ServiceLayer

UserService
+ Register(email: String, Password:String,):Json
+ DeleteUser(user: User): Json
+ Login(email: string, password:String): Json
+ Logout(user: User): Json
+ SetPassword(user: User, old: String, new: string): Json
+ SetEmail(user: User, new: string): Json

BoardsControllerService
+ AddBoard (email: string, name: string):Json
+ RemoveBoard (email: string, name: string): Json
+ GetAllTasksByState(user:User, state: string): Json

BoardsControllerService
+ AddTask (string email, string boardName, string title, string description, string dueDate): Json
+ RemoveTask (string email, string boardTitle, int taskId): Json
+ LimitColumn(string email, string boardName, int columnOrdinal, int limit)
+ GetColumnLimit(string email, string boardName, int columnOrdinal)
+ GetColumnName(string email, string boardName, int columnOrdinal)
+ GetColumn(string email, string boardName, int columnOrdinal)

Response
+ message(): string
+ operationState: bool

JsonController
+ Serialize<T>(T obj): Json
+ Deserialize<T>(string json): T

BoardsControllerService
+ AdvanceTask(string email, string boardName, int columnOrdinal, int taskId): Json
+ UpdateTaskTitle (string email, string boardName, int columnOrdinal, int taskId, string title): Json
+ UpdateTaskDueDate (string email, string boardName, int columnOrdinal, int taskId, string dueDate): Json
+ UpdateTaskDescription (string email, string boardName, int columnOrdinal, int taskId, string description): Json

1

Dependency

BusinessLayer

UserController
- userData: UserData
- static MIN_PASS_LENGTH: int
- static MAX_PASS_LENGTH: int

+ Register(email: string, Password:String):void
+ DeleteUser(user: User):void
+ LogIn(email: string, password: string): void
+ LogOut(user: User): void
+ SetPassword(user: User, oldP: string, newP: string): void
+ SetEmail(user: User, newE:string): void
+ SearchUser(email: string): User
- IsLegalPassword(pass: string)

BoardController
- userData: UserData

+ CreateBoard (user:User, title: string):void
+ DeleteBoard (user:User, title: string): void
+ GetAllTasksByState(user:User, state:enum): LinkedList<Task>
+ GetBoards(user: User): LinkedList<Boards>
+ SearchBoard(user: User, title: string): Board

Use

Use

UserData
- BinaryTree<string, DataUnit> tree;
- HashSet<string> loggedIn

SearchUser(string email): User
AddUser(string email, string password): User
RemoveUser(string email): void
ContainsUser(string email): bool
UserLoggedInStatus(string email): bool
SetLoggedIn(string email): void
SetLoggedOut(string email): void
GetAllBoards(string email): LinkedList<Board>
AddBoard(string email, string title): Board
RemoveBoard(string email, string title): void

User Implements IComperable
- email: string
- password: string
- hasher: PasswordHash

+ SetPassword(newP: string): void
+ SetEmail(newE: string): void
+ GetEmail(): string
+ CheckPasswordMatch(pass: String): bool
+ CompareTo(obj: object): int

Use

Use

Board
- title: String
- backlog: LinkedList<Task>
- inProgress: LinkedList<Task>
- done: LinkedList<Task>

+ Title: String (getter & setter)
+ BackLog: LinkedList<Task> (getter only)
+ InProgress: LinkedList<Task> (getter only)
+ Done: LinkedList<Task> (getter only)

+ GetTasksByType(type: enum): LinkedList<Task>
+ AddTask (title: String, dueDate: Date, description: String): Task
+ RemoveTask (title: String): void
+ AdvanceTask(title: String): void
+ SearchTask(title: String): Task

PasswordHash
- basePrime: long
- primes: long[]
- PRIME_LOWER_BOUND: long
- PRIME_UPPER_BOUND: long

+ Hash(s: string): string
- GeneratePrimes(count: int, lowerBound: long, upperBound: long): long[]
- inflate(s: string): string

Private class DataUnit
+ User: User { get; set; }
+ LinkedList<Board> : Boards { get; set; }

Inner class

BinaryTree<Key,Data>
root: BinaryTreeNode

+ Add(T element): BinaryTreeNode
+ Remove(T element): void
+ Contains(T element): bool
+ IsEmpty(): bool
+ Search(T element): BinaryTreeNode

Inner class

Task
- title: String
- description: String
- creationTime: Date
- dueDate: Date
- state: enum
- descriptionCharCap: bool
- DESCRIPTION_CHAR_CAP: int

+ Title: string (getter & setter)
+ CreationTime: Date (getter only)
+ DueDate: Date (getter & setter)
+ Description: string (getter & setter)
+ State: enum (getter & setter)
+ CharCap: bool (getter & setter)

Private class BinaryTreeNode
- key: Key
- data: Data
- left: BinaryTreeNode
- right: BinaryTreeNode
- parent: BinaryTreeNode

+ Key {get;}
+ Data {get;}
+ Left {get ; set ;}
+ Right {get ; set ;}
+ Parent {get ; set ;}
+ Add(T element): BinaryTreeNode
+ Contains(T element): bool
+ Search(T element): BinaryTreeNode
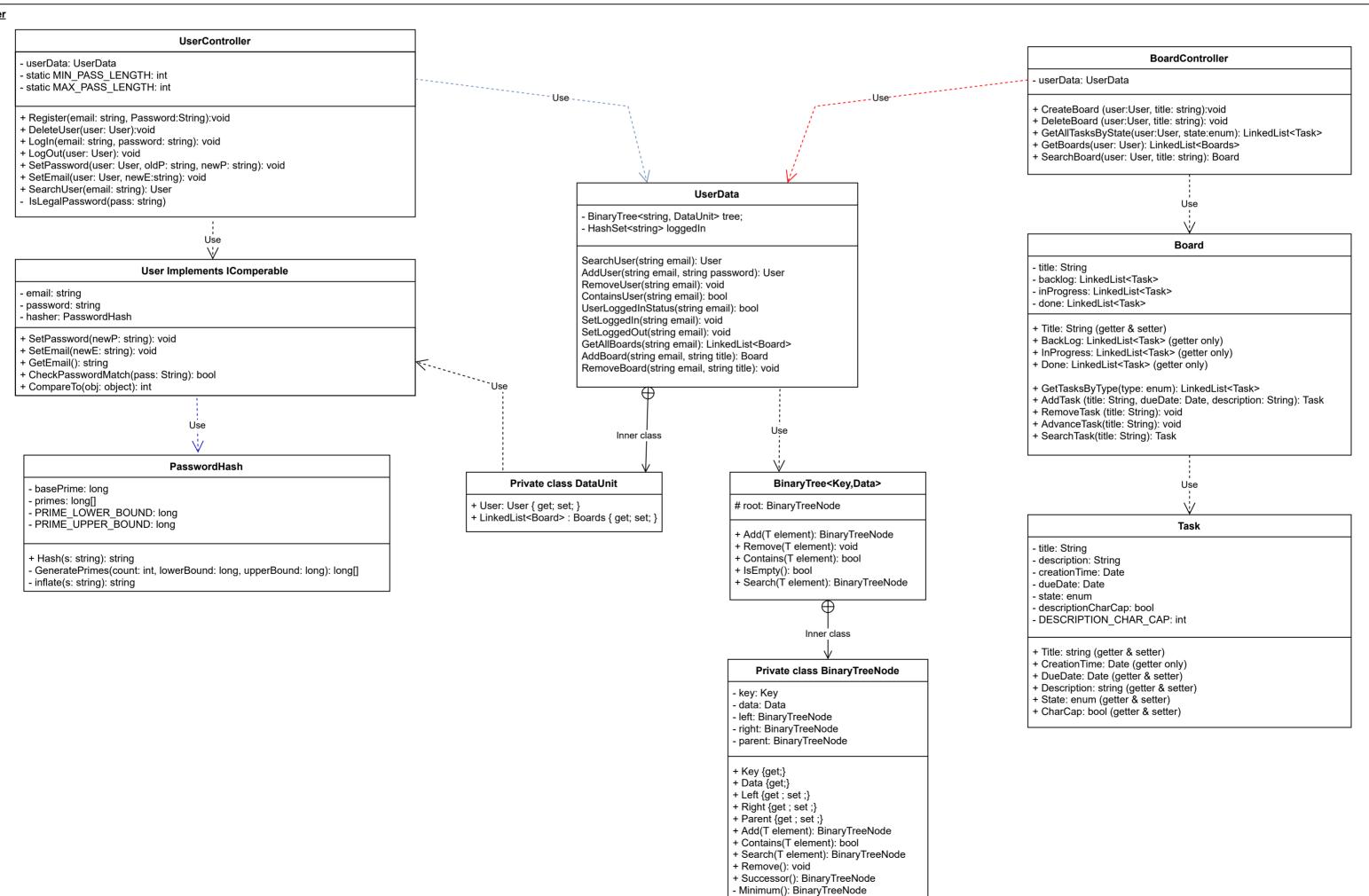+ Remove(): void
+ Successor(): BinaryTreeNode
- Minimum(): BinaryTreeNode
- ThisNodeIsALeftSon(): bool
- ThisNodeIsARightSon(): bool