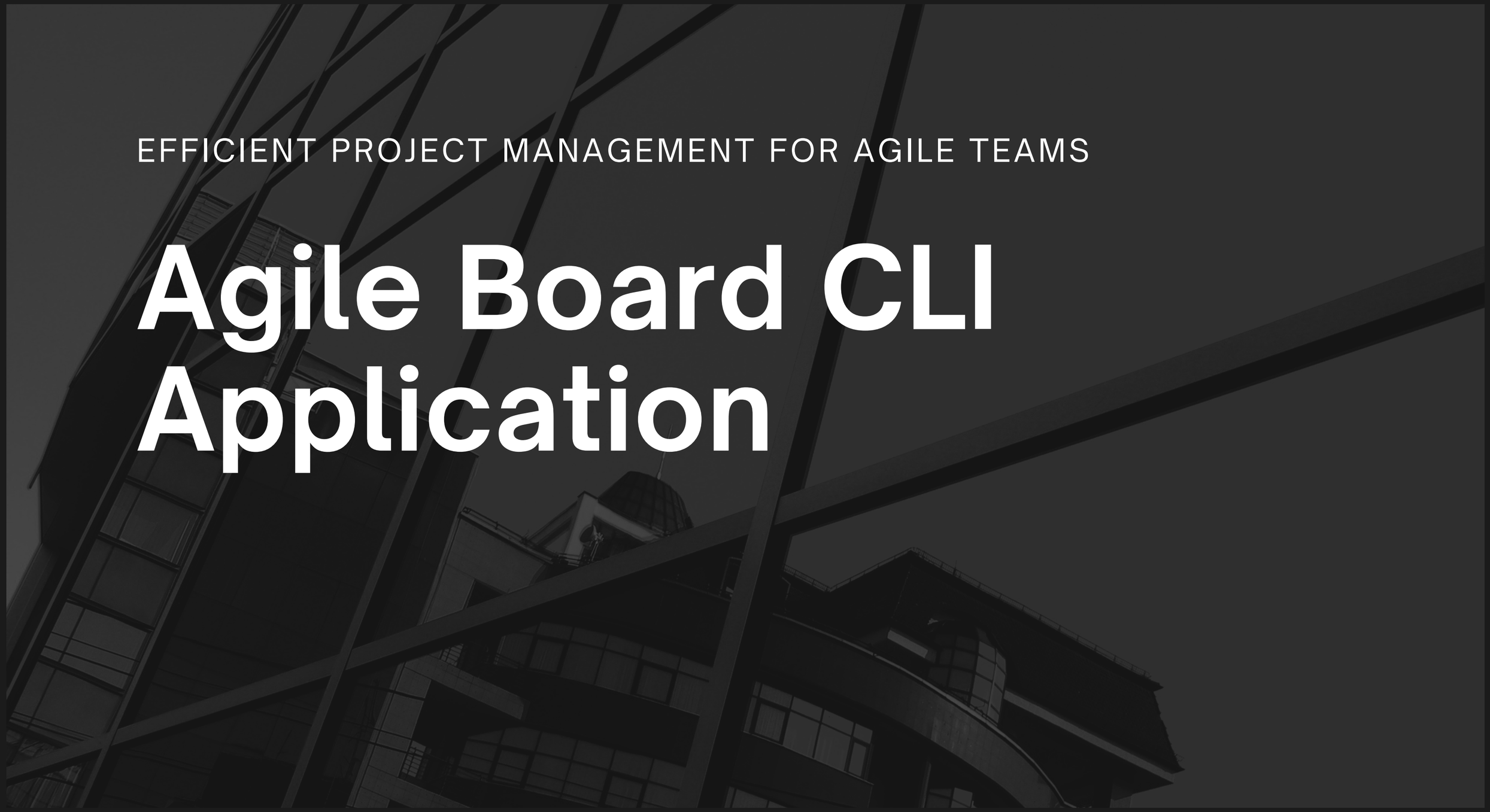


EFFICIENT PROJECT MANAGEMENT FOR AGILE TEAMS

Agile Board CLI Application

TABLE OF CONTENTS

INTRODUCTION

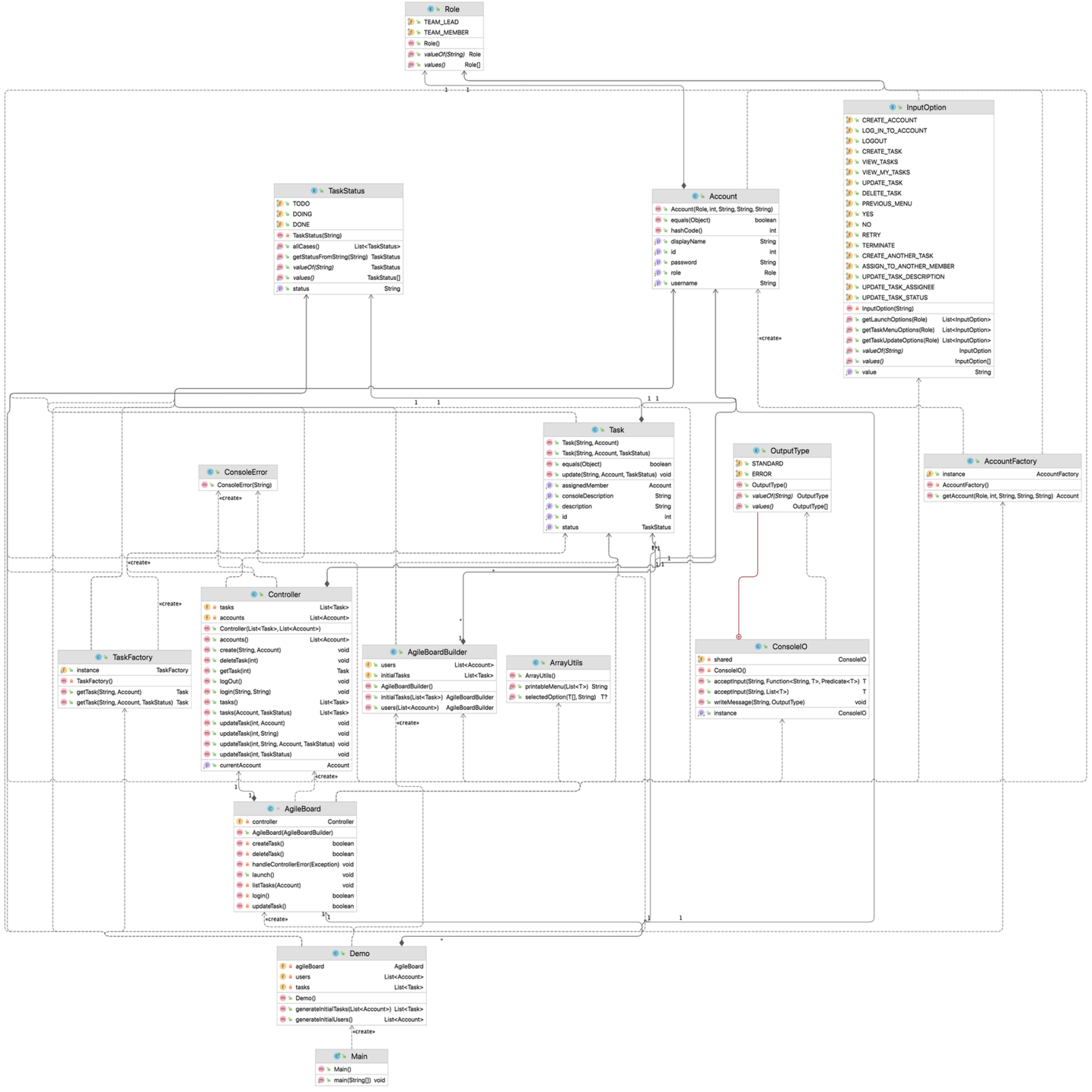
The Agile Board CLI is a command-line interface application designed for streamlined project management using Agile methodologies. It offers a concise and efficient way to create, manage, and track tasks on an Agile board, allowing teams to easily collaborate and stay organized. With intuitive commands, it enables users to add, prioritize, and update tasks effortlessly, fostering agile development practices in a text-based environment. Enhance your team's agility and productivity with the Agile Board CLI.

THE TEAM

NIMISH SHARMA

OLAYINKA OLASUNKANMI

UML DIAGRAM


Role
TEAM_LEAD
TEAM_MEMBER
Role()
valueOf(String) Role
values() Role[]
InputOption
CREATE_ACCOUNT
LOG_IN_TO_ACCOUNT
LOGOUT
CREATE_TASK
VIEW_TASKS
VIEW_MY_TASKS
UPDATE_TASK
DELETE_TASK
PREVIOUS_MENU
YES
NO
RETRY
TERMINATE
CREATE_ANOTHER_TASK
ASSIGN_TO_ANOTHER_MEMBER
UPDATE_TASK_DESCRIPTION
UPDATE_TASK_ASSIGNEE
UPDATE_TASK_STATUS
InputOption(String)
getLaunchOptions(Role) List<InputOption>
getTaskMenuOptions(Role) List<InputOption>
getTaskUpdateOptions(Role) List<InputOption>
valueOf(String) InputOption
values() InputOption[]
value String
TaskStatus
TODO
DOING
DONE
TaskStatus(String)
allCases() List<TaskStatus>
getStatusFromString(String) TaskStatus
valueOf(String) TaskStatus
values() TaskStatus[]
status String
Account
Account(Role, int, String, String, String)
equals(Object) boolean
hashCode() int
displayName String
id int
password String
role Role
username String
«create»
Task
Task(String, Account)
Task(String, Account, TaskStatus)
equals(Object) boolean
update(String, Account, TaskStatus) void
assignedMember Account
consoleDescription String
description String
id int
status TaskStatus
OutputType
STANDARD
ERROR
OutputType()
valueOf(String) OutputType
values() OutputType[]
AccountFactory
instance AccountFactory
AccountFactory()
getAccount(Role, int, String, String, String) Account
ConsoleError
ConsoleError(String)
«create»
Controller
tasks List<Task>
accounts List<Account>
Controller(List<Task>, List<Account>)
accounts() List<Account>
create(String, Account) void
deleteTask(int) void
getTask(int) Task
logOut() void
login(String, String) void
tasks() List<Task>
tasks(Account, TaskStatus) List<Task>
updateTask(int, Account) void
updateTask(int, String) void
updateTask(int, String, Account, TaskStatus) void
updateTask(int, TaskStatus) void
currentAccount Account
TaskFactory
instance TaskFactory
TaskFactory()
getTask(String, Account) Task
getTask(String, Account, TaskStatus) Task
AgileBoardBuilder
users List<Account>
initialTasks List<Task>
AgileBoardBuilder()
initialTasks(List<Task>) AgileBoardBuilder
users(List<Account>) AgileBoardBuilder
ArrayUtils
ArrayUtils()
printableMenu(List<T>) String
selectedOption(T[], String) T?
ConsoleIO
shared ConsoleIO
ConsoleIO()
acceptInput(String, Function<String, T>, Predicate<T>) T
acceptInput(String, List<T>) T
writeMessage(String, OutputType) void
instance ConsoleIO
AgileBoard
controller Controller
AgileBoard(AgileBoardBuilder)
createTask() boolean
deleteTask() boolean
handleControllerError(Exception) void
launch() void
listTasks(Account) void
login() boolean
updateTask() boolean
Demo
agileBoard AgileBoard
users List<Account>
tasks List<Task>
Demo()
generateInitialTasks(List<Account>) List<Task>
generateInitialUsers() List<Account>
Main
Main()
main(String[]) void

Key Features

USER ACCOUNT

User account should have:

- role: ENUM TeamLeader/TeamMember
- id: int
- username: String
- password: String
- displayName: String

TASK

Task should have the following:

- description: String
- status: ENUM TODO/DOING/DONE
- id: int
- assignedMember: Account

TEAM LEADER ROLE

A team lead should be able to:

- Create tasks
- View all tasks
- Delete tasks
- Update task description and assignedMember

TEAM MEMBER ROLE

A team member should be able to:

- view tasks assigned to them
- update tasks status

Design Patterns

MVC

he Model-View-Controller pattern is used to separate the application into three interconnected components to achieve modularity and maintainability. This is used in the entire project design structure.

Singleton Factory

This combine singleton and factory pattern to the factory class has only one instance while producing a factory that manages object creation. This is used in the account and task and creation.

Design Patterns

Facade

This structural design pattern provides a simplified and unified interface to a set of interfaces in a subsystem, making it easier to use and understand. This is used in ConsoleIO class using imported java.utils Interfaces.

Builder

The creational design pattern separates the construction of a complex object from its representation, allowing the step-by-step creation of different configurations of the object using the same construction process. This is used in initializing and building the agile board.

FUTURE IMPROVEMENTS

These are planned or potential future improvements showing our commitment to continuous improvement and staying aligned with evolving Agile practices

Add a GUI for easy use and visualization of application compared to a CLI

GUI

Upgrade project to make use of a RDB and ORM framework to store accounts and tasks data

DATABASE

Add backend controller to create new user accounts for different type of roles

BACKEND

QUESTIONS

The floor is open to any questions, challenges, or ideas you would to suggest

Thank you.