**Yarmouk Private University**

**Faculty of Informatics and**

**Communication Engineering**



**Prepared by:**  
Ayham Kefo Nawrz Qal Amr Barakat

**Supervisor:**  
Eng. Anas Mokhalati

Dr. Hiyam Khaddam

Semester: 2023/2024-2

# Abstract

INK MILITARY is an innovative application designed to streamline and digitize the military recruitment and deferment processes, catering specifically to students and administrative personnel. The app leverages cutting-edge technology to provide a seamless and efficient user experience, ensuring that all interactions, from initial registration to deferment processing, are handled smoothly and securely.

INK MILITARY aims to enhance transparency, reduce administrative burden, and foster a   
well-organized recruitment environment. With comprehensive features for both students and administrative staff, the app ensures a well-coordinated and user-friendly approach to military service management.

Nohoud هي منصة ويب بلا بلا بلا بلا

# Acknowledgments

We extend our heartfelt gratitude to Dr. Hiyam Khaddam and Eng. Anas Mukhalati, our esteemed supervisors, for her invaluable guidance and mentorship, and to Dr. Moutasem Shafaumri, the esteemed Dean of our Faculty, for fostering a culture of innovation and scholarly pursuit. Their wise advice and unwavering support have been instrumental in the development of the NOHOUD project.

Additionally, we are deeply grateful to our families and friends for their continuous encouragement and emotional support. This achievement would not have been possible without their constant backing.

# Table of Contents

Contents

[0](#_Toc172373522)

[Abstract 1](#_Toc172373523)

[Acknowledgments 2](#_Toc172373524)

[Table of Contents 3](#_Toc172373525)

[Table of Figures 5](#_Toc172373526)

[List of Abbreviation and Terminology 6](#_Toc172373527)

[Chapter 1: Introduction 7](#_Toc172373528)

[1.1 Purpose: 8](#_Toc172373529)

[1.2 Problem Description: 8](#_Toc172373530)

[1.2.1 Key Issues: 8](#_Toc172373531)

[1.3 Objectives 9](#_Toc172373532)

[1.4 Scope and Limitations 10](#_Toc172373533)

[1.4.2 Scope: 10](#_Toc172373534)

[1.4.2 Limitations: 10](#_Toc172373535)

[1.5 Overview 12](#_Toc172373536)

[Chapter 2: Methodology 13](#_Toc172373537)

[2.1.1 Requirement Analysis and Gathering 14](#_Toc172373538)

[2.1.2 Design Inspiration 14](#_Toc172373539)

[A-UI/UX Principles 15](#_Toc172373540)

[B-User Interface Features 15](#_Toc172373541)

[2.2 Development 16](#_Toc172373542)

[A- Agile 16](#_Toc172373543)

[Iterative Development 16](#_Toc172373544)

[Collaboration 16](#_Toc172373545)

[B- Technology Stack 16](#_Toc172373546)

[Chapter 3:Comparative Analysis with Similar Platforms 20](#_Toc172373547)

[3.1 Introduction 21](#_Toc172373548)

[3.2 Overview of Similar Systems 21](#_Toc172373549)

[3.3 Unique Positioning of INK MILITARY Project 21](#_Toc172373550)

[3.4 Comparison of Functions and Features 23](#_Toc172373551)

[Chapter 4: Requirements engineering 24](#_Toc172373552)

[4.1 Requirements elicitation 25](#_Toc172373553)

[4.1.1 Functional Requirements 25](#_Toc172373554)

[4.1.2 Non-Functional Requirements 26](#_Toc172373555)

[4.2 Requirements Analysis 28](#_Toc172373556)

[4.2.2 Use-case Specification 29](#_Toc172373557)

[5.2 Class Diagram 43](#_Toc172373558)

[Chapter 5: System Design 44](#_Toc172373559)

[5.1 Entity Relationship Diagram 45](#_Toc172373560)

[5.2 Sequence diagrams 46](#_Toc172373561)

[Chapter 6: System Architecture 53](#_Toc172373562)

[6.1 MVC 54](#_Toc172373563)

[6.1.1 Models 54](#_Toc172373564)

[6.1.2 Views 54](#_Toc172373565)

[6.2.3 Controler 54](#_Toc172373566)

[6.2 Overview of NodeJS Architecture 55](#_Toc172373567)

[6.3 Why we use NodeJS 55](#_Toc172373568)

[6.3 Why we use MongoDB 56](#_Toc172373569)

[6.4 3rd Party Services 58](#_Toc172373570)

[Chapter 7: Implementation & Testing 59](#_Toc172373571)

[7.1 Student Application Interfaces 60](#_Toc172373572)

[7.2 Employee Dashboard Interfaces 65](#_Toc172373573)

[7.3 Admin Dashboard Interfaces 67](#_Toc172373574)

[7.3 Testing 71](#_Toc172373575)

[Chapter 8: Summary & Conclusion 77](#_Toc172373576)

[8.1 Project Summary 78](#_Toc172373577)

[8.2 Future Works 78](#_Toc172373578)

[8.3 References 80](#_Toc172373579)

# Table of Figures

[Figure 1:logo 8](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373580)

[Figure 2: Nodejs 17](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373581)

[Figure 3: Express js 17](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373582)

[Figure 4: Express architecture 17](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373583)

[Figure 5: MongoDB 17](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373584)

[Figure 6: Mongoose 18](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373585)

[Figure 7: scss 18](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373586)

[Figure 8: EJS template engine 18](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373587)

[Figure 9: Git 18](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373588)

[Figure 10: Github 18](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373589)

[Figure 11: Vs Code 19](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373590)

[Figure 12: Syria-Visa.sy 21](#_Toc172373591)

[Figure 13: UseCase Diagram 28](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373592)

[Figure 14: class diagram 43](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373593)

[Figure 15: entity relationship diagram 45](file:///C:\Users\ACER\Desktop\APPLIED-vamos.docx#_Toc172373594)

[Figure 16: Register Sequence diagram 46](#_Toc172373595)

[Figure 17: book appointment Sequence diagram 46](#_Toc172373596)

[Figure 18: Login Sequence diagram 47](#_Toc172373597)

[Figure 19: logout Sequence diagram 47](#_Toc172373598)

[Figure 20: Reset password Sequence diagram 48](#_Toc172373599)

[Figure 21: check wallet Sequence diagram 48](#_Toc172373600)

[Figure 22: add money Sequence diagram 49](#_Toc172373601)

[Figure 23: Add Ddferment Sequence diagram 50](#_Toc172373602)

[Figure 24: request state Sequence diagram 50](#_Toc172373603)

[Figure 25: Browse Requests Sequence diagram 51](#_Toc172373604)

[Figure 26: Add Request to work list Sequence diagram 51](#_Toc172373605)

[Figure 27: change Request State Sequence diagram 52](#_Toc172373606)

[Figure 28: Nodejs architecture 55](#_Toc172373607)

[Figure 29: email notification 58](#_Toc172373608)

# List of Abbreviation and Terminology

|  |  |
| --- | --- |
| **Abbreviation/Term** | **Definition** |
| **Nohoud** | An innovative platform designed to streamline military recruitment and deferment processes for students and administrative personnel. |
| **UI/UX** | User Interface/User Experience; principles and practices involved in designing the interface and experience of the platform. |
| **API** | Application Programming Interface; a set of protocols and tools for building software applications. |
| **BSON** | Binary JSON; a binary-encoded serialization of JSON-like documents. |

# Chapter 1: Introduction

# 1.1 Purpose:

Figure 1:logo

# 1.2 Problem Description:

### 1.2.1 Key Issues:

# 1.3 Objectives

# 1.4 Scope and Limitations

### 1.4.2 Scope:

### 1.4.2 Limitations:

# 1.5 Overview

# Chapter 2: Project Management & Methodology

**2.1 Development Methodology: Agile Scrum**

We adopted **Agile Scrum** as the development methodology for this project due to its flexibility, iterative nature, and proven effectiveness in managing complex projects with evolving requirements. With a team of three members possessing diverse skills, ranging from backend development to AI services. Scrum enabled us to deliver functionality incrementally while continuously adapting to new insights.

### 2.1.1 Why Agile Scrum Fits This Project

* **Adaptability to Changing Requirements**: Our platform targets nuanced needs of migrants and refugees. As we progressed and understood our users better, our requirements evolved. Scrum allows backlog reprioritization between sprints without derailing the project plan.
* **Iterative and Incremental Delivery**: We used 1-week sprints. At the end of each sprint, we delivered a working increment—whether a functional endpoint, a microservice, or a UI

feature—allowing for constant feedback and continuous validation.

* **Hands-on Collaboration**: As a small team, we benefitted from close communication and fast decision-making. The daily standups (virtual or async) enabled full visibility of who was working on what, and reduced blockers.

### 2.1.2. Roles and Responsibilities

While a formal Scrum team has distinct roles (Scrum Master, Product Owner, Development Team), we streamlined responsibilities for our context:

* **Scrum Facilitator** (nawrz qal): Ensured Scrum events happened on time, facilitated

planning, and managed progress tracking.

* **Product Ownership** (shared): Prioritized backlog items based on user impact and project goals. Decisions were made collaboratively.
* **Development Team**: All three members contributed to development across all services, splitting work based on technical strength and availability.

### 2.1.3. Sprint Planning and Execution

 We held sprint planning at the start of each sprint to:

* Define the **Sprint Goal**.
* Select user stories from the Product Backlog.
* Break them down into tasks and estimate effort (using relative sizing).

 We executed in **short, focused iterations**, allowing us to continuously adjust course based on learnings or blockers.

 Example Sprint Goal: "Implement authentication service and integrate user role-based

routing for the job platform."

**2.1.4. Self-Organizing Team Structure**

Our team operated as a self-organizing unit:

* **Autonomy**: Each member owned their tasks. Work was pulled, not pushed.
* **Cross-fertilization**: Our team members had varied skills (backend, frontend, AI, DevOps), allowing diverse input across the board.
* **Requisite Variety**: When tasks demanded new approaches, team members quickly adapted and experimented (e.g., new workflows for handling async communication or CI scripts).
* **Learning to Learn**: Retrospectives were used to re-evaluate how we work, not just what we built.
* **Self-evaluation**: We reflected on velocity, quality, and team satisfaction every sprint and adjusted our flow.

**2.1.5. Scrum Ceremonies**

 **Daily Stand-ups**: Brief check-ins, async via chat or GitHub comments. Covered progress, blockers, and coordination.

 **Sprint Review**: Demoed completed features and gathered feedback (both internal and

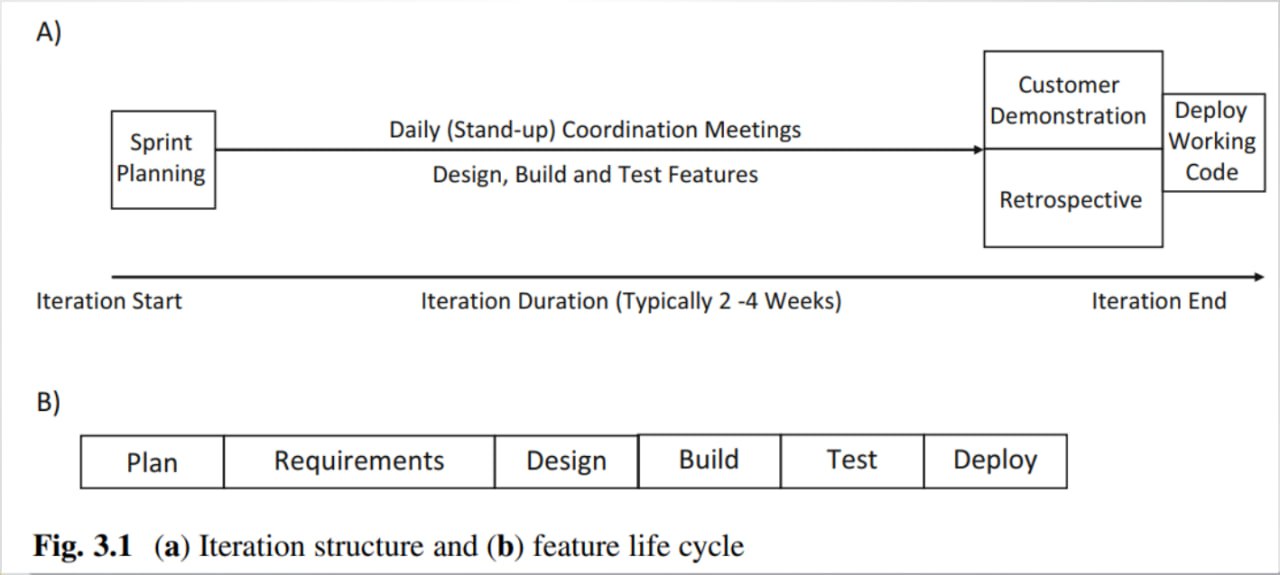
supervisor input).

 **Sprint Retrospective**: Reviewed what went well, what didn’t, and what to improve. Actions were added to the next sprint’s plan.

**2.1.6. Sustainable Engineering Practices**

* We operated under the principle of **Sustainable Pace**. No team member worked nights or weekends. This led to higher-quality, more maintainable code, and consistent delivery.
* We enforced **Collective Code Ownership** through pull requests, shared knowledge of services, and agreed coding standards.
* No module or service was siloed—anyone could improve or refactor any part of the   
  system with proper testing and PR review.

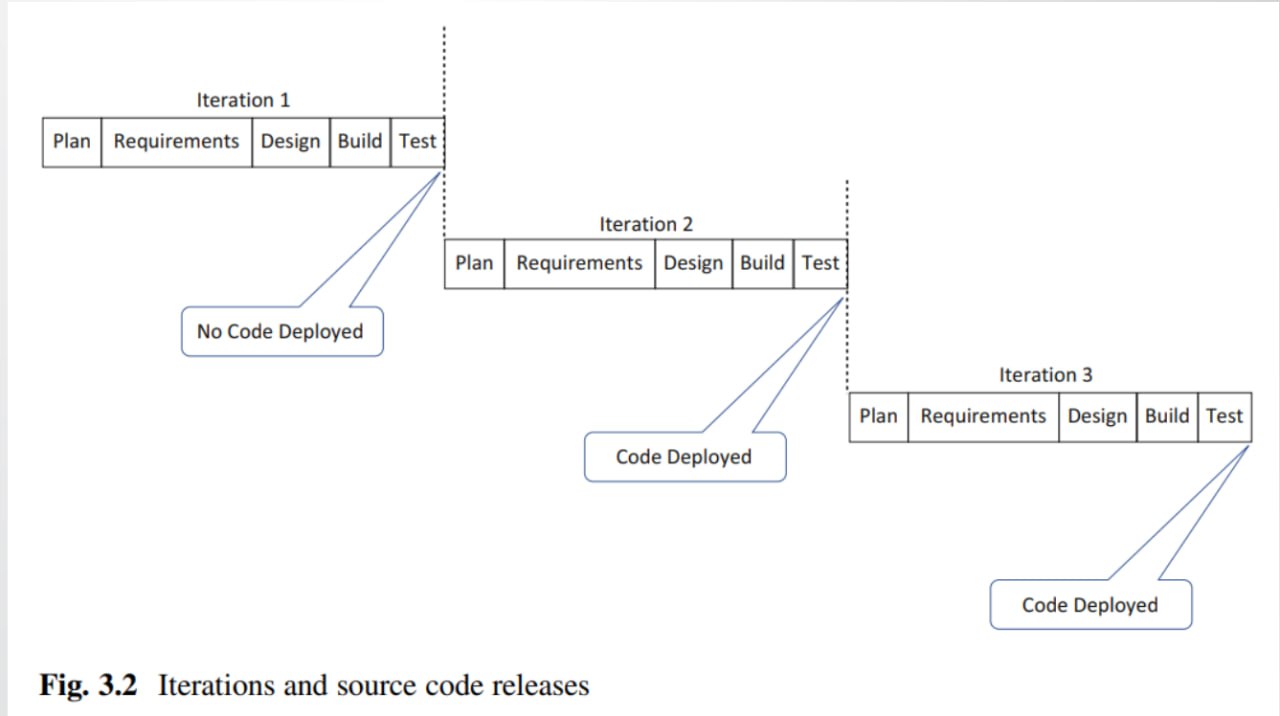
**2.1.7. Iteration Structure and Feature Lifecycle**

****

Iteration structure showing the lifecycle of features through Plan → Requirements → Design → Build → Test within each sprint.

This diagram reflects how we structured each 1-week sprint: we planned, broke down user stories, designed, built locally, and tested before merging.

**2.1.8. Code Publication & Versioning Strategy**

****

**2.2. Remote Pair Programming for Complex Tasks**Although our team was small and partially co-located, we operated as a **virtual team** due to varying schedules and remote collaboration. To overcome the challenges of asynchronous development, we adopted **Remote Pair Programming** during critical implementation phases—especially for complex services like authentication, AI job matching logic, and cross-service integration.

**2.2.1 Purpose and Benefits**

* **Tactical Collaboration**: We used remote pair programming as a focused tactic to tackle complex logic, resolve integration bugs, or review sensitive code (e.g., security features, async flows, database interactions).
* **Real-time Code Quality Feedback**: One member acted as the **Driver** (writing the code), while the other was the **Navigator** (reviewing in real time, thinking about structure, edge cases, and design consistency).
* **Onboarding Support**: When a team member worked on an unfamiliar service, pair programming was used to onboard them quickly without long delays or documentation overhead.

**2.2.2 Setup and Approach**

* We used **Google Meet** for implementation sessions.
* Pairing was scheduled during sprint planning when high-risk or shared ownership tasks were identified.
* Sessions had **clear goals** and were time-boxed to avoid fatigue and maintain flow.
* We occasionally used the **ping-pong** method: one person wrote tests, the other wrote the implementation, and then roles were swapped.

**2.3. Managing a Multi-Service Platform with Scrum**

Our project consists of four core services, each with its own focus, but all contributing to a

unified platform. Scrum helped us manage this structure effectively:

* **Single Product Backlog**: Centralized and prioritized across all services. This ensured that we always focused on the most impactful items regardless of where they belonged in the system.
* **Service-Focused Sprint Goals**: Each sprint goal either cut across multiple services or focused on delivering complete functionality within one. This avoided context switching and maintained team focus.
* **Example**: One sprint focused on user authentication (Auth + Frontend), while another focused on job listing filtering and display logic (Job Service + Search Layer).

**2.4. Project Management Tools: Jira**

We used **Jira** as our main project management and tracking tool, configured to reflect Scrum best practices.

**Jira** is a **project management and issue tracking software** developed by **Atlassian**. It is widely used by teams to plan, track, and manage tasks, bugs, and agile (Scrum & Kanban)   
workflows.

**Key Features of Jira:**

* **Issue Tracking** – Log and track bugs, tasks, and improvements.
* **Agile Project Management** – Supports Scrum and Kanban boards for sprint planning.
* **Customizable Workflows** – Adapt processes to team needs.
* **Dashboards & Reports** – Visualize progress with burndown charts, velocity reports, etc.
* **Integration** – Works with Confluence, Bitbucket, Slack, and other tools.

**2.3.1. Epics, User Stories, and Tasks**

* **Epics**: Represented large features or services (e.g., AI Recommendation, Job Matching,

Authentication).

* **User Stories**: Written in user-centric format and linked to the product goals.
* As a job seeker, I want to receive personalized job listings based on my profile so that I can find relevant opportunities faster.
* **Tasks/Sub-tasks**: Broke user stories down into actionable development items with effort

estimates.

**2.3.2. Sprint Board and Burndown Charts**

* The Jira board had columns for To Do, In Progress, Code Review, and Done.
* **Burndown Charts** were used to track progress during each sprint. Irregularities in the burndown graph helped us identify underestimations or scope creep early.

### 2.3.3. Visual Artifacts from Jira Workspace

**Still empty**

**2.5. Version Control Strategy**

We used **Git** and managed our codebase under a **GitHub Organization** to support collaboration, issue tracking, and code review workflows.

### 2.4.1. Branching and Git

* **Branch Strategy**:
  + main: Production-ready code.
  + develop: Latest working build.
  + feature/xyz: Feature development.
* **Pull Request Process**:
  + Each feature or bug fix was submitted via PR.
  + At least one team member reviewed and approved it before merging.
  + Linked Jira issues and included testing instructions when relevant.

**2.6. Communication and Collaboration**

Effective communication and collaboration were essential to the success of our self-organizing team, especially given our hybrid work environment. We established clear communication channels and structured interactions to ensure transparency, accountability, and fast feedback.

### 2.6.1 Primary Communication Channels

* **Google Meet**: Used for real-time meetings such as sprint planning, sprint reviews, and occasional ad-hoc discussions.
* **Telegram Group**: Our central communication hub. We organized conversations into topic-specific threads to maintain clarity and focus:
  + #visual-identity: UI/UX and branding discussions.
  + #discussion: General project decisions, blockers, and coordination.
  + #files-and-docs: Shared links to documentation, files, and assets.

### 2.6.2 Meeting Cadence

* **Weekly Sprint Meetings**:
  + Sprint Planning at the start of each sprint.
  + Sprint Review/Demo at the end of the sprint.
* **Daily Async Stand-ups**:
  + Team members posted brief updates (what was done, what’s next, blockers) in the Telegram #discussion thread or via GitHub issue comments.

Note:

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………

……………………………………………