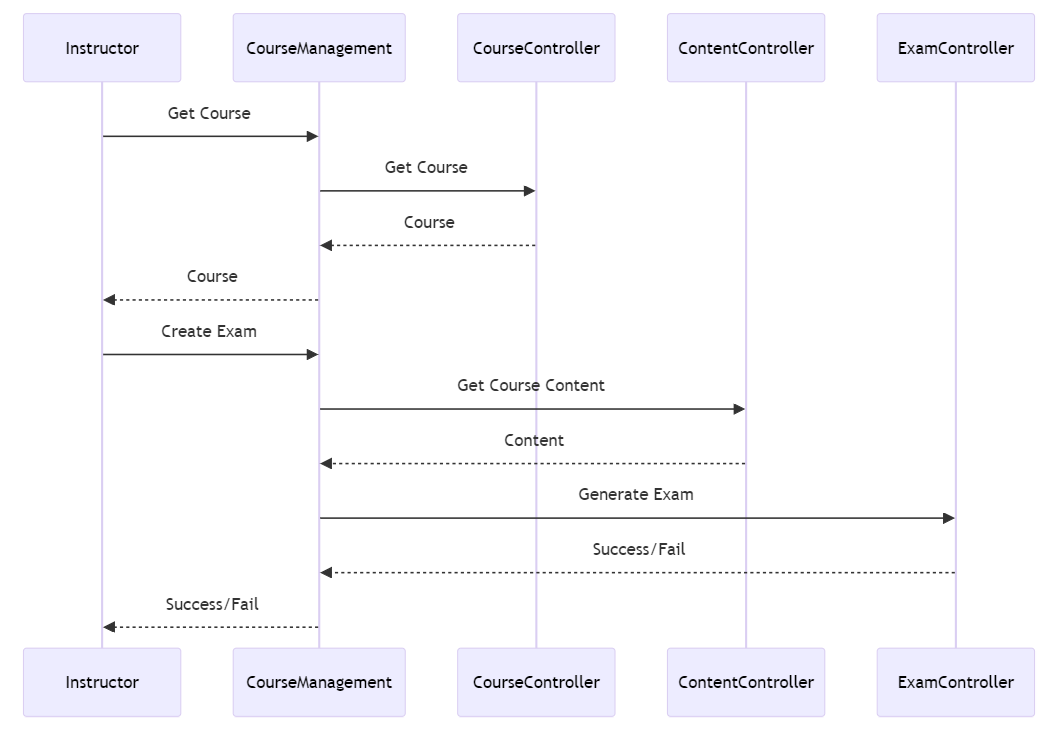
1. **Create and Assign Tasks**

****

1. **Manage Content**

****

1. **Generate Exams**

****

## Events

Some software systems are best described and understood in terms of the events that control the behavior of these systems, i.e., what “happens”, and how the system reacts in response. If your system fits this category, then detail and describe the events that govern the behavior of your system.

**Example Scenario** Upon receiving a shutdown signal, the program should write a message to the system logger, save all open buffers, close all open files, including user files, system files, and configuration files, and terminate.

## States

Some software systems are best described and understood in terms of a state- machine formalism such as *state charts*. If your system fits this category, then detail and describe the relevant states.

Most systems have a few trivial states, such as logged in / logged out, or online / offline. Specify the main states of your system even if full state/events analysis for the entire system is not appropriate.

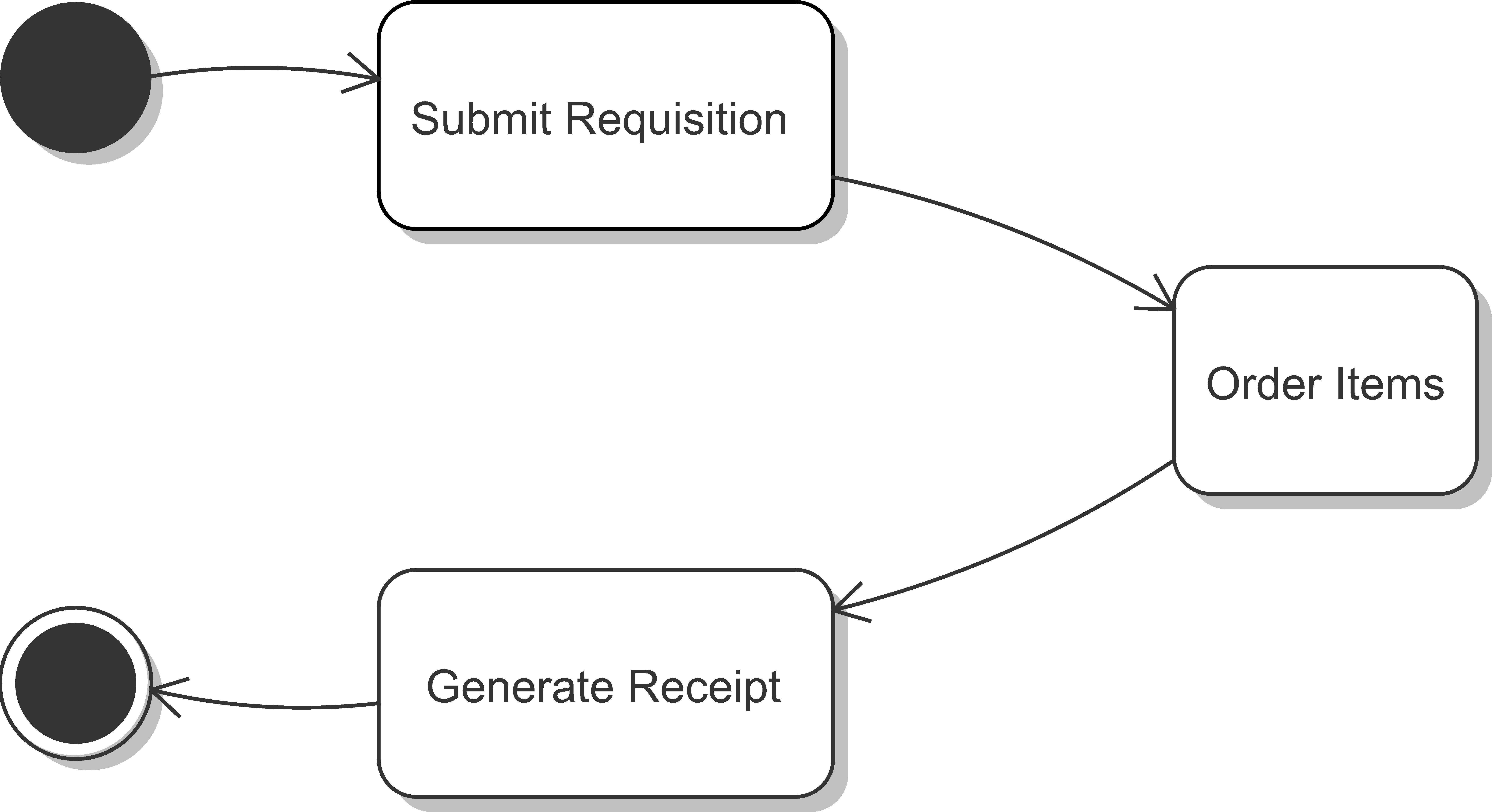


Figure 4.1: Handing a purchase request — An example of a state-driven logic

**Chapter 5**

# Object-Oriented Analysis

Combine your data and behavioral analysis into classes, abstract classes, inter- faces, and packages. Make use of *design patterns* to improve the structure of your code and the quality of your design.

## Class Diagrams

Describe the classes in your system. For each class, mention attributes, opera- tions, and relations to other classes.

## Class Description

For the main classes described in Section [5.1,](#_bookmark11) provide

* + - A textual description of its responsibilities

For the main methods, list any invariants, pre-conditions, post-conditions (e.g., ID is never 0, *age >* 0, etc)

*•*

* + - Any implementation hints (e.g., *method foo should be declared synchronized* ) You may want to use the Object-Constraint Language (OCL) wherever possible,

in order to keep the description concise and precise.

## Packages

Any large software system will consist of many classes, which are best organized into a hierarchy of packages. Describe the package hierarchy of your software system.

## Unit Testing

Design unit tests for each class. When designing the tests, try not to think about actual input and output values and concrete functions, but rather about properties of your classes and methods and their invariants.

Go over Section [5.2](#_bookmark12) and make sure you test the invariants, pre-conditions, post-conditions of your methods. Make sure you test any boundary conditions.

**Chapter 6**

# User Interface Draft

Describe the main user interfaces of your system. Prepare simple drawings of the main screens. Keep the discussion at the level of inputs & outputs, rather than going into aesthetic details.

**Chapter 7**

# Testing

Describe how you plan on testing the entire system. For each of the non- functional constraints listed in the ARD, describe how you intend to test that your software meets that constraint. Describe your actual tests. For each sub- system, describe the tests for this subsystem, the steps to conduct each test, the expected results and the actual results obtained.