Pak-Austria Fachhochschule

Institute of Applied sciences And Technology

Group Members:	**Registration No.**
1) Muhammad Saleh Janjua	B22F0086SE017
2) Muhammad Asadullah khan	B22F1392SE153
3) Tayyab khan	B22F0411SE055
4) Anees Ahmad	B22F1231SE145

Course instructor: Dr. Nabeel Ahmad

Department: IT and Cs (Software engineering-22) Section Green

Project Final Report (Software Construction and Development)

Submitted Date: 27th April, 2025.

Agile Team Capacity Tracker - Final Semester Project Report

GitHub Repository Link:

- https://github.com/MuhammadSalehJanjua/Agile-Capacity-Tracker.git

Project Overview:

The Agile Team Capacity Tracker is a full-stack application designed to help software teams track their capacity, plan sprints, and visualize workload distribution efficiently.
The project integrates **Java Spring Boot** backend APIs with a **Next.js** frontend, styled using **Tailwind CSS**, and uses **PostgreSQL** for database management.

Design Pattern Used:

Design Pattern: Singleton Pattern

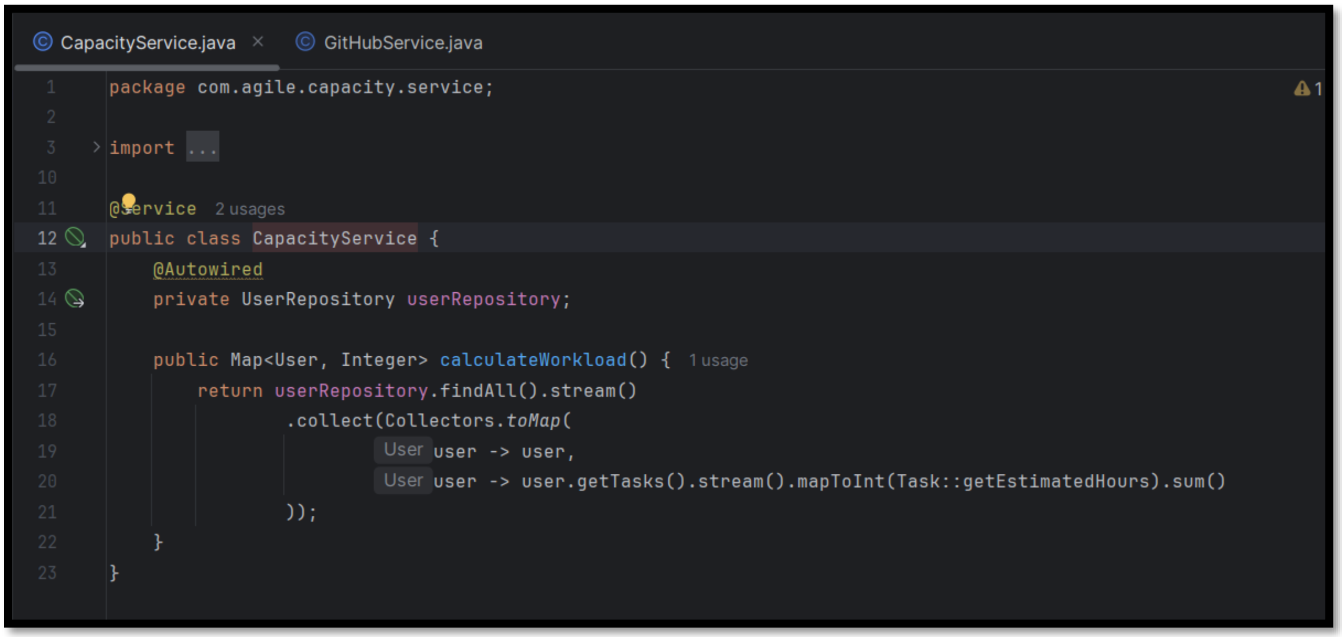
The **Singleton Pattern** ensures that a class has only one instance throughout the application and provides a global point of access to it.

How Design Pattern is Implemented:

In our project, the **CapacityService** class follows the Singleton pattern:

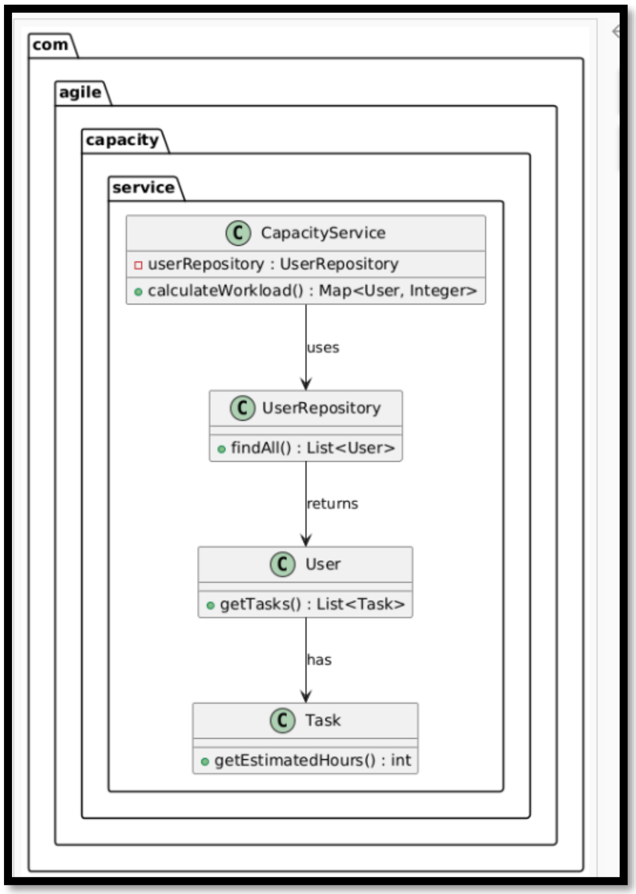
- Only one instance of CapacityService is created.
- This instance handles all operations related to team capacity management, sprint planning, and workload updates.
- Other classes (like Controllers) access this centralized instance for consistent data management.

Important Code Structure (for Singleton):



```java
package com.agile.capacity.service;

import ...

@Service  2 usages
public class CapacityService {
    @Autowired
    private UserRepository userRepository;

    public Map<User, Integer> calculateWorkload() {  1 usage
        return userRepository.findAll().stream()
                .collect(Collectors.toMap(
                        User user -> user,
                        User user -> user.getTasks().stream().mapToInt(Task::getEstimatedHours).sum()
                ));
    }
}
```

Class Diagram for Singleton Design Pattern:

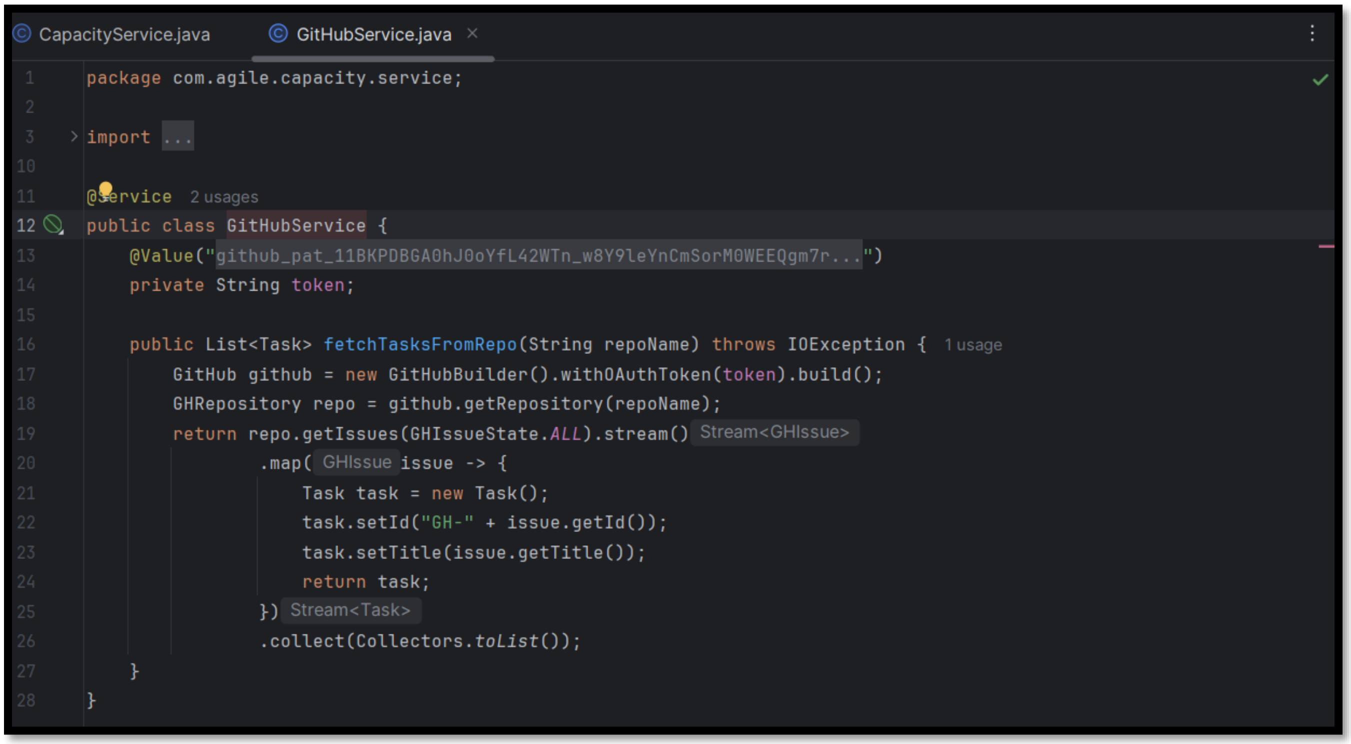


What this shows:

- CapacityService is a service class.
- CapacityService **depends on** UserRepository.
- UserRepository **returns** list of User objects.
- User **has** list of Task objects.
- Task has method getEstimatedHours().
- **Note added** explaining that Singleton behavior is handled by Spring automatically.

Important Code Parts:

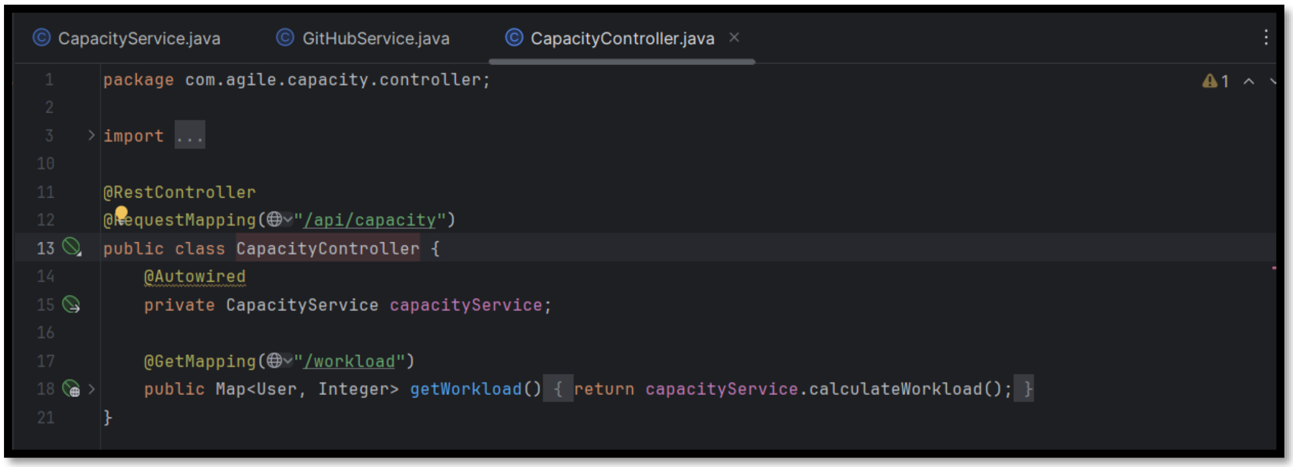
GitHubService.java:



CapacityService.java   GitHubService.java

```java
package com.agile.capacity.service;

import ...

@Service  2 usages
public class GitHubService {
    @Value("github_pat_11BKPDBGA0hJ0oYfL42WTn_w8Y9leYnCmSorM0WEEQgm7r...")
    private String token;

    public List<Task> fetchTasksFromRepo(String repoName) throws IOException {  1 usage
        GitHub github = new GitHubBuilder().withOAuthToken(token).build();
        GHRepository repo = github.getRepository(repoName);
        return repo.getIssues(GHIssueState.ALL).stream() Stream<GHIssue>
                .map( GHIssue issue -> {
                    Task task = new Task();
                    task.setId("GH-" + issue.getId());
                    task.setTitle(issue.getTitle());
                    return task;
                }) Stream<Task>
                .collect(Collectors.toList());
    }
}
```

CapacityController.java:

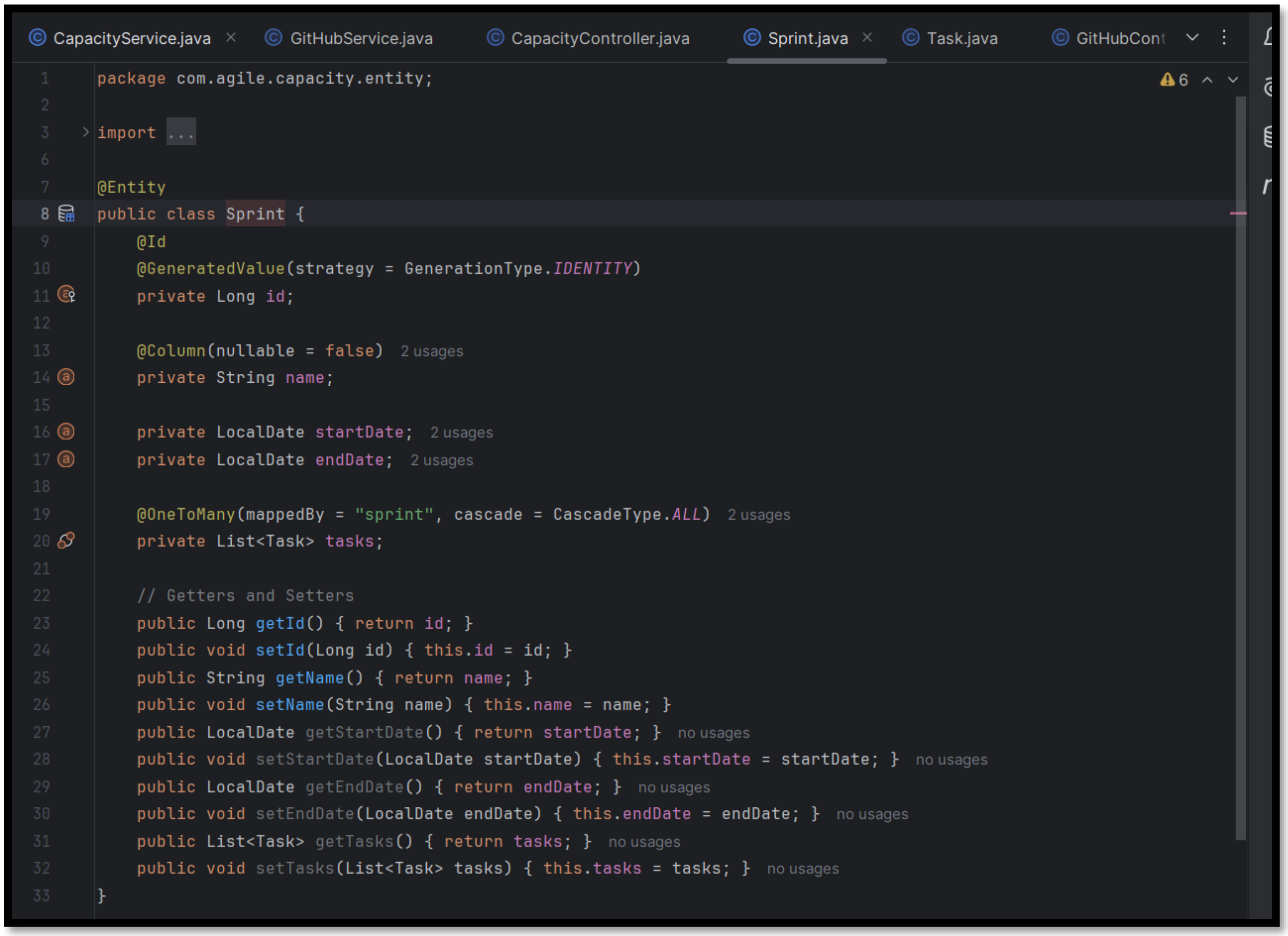


CapacityService.java   GitHubService.java   CapacityController.java

```java
package com.agile.capacity.controller;

import ...

@RestController
@RequestMapping("/api/capacity")
public class CapacityController {
    @Autowired
    private CapacityService capacityService;

    @GetMapping("/workload")
    public Map<User, Integer> getWorkload() { return capacityService.calculateWorkload(); }
}
```

Entity:

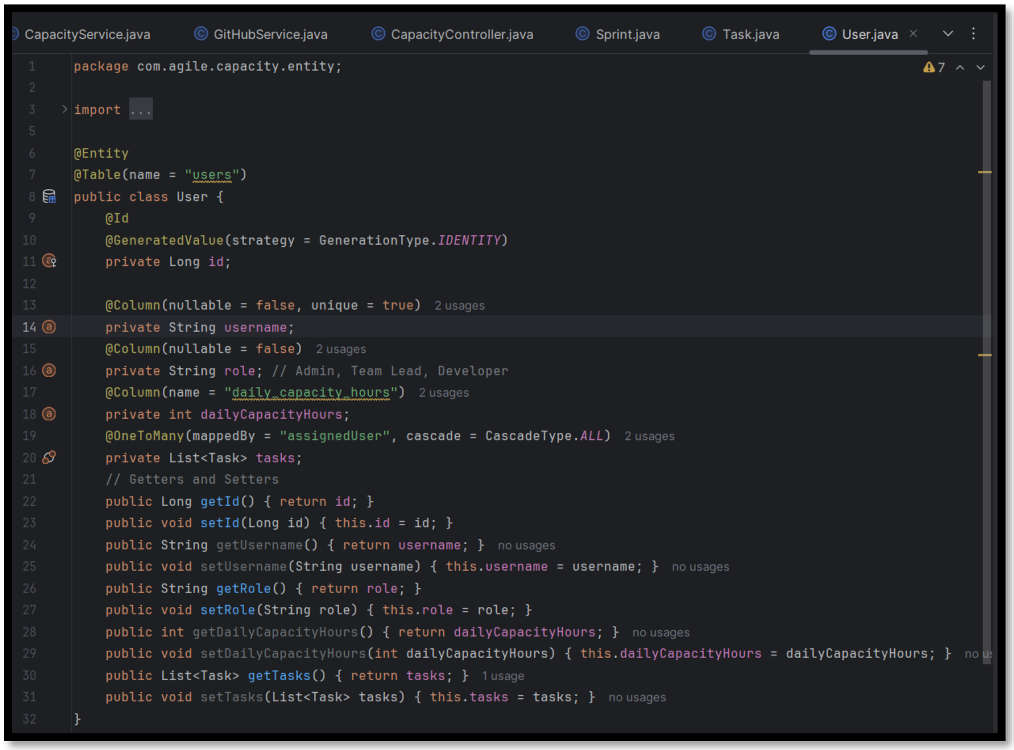
Sprint.java:



```java
package com.agile.capacity.entity;

import ...

@Entity
public class Sprint {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)  2 usages
    private String name;

    private LocalDate startDate;  2 usages
    private LocalDate endDate;  2 usages

    @OneToMany(mappedBy = "sprint", cascade = CascadeType.ALL)  2 usages
    private List<Task> tasks;

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public LocalDate getStartDate() { return startDate; }  no usages
    public void setStartDate(LocalDate startDate) { this.startDate = startDate; }  no usages
    public LocalDate getEndDate() { return endDate; }  no usages
    public void setEndDate(LocalDate endDate) { this.endDate = endDate; }  no usages
    public List<Task> getTasks() { return tasks; }  no usages
    public void setTasks(List<Task> tasks) { this.tasks = tasks; }  no usages
}
```

User.java:

Pak-Austria Fachhochschule
Institute of Applied sciences And Technology

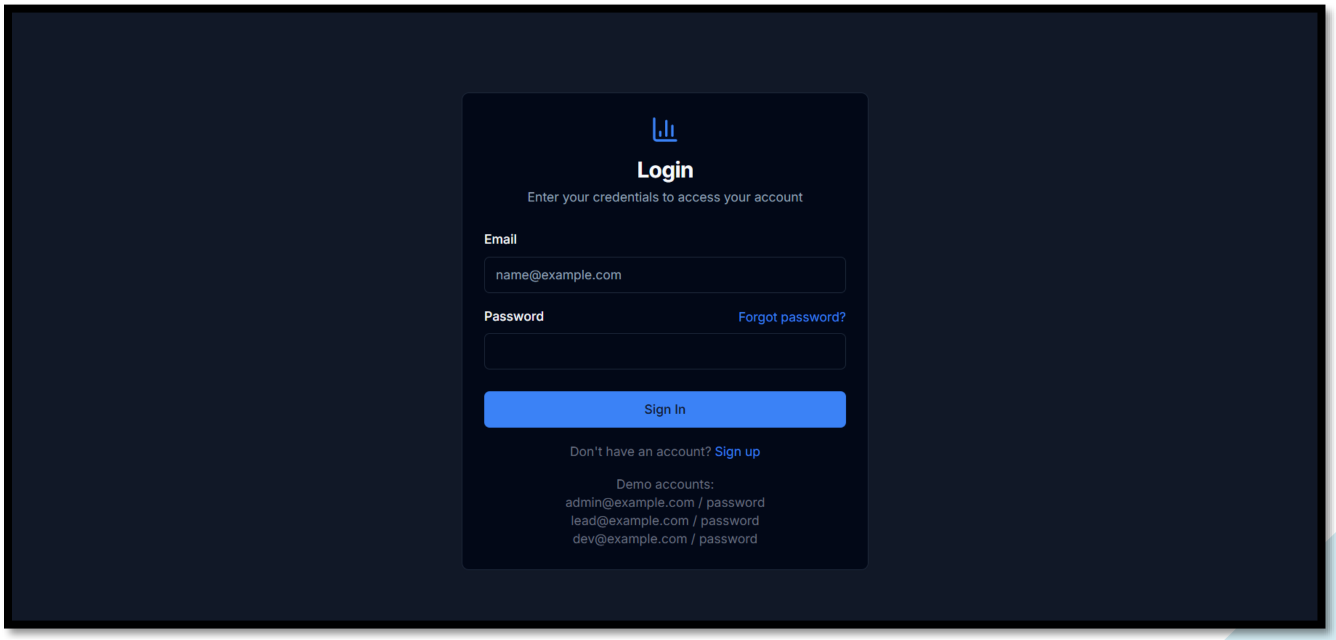


CapacityService.java GitHubService.java CapacityController.java Sprint.java Task.java User.java

```
package com.agile.capacity.entity;

import ...

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)  2 usages
    private String username;
    @Column(nullable = false)  2 usages
    private String role; // Admin, Team Lead, Developer
    @Column(name = "daily_capacity_hours")  2 usages
    private int dailyCapacityHours;
    @OneToMany(mappedBy = "assignedUser", cascade = CascadeType.ALL)  2 usages
    private List<Task> tasks;
    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getUsername() { return username; }  no usages
    public void setUsername(String username) { this.username = username; }  no usages
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
    public int getDailyCapacityHours() { return dailyCapacityHours; }  no usages
    public void setDailyCapacityHours(int dailyCapacityHours) { this.dailyCapacityHours = dailyCapacityHours; }  no u
    public List<Task> getTasks() { return tasks; }  1 usage
    public void setTasks(List<Task> tasks) { this.tasks = tasks; }  no usages
}
```

UI Screenshots of Key Features:

Login Page:



Login

Enter your credentials to access your account

Email

name@example.com

Password

Forgot password?

Sign In

Don't have an account? Sign up

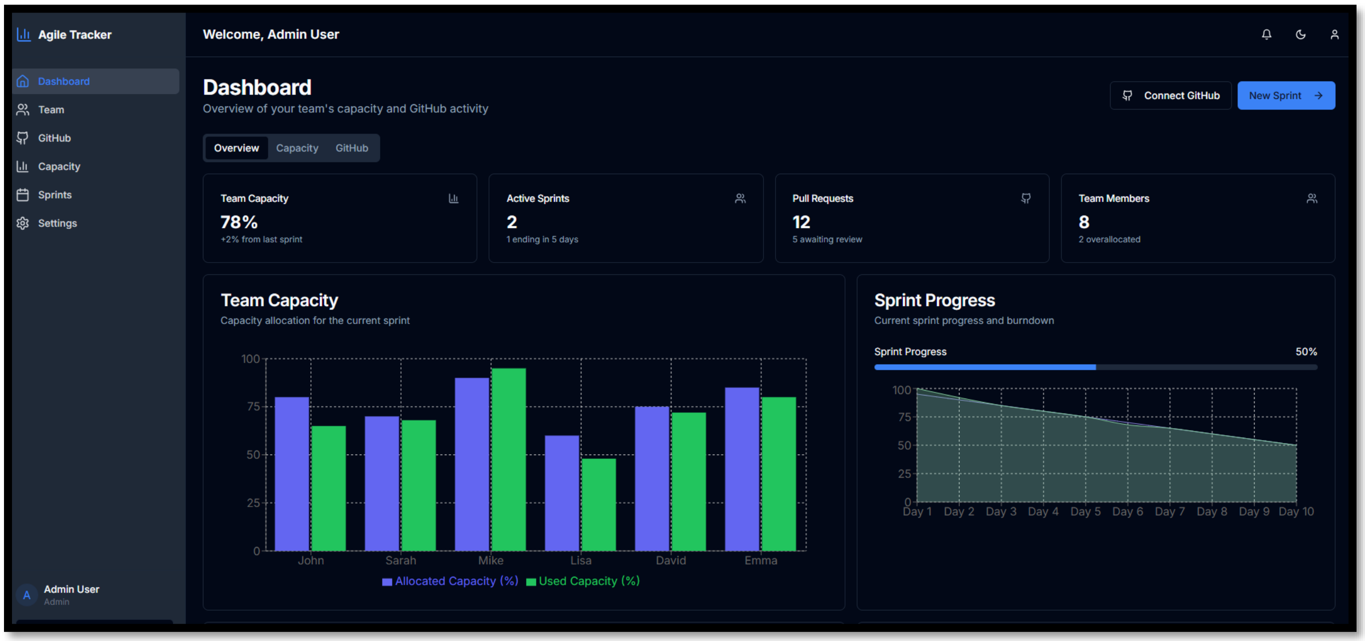
Demo accounts:
admin@example.com / password
lead@example.com / password
dev@example.com / password

Pak-Austria Fachhochschule

Institute of Applied sciences And Technology

Dashboard Overview:

- Team Capacity Visualization



Agile Tracker

Dashboard
Team
GitHub
Capacity
Sprints
Settings

Admin User
Admin

Welcome, Admin User

Dashboard

Overview of your team's capacity and GitHub activity

Connect GitHub | New Sprint

Overview | Capacity | GitHub

Team Capacity	Active Sprints	Pull Requests	Team Members
78%	2	12	8
+2% from last sprint	1 ending in 5 days	5 awaiting review	2 overallocated

Team Capacity

Capacity allocation for the current sprint

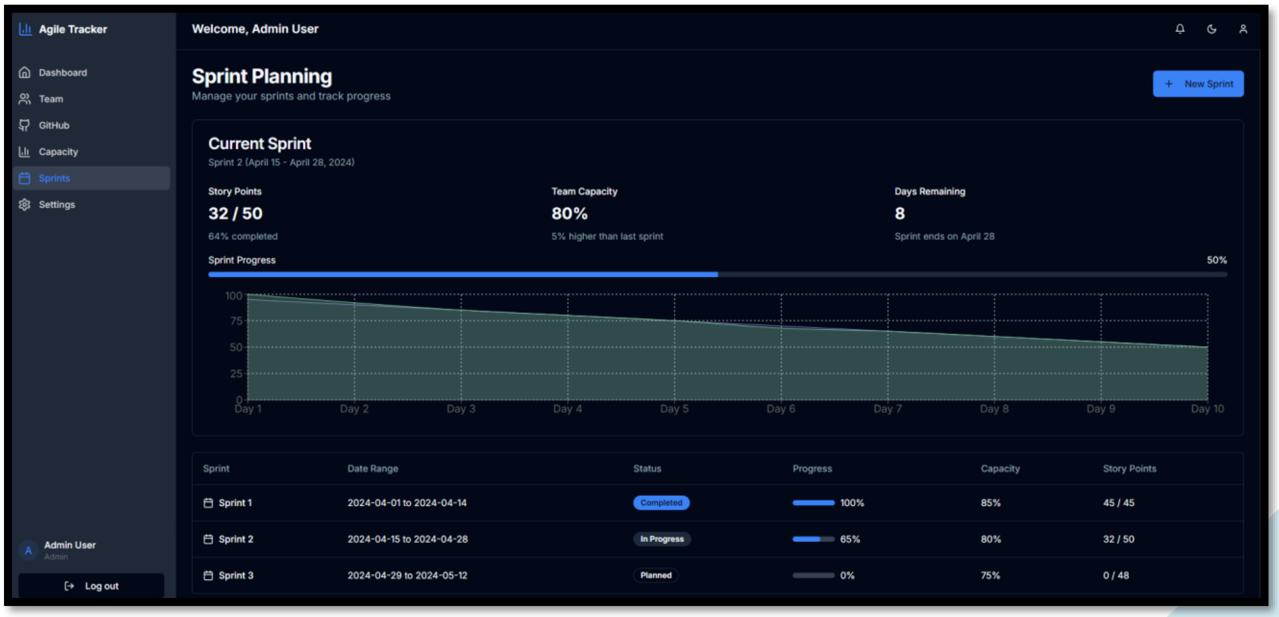
Sprint Progress

Current sprint progress and burndown

Sprint Progress 50%

Sprint Planning Page

- Create Sprints
- Track Capacity vs Actual Workload



Agile Tracker

Dashboard
Team
GitHub
Capacity
Sprints
Settings

Admin User
Admin
Log out

Welcome, Admin User

Sprint Planning

Manage your sprints and track progress

+ New Sprint

Current Sprint

Sprint 2 (April 15 - April 28, 2024)

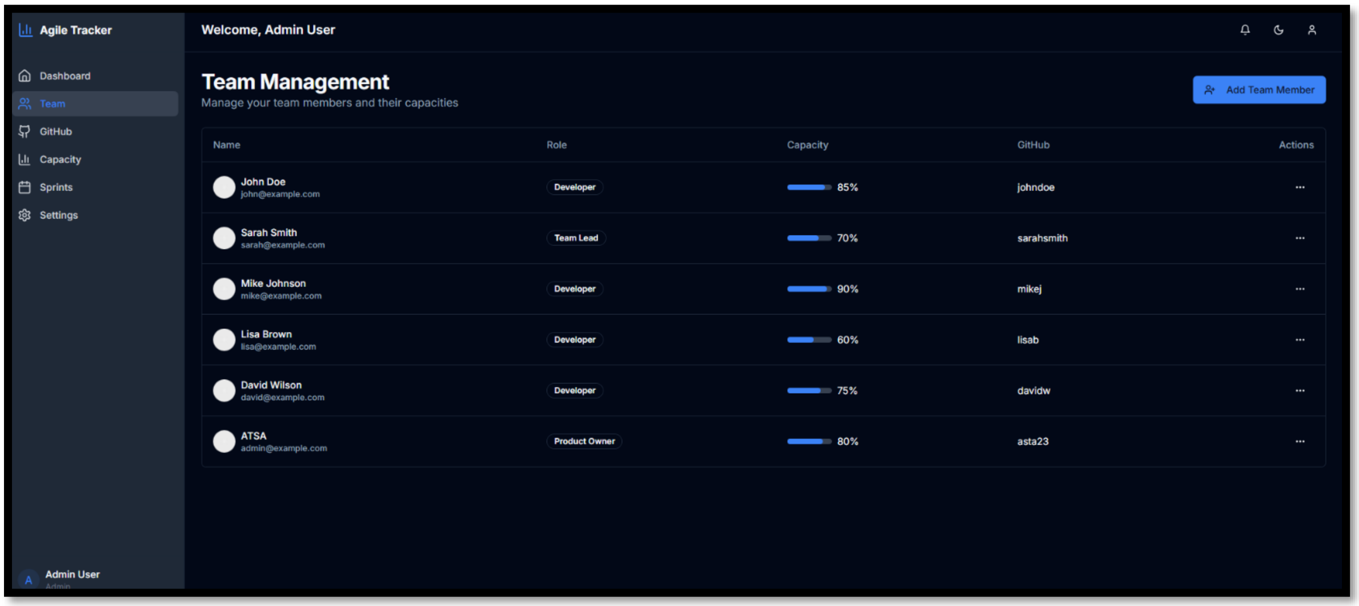
Story Points	Team Capacity	Days Remaining
32 / 50	80%	8
64% completed	5% higher than last sprint	Sprint ends on April 28

Sprint Progress 50%

Sprint	Date Range	Status	Progress	Capacity	Story Points
Sprint 1	2024-04-01 to 2024-04-14	Completed	100%	85%	45 / 45
Sprint 2	2024-04-15 to 2024-04-28	In Progress	65%	80%	32 / 50
Sprint 3	2024-04-29 to 2024-05-12	Planned	0%	75%	0 / 48

Member Management Page:

- Add Members
- Manage Availability and Leaves
- We add ASTA a product owner



Team Management
Manage your team members and their capacities

Name	Role	Capacity	GitHub	Actions
John Doe john@example.com	Developer	85%	johndoe	···
Sarah Smith sarah@example.com	Team Lead	70%	sarahsmith	···
Mike Johnson mike@example.com	Developer	90%	mikej	···
Lisa Brown lisa@example.com	Developer	60%	lisab	···
David Wilson david@example.com	Developer	75%	davidw	···
ATSA admin@example.com	Product Owner	80%	asta23	···

Data Flow Diagrams (DFD):

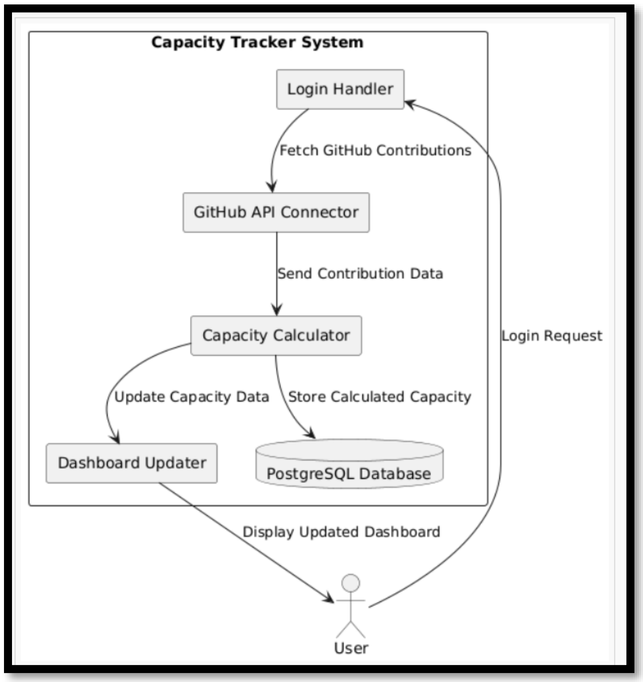
Level 0: Overall System Diagram:

User --> Agile Capacity Tracker System --> GitHub API + Database --> Dashboard & Reports

Level 1: Capacity Tracking Process:

Explanation of the Flow:

- **User** logs into the system.
- **Login Handler** processes the login.
- **GitHub API Connector** fetches contribution data from GitHub.
- **Capacity Calculator** processes this data (workload calculation).
- **Dashboard Updater** updates the user interface with fresh capacity values.
- **Database** stores the calculated capacity for persistence.



Technology Stack:

Layer	Technology
Frontend	Next.js (React), Tailwind CSS
Backend	Java Spring Boot
Database	PostgreSQL
Deployment	Vercel (Frontend), GitHub Actions (CI/CD)

Conclusion:

The Agile Team Capacity Tracker provides a complete solution for sprint planning, workload management, and team availability tracking.

The system ensures efficient, centralized management by using the **Singleton Pattern** for backend services and integrates GitHub APIs for real-time contribution tracking.