

**הנחיות להגשת תוכנה ברמת הצעירות יתרה לתואר
שני במדעי המחשב**

**Guidelines for Submitting Excellent Software for M.Sc. in
Computer Science**

ד"ר יורם סgal

כל הזכויות שמורות - Dr. Segal Yoram ©

2025

תוכן העניינים

4	1 סקירה כללית
4	2 מסמכי פרויקט ותכנון
4	2.1 מסמך דרישות המוצר
4	2.2 מסמך ארכיטקטורה
5	3 תיעוד קוד ומבנה פרויקט
5	3.1 קובץ EMDAER מקיין
5	3.2 מבנה פרויקט מודולרי
6	3.3 איניות קוד והערות
6	4 ניהול קונפיגורציה ובטיחות מידע
7	4.1 קבצי קונפיגורציה
7	4.2 אבטחת מידע
7	5 בדיקות ואיניות תוכנה
7	5.1 בדיקות ייחוד
8	5.2 טיפול במקרי קיצון ותקלות
8	5.3 תוצאות בדיקה צפויות
8	6 מחקר וניתוח תוצאות
8	6.1 חקר פרמטרים
9	6.2 מהברת ניתוח תוצאות
9	6.3 הצגה ויוזאלית של תוצאות
9	7 ממשק משתמש וחווית משתמש
9	7.1 קרייטריוני איניות
10	7.2 תיעוד ממשק
10	8 ניהול גרסאות ותיעוד פיתוח
10	8.1 שיטות עבודה מומלצות עם tiG
10	8.2 ספר הפרומפטים
11	9 עליות ותמהror
11	9.1 ניתוח עליות
11	9.2 ניהול תקציב

11	הרחבת ותחזוקתיות	10
11	נקודות הרחבת	10.1
12	תחזוקתיות	10.2
12	תקני איכות בינלאומיים	11
12	מאפייני איכות מוצר	11.1
13	רשימת בדיקה סופית	12
13	מקורות ותקנים נוספים	13
13	הערה חשובה	14
15 English References		14

1 סקירה כללית

מסמך זה מגדר את הקритריונים להגשת פרויקט תוכנה ברמת מצוינות אקדמית המתאימה לסטודנטים מצטיינים במיוחד לתואר שני במדעי המחשב [1], [2]. הדרישות מתמקדות בפתרונות אינטגרטיביים, תיעוד מקיף, ובהדגמת יכולות מחקר ופיתוח ברמה גבוהה. בכל מקום שכותב "פרויקט" הכוונה למטרה, משימה או פרויקט שנייהים במסגרת הקורס להגשת.

2 מסמכים פרויקט ותכנון

כל פרויקט תוכנה מכוון מתחילה בתיעוד ברור ומקיף של הדרישות והתכנון. מסמכים אלה משמשים כבסיס לפיתוח, מבטיחים הבנה משותפת בין כל הגורמים המעורבים, ומאפשרים מעקב אחר התקדמות הפרויקט לאורץ זמן.

2.1 מסמך דרישות המוצר

מסמך דרישות המוצר, או Product Requirements Document (PRD), הוא המסמך המרכזי המגדיר את מטרת הפרויקט ואת הדרישות ממנה [3], [4], [5]. המסמך מתחילה בסקירה כללית של הפרויקט והקשר בו הוא פועל, כולל תיאור ברור של עיית המשמש שהפרויקט נועד לפטור, ניתוח של השוק התחרותי והצבה אסטרטגית של הפרויקט, ויזיהו של קהל היעד והצדדים המעורבאים (stakeholders). לאחר מכן, המסמך מגדר את יעד הפרויקט ומדדי ההצלחה שלו, תוך הגדרת יעדים מדדיים וברורים, מדדי KPI לכימות ההשפעה וההצלחה, וקריטריוני קבלה (acceptance criteria) המאפשרים להערכץ האם הפרויקט עומד בדרישות. חלק מרכזי נוסף במסמך הוא תיאור הדרישות הפונקציונליות והלא-פונקציונליות. זה כולל רשימת תכונות מפורטת עם עדיפויות ברורות, סיפורי משתמש (user stories) ותרחישי שימוש (use cases) המתארים כיצד המשתמשים יפעלו עם המערכת, ודרישות ביצועים, אבטחה, זמינות וscalability [6], [7], [8] המבטיחות שהמערכת תוכל לפעול בצורה יעילה ואמינה. המסמך גם מזהה הנחות, תלויות ומגבליות, כולל מערכות חיצונית ותלויות טכנולוגיות, מגבלות טכניות וארגוניות, ופריטים שנמצאים מחוץ לתחום הפרויקט (out-of-scope items). לבסוף, המסמך כולל ציר זמן מפורט ואבני דרך (timeline and milestones) עם לוח זמנים המציין נקודות ביקורת ומשולחים צפויים (deliverables) בכל שלב.

2.2 מסמך ארכיטקטורה

מסמך הארכיטקטורה (Architecture Documentation) מספק תיאור טכני מקיף של מבנה המערכת ואופן פעולתה. המסמך כולל תרשימים ויזואליים המסבירים את הארכיטקטורה ברמות שונות של פירוט, כגון תרשימי C4 Model המציגים את המערכת ברמות Context, Component, Code-Container, Component וtrsheimi deployment המציגים תהליכי מורכבים וឥינטראקציות בין רכיבים, וtrsheimi deployment המציגים את התשתיות הטכולוגיות והארכיטקטורה התפעולית (operational architecture).

נוסף על התרשימים, המסמך כולל תיעוד החלטות ארכיטקטוניות (Architecture Decisions - ADRs) המסביר את הרצינול להחלטות ארכיטקטוניות מרכזיות, ניתוח של trade-offs וآلטרנטיבות שנשקלו בתהליך התכנון. המסמך גם מספק תיעוד מפורט של ממשקי התכונות (API) וממשקים אחרים, כולל תיאור מפורט של כל ממשק ציבורי, סכימות נתונים (data schemas) וקונטראקטים המגדירים את האינטראקציות בין רכיבי המערכת.

3. **תיעוד קוד ומבנה פרויקט**

תיעוד קוד נכון ומבנה פרויקט מסודר הם יסודות חיוניים לפיתוח תוכנה מקצועית. תיעוד טוב מאפשר למפתחים אחרים להבין את הקוד במדויק, להשתמש בו בצורה נכונה, ולתרום לפרויקט ביעילות.

3.1 **קובץ README מקייף**

קובץ README הוא המסמך המרכזי המלאה כל פרויקט תוכנה ומשמש כמדריך למשתמש ברמת user manual מלא[9], [10]. הקובץ מתייחס בהוראות התקנה (Installation Instructions) המפרטות את דרישות המערכת (system requirements), מספקות הוראות התקנה שלב-אחר-שלב לשביבות שונות, מסבירות כיצד משתני סביבה (environment variables), וכוללות מדריך לפתרון בעיות נפוצות (troubleshooting).

לאחר ההתקנה, הקובץ מספק הוראות הפעלה (Usage Instructions) המתארות כיצד להריץ את התוכנה במצבים שונים, מסבירות דגלים ואפשרויות ב-CLI או GUI, ומציגות Examples & Demonstrations workflow טיפוסי למשתמש. הקובץ גם כולל דוגמאות והדגמות (Examples & Demonstrations) עם דוגמאות קוד מעשיות להרצאה, צילומי מסך של ממשק המשתמש, תרחישים שימוש נפוצים, וקישורים לסרטוני הדוגמה אם רלוונטי.

נוסף על כך, README כולל מדריך תצורה (Configuration Guide) המסביר את קבצי הקונפיגורציה והפרמטרים הנחוצים לכיוול והשפעתם על פועלות המערכת, הנחיות לתרומות קוד (Contribution Guidelines) הכוללות תקני קוד וסגנון, ומידע על רישוי השימוש וייחוס לספריות צד שלישי ותורמים (License & Credits).

3.2 **מבנה פרויקט מודולרי**

ארגון נכון של מבנה הפרויקט הוא מפתח לתחזקה עילית ולהתפתחות עתידית של הקוד. עקרונות הארגון כוללים חלוקה לוגית של הפרויקט לתיקות לפי תפקיד, כגון קוד מקור, בדיקות, תיעוד, נתונים, תוכאות, קונפיגורציה ומשאבים. הארגון יכול להיות מבוסס-תכונות (feature-based) או בארכיטקטורה שכבותית (layered architecture), תוך הפרדה ברורה בין קוד, נתונים, תוכאות ותיעוד.

גודל הקבצים הוא שיקול חשוב בארגון הפרויקט. קבצי קוד לא צריכים לעלות על כ-150 שורות, מה שבטיחת שכל קובץ ממוקד באחריות אחת וקל להבנה. כאשר קובץ הופך גדול מדי, יש לפרק אותו לפונקציות ומודולים קטנים יותר תוך שמירה על הפרדת

אחריות (separation of concerns). חשוב גם לשמור על עקביות בשמות תיקיות וקבצים, תוך שימוש באותו סגנון naming בכל הפרויקט.
דוגמה למבנה פרויקט מומלץ:

```
project-root/ └── elpmaxE erutcurtS tcejorP  
    ├── src/          # Source code|__  
    │   ├── agents/    # Agent modules|__  
    │   ├── utils/     # Helper functions|__  
    │   └── config/    # Configuration code|__  
    ├── tests/        # Unit and integration tests|__  
    ├── data/         # Databases and input files|__  
    ├── results/      # Experiment results|__  
    ├── docs/         # Additional documentation|__  
    ├── config/       # Configuration files|__  
    ├── assets/        # Images, graphs, resources|__  
    ├── notebooks/    # Analysis notebooks|__  
    ├── README.md|__  
    ├── requirements.txt|__  
    └── .gitignore
```

3.3 איקות קוד והערות

איקות הקוד נמדדת לא רק בפונקציונליות שלו אלא גם בקלות הקריאה והתחזוקה שלו. תקני הערות בקוד (Code Comments Standards) [11], [12], [13] דורשים שהערות יסבירו את ה-"למה" ולא רק את ה-"מה", כלומר יתמקדו בהחלטות עיצוב ורציונל ולא רק בתיאור הפעולות. כל פונקציה, מחלקה (class) ומודול (module) צריים לכלול Docstrings המסבירים את התכליית, הפרמטרים וערך החזרה. הערות צרכיות להסביר החלטות עיצוב מורכבות, לתעד הנחות ותנאים מוקדמים, ולהתעדכן יחד עם שינויי הקוד.

עקרונות כתיבת קוד איקוטי כוללים שימוש בשמות משתנים ופונקציות תיאוריים ומדויקים, כתיבת פונקציות קצרות ומוקדמות בעיקרון האחירות היחידה (single responsibility), הימנעות מקוד כפול לפי עיקרונו (DRY - Don't Repeat Yourself), ושמירה על עקביות בסגנון הקוד לאורך כל הפרויקט.

4 ניהול קונפיגורציה ואבטחת מידע

ניהול נכון של קונפיגורציה ואבטחת מידע הם קריטיים להפעלה בטוחה של מערכות תוכנה, במיוחד כאשר מדובר בסביבות ייצור או עבודה עם מידע רגיש.

4.1 קבצי קונפיגורציה

הפרדת הגדרות הקונפיגורציה מהקוד היא עיקרון יסודי בפיתוח תוכנה מודרני. ניהול הגדרות נעשה באמצעות קבצי קונפיגורציה נפרדים בפורמטים סטנדרטיים כגון `son`, `yaml` או `env`. תוך הימנעות מקובעים (hardcoded values) בתוך הקוד עצמו. חשוב לספק קבצי דוגמה כגון `example.env`. עם ברירות מחדל שומרות על אבטחה, ולתעד כל פרמטר קונפיגורציה כדי להקל על המשתמשים להבין את ההשפעות של כל הגדרה. בעבודה עם מערכת בקרת גרסאות Git, חשוב מאוד להשתמש בקובץ `gitignore`. כדי למנוע העלאה בטעות של קבצי קונפיגורציה רגילים למאגר הקוד. כמו כן, מומלץ ליצור קבצי `template` לקונפיגורציה עבור סביבות שונות כגון פיתוח (`dev`), ביניים (`staging`) ויצור (`production`), כדי להקל על המעבר בין סביבות תוך שמירה על הפרדה נכונה.

4.2 אבטחת מידע

הגנה על מפתחות API וסודות אחרים היא קריטית למניעת דליפת מידע ושימוש לא מורשה [14], [15]. הכלל המרכזי הוא שאסור בהחלה לשמר מפתחות API בקוד המקורי (`environment variables`) בלבד. במקרה זה, יש להשתמש אך ורק במשתני סביבה (`os.environ.get("API_KEY")`). יש להסתיר את קבצי secrets מהמערכת `gitignore`, ובנסיבות יוצר להשתמש בכלים ניהול סודות (`-man`) מתקדמים (management tools).

בנוסף על הגנה בסיסית, חשוב לישם החלפה תקופתית של מפתחות (rotation), לנטר השימוש במפתחות ה-API כדי לאוות חריגות, ולהגביל הרשות למינימום הנדרש (least privilege) כדי להקטין את הנזק האפשרי במקרה של פריצה.

5 בדיקות ואיכות תוכנה

בדיקות תוכנה מקיפות הן אבן הפינה של איכות הקוד ואמינות המערכת. מערכת בדיקות טוביה מגלה באגים מוקדם, מבטיחה שהקוד עומד בדרישות, ומאפשרת ביטחון בשינויים עתידיים.

5.1 בדיקות יחידה

בדיקות יחידה (Unit Tests) בודקות רכיבים בודדים של הקוד באופן מבודד. דרישות כיסוי הבדיקות (Test Coverage) [17], [18] מגדירות שיש לשאוף לכיסוי מינימלי של 70-80% ל코드 חדש, עם דגש על כיסוי מוגבר לקוד קריטי ולוגיקה עסקית. הבדיקות צריכות לכיסות לא רק את המסלולים הרגילים אלא גם מקרים קיצוניים (edge cases) ותנאי גבול.

סוגים הקיימים הנדרשים כוללים כיסוי החלטות (Statement coverage) המבטיח שככל שורת קוד הורצת לפחות פעם אחת, כיסוי ענפים (Branch coverage) המבטיח שככל החלטה נבדקה עם כל האפשרויות, וכיסוי מסלולים (Path coverage) למסלולים קריטיים המבטיח שצירופים שונים של החלטות נבדקו. לביצוע הבדיקות יש להשתמש במסגרות בדיקה סטנדרטיות

כגון unittest או pytest, לבצע אוטומציה של הבדיקות ב-pipeline CI/CD, וליצור דוחות כיסוי (coverage reports) המאפשרים מעקב אחר איכות הבדיקות.

5.2 טיפול במרקרי קיצון ותקלות

זהוי ותיעוד מקרים קיצון (Edge Cases) הוא חלק חיוני מפיתוח תוכנה איכותית. יש להזות באופן שיטתי תנאי גבול ומרקרי קיצון, לטעד כל מקרה עם תיאור מפורט של הקלט הצפוי והתגובה הנדרשת, ולכלול צילומי מסך של תקלות כאשר זה רלוונטי. מנגנון טיפול בשגיאות (Error Handling) נדרש לכלול תכנות הגנתי (defensive programming) עם בדיקות קלט מקיפות, הודעות שגיאה ברורות וموעילות למשתמש, רישום לוגים מפורט לצורך ניפוי שגיאות (debugging), והידרדרות חלקה (graceful degradation) במקרים כשל כך שהמערכת ממשיכה לפעול במידת האפשר.

תיעוד התקלות נדרש לכלול תיאור מדויק של התקלה והסיבה לה, תיאור של תגובת המערכת וכי怎ד היא מטפלת בשגיאה, והערכתה של ההשפעה על המשמש או על המערכת כולה. תיעוד זה משמש גם למניעת תקלות דומות בעתיד ולשיפור מתמיד של המערכת.

5.3 תוצאות בדיקה צפויות

תיעוד תוצאות בדיקה צפויות מאפשר השוואת מהירה בין התנагות בפועל לבין התנагות המוצפה. יש לטעד את תוצאות הרצאה הצפויות לכל בדיקה, ליצור דוחות automated testing עם pass/fail rates המציגים את שיעור ההצלחה, ולשמור לוגים של הרצאות מוצלחות וכשלות לניתוח עתידי ולמידה מטוענית.

6 מחקר וניתוח תוצאות

מחקר עמוק וניתוח תוצאות הם שבדיל בין פרויקט תוכנה רגיל לבין עבודה אקדמית ברמת מצוינות. החלק המחקרי כולל ניסויים שיטתיים, ניתוח כמותי ו איכותי, הציג והזואלית של התוצאות.

6.1 חקר פרמטרים

ניתוח רגישות פרמטרים (Sensitivity Analysis) הוא תהליך שיטתי של בדיקת השפעת פרמטרים שונים על ביצוע המערכת. התהליך כולל ביצוע ניסויים שיטתיים עם שינוי מבוקר של פרמטרים, תיעוד מדויק של השפעת כל פרמטר על התוצאות, ושימוש בשיטות ניתוח מתוקדמות כגון נגזרות חלקיות (partial derivatives), ניתוח מבוסס שונות (variance-based analysis), או גישת "פרמטר-אחד-בכל-פעם" (one-at-a-time, OAT). המטריה היא זהות את הפרמטרים הקритיים שימושיים ביותר על הביצועים ולהבין את הקשרים ביניהם.

תיעוד הניסויים כולל יצירת טבלה מסודרת של כל הניסויים עם ערכי הפרמטרים והתוצאות המתאימות, הפקת גרפים ממוחשיים כגון line charts למגמות, heatmaps לרגישות

בין-פרמטרית, ו-plots sensitivity להערכת השפעות, וביצוע ניתוח סטטיסטי של התוצאות לקבעת מובהקות ורמת ביטחון.

6.2 מחברת ניתוח תוצאות

מחברת ניתוח תוצאות Results Analysis Notebook היא כלי מרכזי להצגת הממחקר בפורמט אינטראקטיבית ומפורטת. עומק הניתוח מושג באמצעות שימוש ב-Jupyter Notebook או כלים דומים המאפשרים שילוב של קוד, טקסט ותוצאות, ביצוע ניתוח מתודדי ושיטתי של תוצאות הניסויים, השוואת בין אלגוריתמים שונים, תוצאות או גישות מתודולוגיות, והכללת הוכחות מתמטיות או ניתוחים תיאורתיים כאשר זה רלוונטי.

הכללת נוסחאות ונתוניים נעשית באמצעות LaTeX לכתיבת משוואות ונוסחאות מנוסחים, הסברים מתמטיים מפורטים למודלים ואלגוריתמים הכלולים את העקרונות התיאורתיים, ואסמכתאות לספרות אקדמית ומחקרים קודמים המקנים אמינות ומקצועיות לעבודה.

6.3 הציג ויזואלית של תוצאות

ויזואליזציה איקוונית של נתונים היא חיונית להעברת המסר המ[hash] בפורמה ומשכנת. סוגי הוויזואלייזציות כוללים Bar charts להשוואות קטגוריות המאפשרות השוואת מהירה בין אפשרויות שונות, Line charts ל\Migrations לאורך זמן המראות שינויים והתפתחויות, Scatter plots לזיהוי מתאימים וקשרים בין משתנים, Heatmaps להציג רגישות פרמטרים בשתי ממדים, Box plots להציג התפלגות ומדדים סטטיסטיים, ו-Waterfall charts לניתוח שינויים רציפים ותרומות יחסיות.

aicoot הגרפים נמדדת בהירות ובדיקה התווות, בשימושocabים עקבאים ונגישים שמתאים גם לאנשים עם לקויות ראייה, בכלל כתובים captions מפורטים ומקרא legends) ברור, וברזולציה גבוהה המתאימה לפרטומים אקדמיים או מקצועיים. ככל שהגרפים מקצועיים וברורים יותר, כך ההשפעה של הממחקר גדולה יותר.

7 ממשק משתמש וחווית משתמש

משק משתמש (UI - User Interface) וחווית משתמש (User Experience - UX) טובים הם קריטיים להצלחת כל מערכת תוכנה. אפילו מערכת עם פונקציונליות מעולה עלולה להיכשל אם המשתמשים מתकשים להשתמש בה.

7.1 קритריוני איקות

kritérioni השימוש (Usability Criteria) כוללים מספר מימדים חשובים. קלות למידה (Learnability) מודדת עד כמה קל למשתמשים חדשים ללמידה לשימוש המערכת,יעילות (Efficiency) בודקת עד כמה מהר משתמשים מנוסים יכולם לבצע משימות, זכירות (Memorability) מעריכה עד כמה קל למשתמשים לחזור למערכת לאחר הפסקה ולזכור

כיצד להשתמש בה, מניעת שגיאות (Error Prevention) בודקת עד כמה המערכת מגינה על המשתמש מפני טעויות, ושביעות רצון (Satisfaction) מודדת עד כמה משתמשים נהנים מהעבודה עם המערכת.

עשרה ההיוריסטיות של נילסן (Nielsen's 10 Heuristics) [21] הן קבוצה מוכרת של עקרונות לעיצוב ממשקם. הן כוללות נראות סטטוס המערכת (Visibility of system status), התאמה בין המערכת והעולם האמיתי (Match between system and real world), שליטה וחופש למשתמש (Consistency and standards), עקביות ותקינה (User control and freedom), מניעת שגיאות (Error prevention), זיהוי במקום זיכרון (Recognition over recall), גמישות ויעילות שימוש (Flexibility and efficiency of use), עיצוב אסתטי ומינימליסטי (Aesthetic and minimalist design), עזרה למשתמשים לזהות ולהתואושש משגיאות (Help users recognize and recover from errors), ועזרה ותיעוד (Help and documentation).

7.2 תיעוד ממشك

תיעוד מקיף של הממשק כולל צילומי מסך של כל מסך ומצב אפשרי במערכת, תיאור מפורט של workflow טיפוסי של משתמש המראה את המסלול שלם מתחילה השימוש ועד להשגת המטרה, הסברים על אינטראקציות ופידבק שהמערכת נותנת למשתמש בתגובה לפעולות שונות, ושיקולי נגישות (accessibility considerations) המבטיחים שהמערכת שמיישת גם לאנשים עם מוגבלות.

8 ניהול גרסאות ותיעוד פיתוח

ניהול גרסאות נכון הוא חיוני לעובדות צוות, למעקב אחר שינויים, ולאפשרות לחזור לגרסאות קודמות במקרה הצורך. בנוסף, תיעוד תהליכי הפיתוח עוזר להבין את החלטות שנתקבלו לאורך הדרכ.

8.1 שיטות עבודה מומלצות עם Git

שיטות עבודה מומלצות (Git Best Practices) כוללות שמירה על היסטוריה commits ברורה עם הודעות משמעותיות המתארות מה השתנה ולמה, שימוש ב-branches נפרדים לפיתוח תוכנות חדשות כדי לשמר על יציבות הענף הראשי, ביצוע סקירות קוד (code reviews) באמצעות Pull Requests לפני מיזוג שינויים, ושימוש ב-tagging לסימון גרסאות מרכזיות ומשמעותיות של המערכת.

8.2 ספר הפרומפטים

תיעוד תהליכי הפיתוח עם בינה מלאכותית (Prompt Engineering Log) הוא חלק חדש וחשוב בפיתוח תוכנה מודרני. התיעוד כולל רשימה של כל הפרומפטים המשמעותיים ששימושו לבנייתuproject, תיאור של ההקשר והמטרה של כל פרומפט, דוגמאות לפלאטים שהתקבלו ואיך הם שולבו בפרויקט, תיעוד של שיפורים איטרטיביים של פרומפטים לאורץ זמן, ושיטות

עובדת מומלצות (best practices) שהופקו מהניסיון. מבנה מומלץ לティקית הפרומפטים כולל תייקיות נפרדות לפרומפטים של תכנון ארכיטקטורה, ייצור קוד, בדיקות, ותיעוד, עם קובץ סקירה כללי.

9 עלויות ותមhor

הבנת עלויות הפיתוח והתפעול היא חיונית לתכנון נכון של הפרויקט ולקבלת החלטות מושכלות לגבי משאבם וטכנולוגיות.

9.1 ניתוח עלויות

ניתוח עלויות (Cost Breakdown) של שימוש ב-API Tokens כולל ספירה מדוקית של tokens בכניסה והן ביציאה (input/output tokens), חישוב העלות למיליאן tokens (per Mtokens) לפי התעריפים של כל ספק שירות, והערכת העלות הכוללת לפי מודל ושירות. הדוגמה הבאה מציגה ניתוח עלויות טיפוסי:

טבלה 1: ניתוח עלויות API Tokens

Total Cost	עלות כוללת / Output Tokens	Input Tokens	מודל / Model
\$45.67	523,000	1,245,000	GPT-4
\$32.11	412,000	890,000	Claude 3
\$77.78	935,000	2,135,000	סה"כ / Total

אסטרטגיית אופטימיזציה כוללת הפחתת שימוש ב-tokens באמצעות סיכום וקיצור של פרומפטים, שימוש ב-batch processing לעיבוד מרובה ויעיל יותר, ובבחירה מודלים לפי יחס עלות-תועלת (cost-effectiveness) כאשר מודלים זולים יכולים לתת תוצאות מספקות.

9.2 ניהול תקציב

ניהול תקציבiesel כולל תחזית עלויות לסקירה עתידית כדי להבין את העליות הצפויות כאשר המערכת גדלה, ניטור (monitoring) של שימוש בזמן אמיתי להזנות חריגות מוקדם, והגדלת התראות על חריגת מתќציב כדי למנוע הוצאות לא מתוכננות.

10 הרחבה ותחזוקתיות

תכנן מערכת שנייה להרחבת ולתחזק בקלות הוא השקעה לטווח ארוך שמשתלמת כאשר המערכת צריכה להפתח ולהשתנות.

10.1 נקודות הרחבה

ארQUITECTURA TOSPFIM (Plugins Architecture) מאפשרת הוספה פונקציונליות חדשה ללא שינוי בקוד הליבה. זה מושג באמצעות הגדרת ממשקים (interfaces) ברורים להרחבה

המגדירים את החוצה בין הליבה לתוספים, הוספת נקודות חיבור (lifecycle hooks) כגון beforeCreate, afterUpdate middleware לעיבוד שרשרת של בקשות, ועיצוב מבוסס-API (API-first design) שמבטיח שכל הפונקציונליות נגישה דרך ממשקים מוגדרים היטב.

תיעוד הרחבה כולל הדרכה מפורטת לפיתוח plugins, דוגמאות לתוספים פשוטים ומורכבים, ותיאור של כללי ונהלים (conventions) להרחבה בטוחה שלא תשבור את המערכת.

10.2 תחזוקתיות

קוד ניתן לתחזקה (Maintainable Code) מאופיין במודולריות (Modularity) והפרדת אחריות (separation of concerns) כך שכל חלק בקוד אחראי על דבר אחד בלבד, שימוש חוזר (Reusability) של קומponentות כך שקוד כתוב פעם אחת ומשמש במקומות רבים, ניתנות לניתוח (Analyzability) כך שקל להבין את הקוד ולזהותו בעיות, וניתנות לבדיקה (Testability) כך שקל לכתוב בדיקות אוטומטיות לקוד.

11 תקני איכות בינהוומיים

תקן ISO/IEC 25010 [22] מגדיר מודל מكيف לאיכות תוכנה המכסה שמות מאפייני איכות עיקריים. כל מאפיין מחלק למאפייני משנה המאפשרים הערכה מפורטת ואובייקטיבית של איכות המוצר.

11.1 מאפייני איכות מוצר

התאמה פונקציונלית (Functional Suitability) בודקת עד כמה המערכת עומדת בדרישות הפונקציונליות, כולל שלמות (Completeness) של כיסוי כל התכונות הנדרשות, נכונות (-Correctness) של התוצאות, והתאמה (Appropriateness) למשימות שהמערכת אמורה לבצע. יעילות ביצועים (Performance Efficiency) מעריכה את התנהלות הזמן (Time behavior) כולל זמני תגובה, ניצול משאבים (Resource utilization) כולל זיכרון ומעבד, ויכולת (Capacity) להתמודד עם עומסים גדולים.

תאימות (Compatibility) בוחנת יכולת פעולה הדדית (Interoperability) עם מערכות אחרות ודו-קיום (Coexistence) במקביל למערכות אחרות. שימושיות (Usability) כוללת קלות למדיה, יכולת הפעלה, נגישות, הגנה מפני שגיאות משתמש, ואסתטיקה של הממשק. אמינות (Reliability) מודדת בשלות המערכת (Maturity), זמינות (Availability), סובלנות (Recoverability), ויכולת התאוששות (Fault tolerance).

אבטחה (Authenticity) כוללת סודיות (Confidentiality), אינטגריות (Integrity), שמאות (Security) Accountability, וא-הכחשה (Non-repudiation). תחזוקתיות (Maintainability) מעריצה מודולריות, שימוש雄厚, ניתנות לניתוח, ניתנות לשינוי (Modifiability), וניתנות לבדיקה. לבסוף, ניידות (Portability) בוחנת התאמה (Adaptability) לנסיבות שונות, ניתנות להתקנה (Installability), וניתנות להחלפה (Replaceability).

12 רשימת בדיקה סופית

לפני הגשת הפרויקט, חשוב לעבור על רשימת בדיקה מקיפה כדי לוודא שכל הדרישות מולאו. התיעוד צריך לכלול מסמך PRD מפורט עם כל המרכיבים, תיעוד ארכיטקטורה עם תרשימי בלוקים ברורים, קובץ README מקיים בرمת מדריך משתמש מלא, תיעוד API מלא לכל המשקיפים הציבוריים, וספר פרומפטים מתועד. הקוד עצמו צריך להיות מאורגן במבנה פרויקט מודולרי ומסודר, עם קבצים שלא עולים על 150 שורות, הערות קוד מקיפות ו-docstrings לכל פונקציה ומחלקה, ועקבות בסגנון הקוד לאורך כל הפרויקט.

הكونfigורציה צריכה להיות נפרדת מהקוד, עם קבצי דוגמה כמו example..env, ללא מפתחות API בקוד המקורי, ועם gitignore. מעודכן. הבדיקות לצריכה לכלול unit tests עם כיסוי של לפחות 70%, תיעוד מקרי קיצון (edge cases), טיפול מיידי בשגיאות (error handling), ודוחות בדיקה אוטומטיים. החלק המחקרי צריך לכלול ניסויים עם שינוי פרמטרים, ניתוח רגשות מתועד, לחברת ניתוח עם גרפים ממחישים, ונוסחאות מתמטיות כאשר רלוונטי.

הויזואלייזציה צריכה לכלול גרפים איקוטיים של תוצאות, צילומי מסך של ממושך המשמש, ותרשיimi ארכיטקטורה ברורים. ניתוח העליות צריך לכלול טבלת שימוש tokens, ניתוח עליות מפורט, וסטרטגיית אופטימיזציה. ההרחבה צריכה לכלול נקודות הרחבה (extension points) מתועדות, דוגמאות לפיתוח TosPFS (plugins), וממשקים ברורים להרחבה. לבסוף, יש לוודא היסטוריות Git מסודרת, רישון מצורף, ייחוס לספריות צד שלישי, והוראות התקנה והפעלה (deployment).

13 מקורות ותקנים נוספים

לצורך הכנת פרויקט ברמתมาตรฐาน, מומלץ להתייחס לתקנים ומקורות בינלאומיים מוכרים. אלה כוללים את תכנית אבטחת איקות התוכנה של MIT [23], מודל איקות התוכנה ISO/IEC 25010 [22], שיטות העבודה ההנדסיות של Google [24], ההנחיות ל-API של Microsoft [25] וההיוריסТИקות לשימושות של נילסן [21]. מקורות אלה מספקים בסיס מוצק לעבודה מקצועית וקדמית ברמה גבוהה ביותר.

14 הערכה חשובה

מסמך זה מציג רמת מצוינות גבוהה במיוחד. לא כל סעיף הוא מחויב במלואו, אך ככל שייתר קритריונים מתקיימים, כך הציון והערכת האיקות יהיו גבוהים יותר. התמקד בעומק, במקצועיות ובהדגמת יכולות מחקר ברמה אקדמית גבוהה. מומלץ להשתמש בכלים LLM לעזרה בהשלמת הפרויקט. מובהר כי חלק מהבדיקה יתכן וייעשה שימוש בסוכני AI לביצוע הבדיקה.

15 English References

- 1 MIT ACIS, *Mit software quality assurance plan*, <https://acisweb.mit.edu/acis/sqap/sqap.r1.html>, 2022.
- 2 A. Downey, *Software engineering practices for scientists*, <http://allendowney.blogspot.com/2013/05/software-engineering-practices-for.html>, 2013.
- 3 Monday.com, *Prd template - product requirement document*, <https://monday.com/blog/rnd/prd-template-product-requirement-document/>, 2024.
- 4 Miro, *Modular prd template*, <https://miro.com/templates/modular-prd/>, 2024.
- 5 Aha! *What is a good product requirements document template*, <https://www.aha.io/roadmapping/guide/requirements-management>, 2024.
- 6 Pacific Certification, *Iso 25010 software product quality model*, <https://blog.pacificcert.com/iso-25010-software-product-quality-model/>, 2024.
- 7 ISO 25000, *Iso 25010 standards overview*, <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>, 2024.
- 8 ISO, *Iso/iec 25010:2011 systems and software quality requirements and evaluation*, <https://www.iso.org/standard/35733.html>, 2011.
- 9 Archbee, *Readme files guide*, <https://www.archbee.com/blog/readme-files-guide>, 2024.
- 10 GitHub, *About readmes*, <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes>, 2024.
- 11 Daily.dev, *10 code commenting best practices for developers*, <https://daily.dev/blog/10-code-commenting-best-practices-for-developers>, 2024.
- 12 Stack Overflow, *Best practices for writing code comments*, <https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/>, 2021.
- 13 Codacy, *Code documentation best practices*, <https://blog.codacy.com/code-documentation>, 2024.
- 14 Hoop.dev, *Api security best practices: Protecting secrets with environment variables*, <https://hoop.dev/blog/api-security-best-practices-protecting-secrets-with-environment-variables/>, 2024.

- 15 Claude Support, *Api key best practices: Keeping your keys safe and secure*, <https://support.claude.com/en/articles/9767949-api-key-best-practices>, 2024.
- 16 OpenAI, *Best practices for api key safety*, <https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety>, 2024.
- 17 PractiTest, *Test coverage metrics*, <https://www.practitest.com/resource-center/blog/test-coverage-metrics/>, 2024.
- 18 Google Testing Blog, *Code coverage best practices*, <https://testing.googleblog.com/2020/08/code-coverage-best-practices.html>, 2020.
- 19 UX Design, *Measuring design quality with heuristics*, <https://uxdesign.cc/measuring-design-quality-with-heuristics-44857efa514>, 2024.
- 20 J. Nielsen, *10 usability heuristics for user interface design*, <https://www.nngroup.com/articles/ten-usability-heuristics/>, 1994.
- 21 J. Nielsen, *10 usability heuristics for user interface design*, <https://www.nngroup.com/articles/ten-usability-heuristics/>, 1994.
- 22 ISO, *Iso/iec 25010:2011 systems and software quality requirements and evaluation*, <https://www.iso.org/standard/35733.html>, 2011.
- 23 MIT ACIS, *Mit software quality assurance plan*, <https://acisweb.mit.edu/acis/sqap/sqap.r1.html>, 2022.
- 24 Google, *Google engineering practices documentation*, <https://google.github.io/eng-practices/>, 2023.
- 25 Microsoft, *Microsoft rest api guidelines*, <https://github.com/microsoft/api-guidelines>, 2023.