מפרט תכן תוכנה (SDS)

שימו לב שמסמך זה עלול להתעדכן בהמשך וכן שמסמך הדרישות יכול להשתנות כתוצאה מהעבודה על מסמך זה. מתוך הנחה שלאחר שלב זה אתם כבר מבינים יותר טוב את תכולות הפרויקט, אתם מתבקשים גם לעדכן את רשימת הסיכונים לפרויקט.

תוצרי מסירה

1) מסמך SDS

המכיל תכנון של ארכיטקטורת המוצר שלכם ע"י דיאגראמות UML:

- דיאגראמות הפצה (Deployment),
- דיאגראמות מחלקה (Class)
- דיאגראמות רצף (Sequence).

כמו כן במסמך זה תתחילו כבר לתכנן את הבדיקות הנדרשות מהמוצר ואת התיעוד שיימסר ללקוח (בדרך זו תוכלו בשלב הבא להעריך יותר נכון את המאמץ הכולל הנדרש).

2) סיכום סקר תיכון שנעשה עם נציגי הלקוח (תבנית בהמשך המסמך), הסיכום יכלול את מועד ומקום הפגישה, משתתפים, נקודות שעלו וטבלת משימות הנובעות מהן.
3) עדכון רשימת הסיכונים במסמך תכנון הפרויקט, כך שיכיל טבלה ממוינת לפי חומרת הסיכון, כאשר לכל סיכון מופיעים הפרטים הבאים:
   - הסיכון
   - רמת השפעתו אם יתרחש (גבוה, בינוני, נמוך)
   - רמת הסיכוי שיתרחש (גבוה, בינוני, נמוך)
   - צעדים הנלקחים כדי להנמיך את הסיכון
   - תוכנית חליפית למקרה שיתרחש

מפרט תיכון תוכנה - SDS

מידע כללי

שם הפרויקט:	פרויקט 2 - קוונטיזציה
צוות הפרויקט	ברכה רוטשטיין, אילה לרדו, איילת רפפורט, הדסה בלאק, שרה עייאש, מיכל בקשי.
מסמכים מצורפים:	
מסמכים קשורים:	הצעת פרויקט: https://www.luminpdf.com/viewer/5dd3cc731731380019b2bbd4 מפרט דרישות: https://docs.google.com/document/d/1phZDGuBHouiVsy9FNBy1R5igO-Xwn8SW4yKHOw_vkSs/edit?ts=5dd11c5e#

תוכן העניינים

הסטורית שינויים

גרסה	תאריך	תאור השינוי	מקור השינוי
1.0	8/11/2009	גרסה התחלתית	לא ישים

1. הקדמה

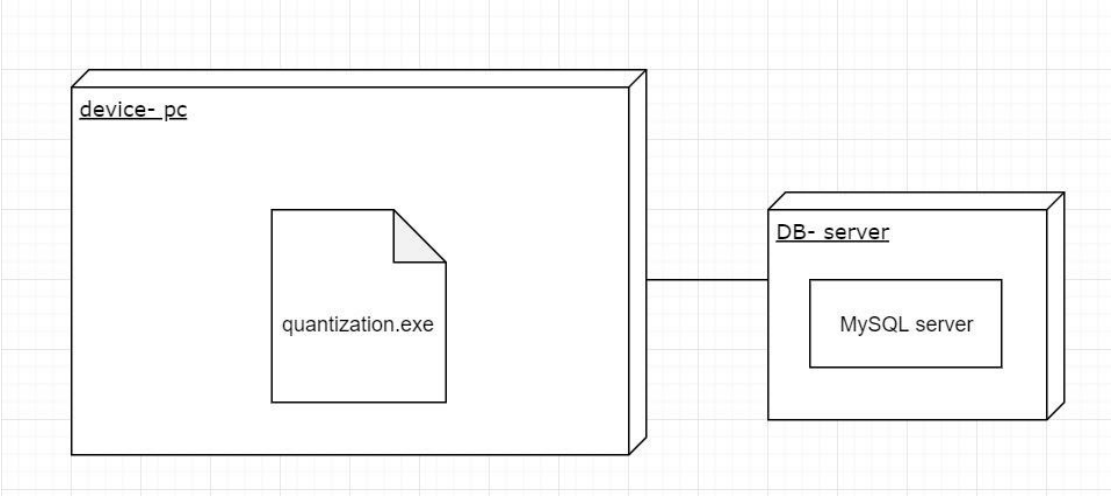
7.1 ארגון המסמך

התוכן העיקרי של המסמך הינו הגדרת המערכת בהתאם לדרישות שפורטו במסמך SRS.
באמצעות תרשימים, נוכל לתכנן את הרכיבים השונים של המערכת ואת הקשרים ביניהם.
וכך יתאפשר לנו לעבור באופן ישיר לשלב המימוש והבדיקות.

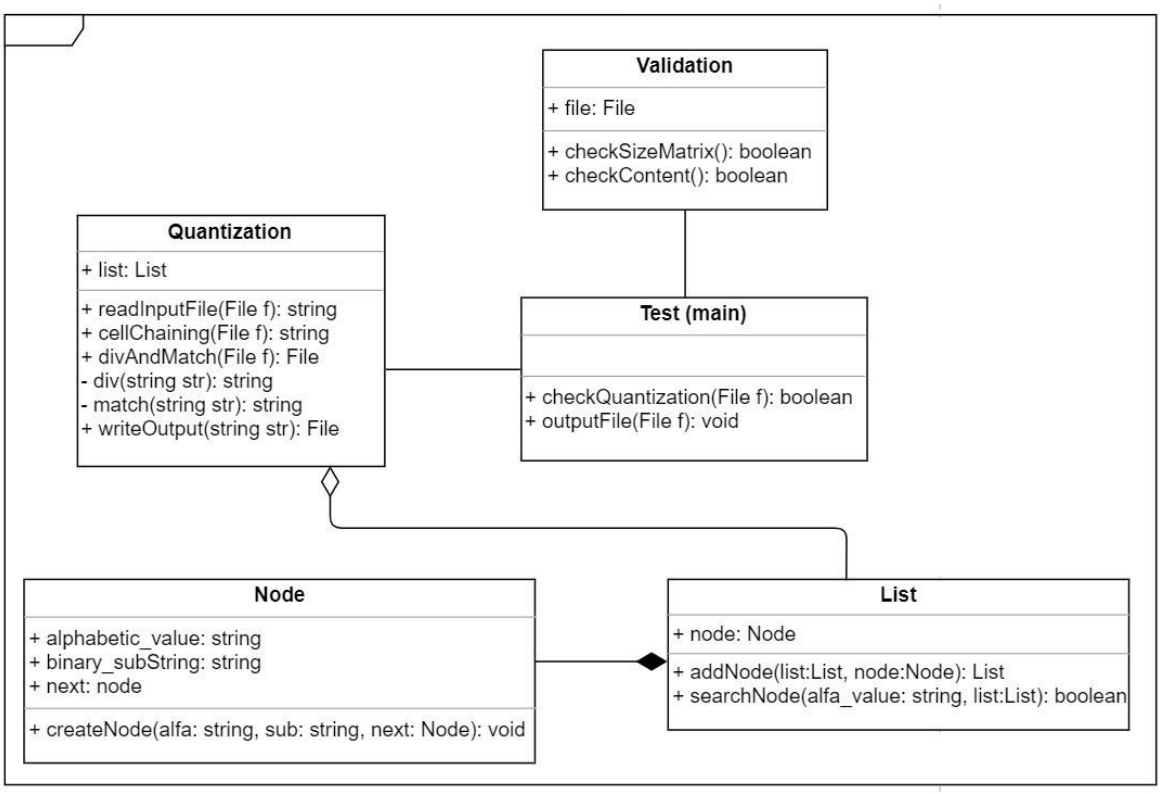
7.2 מטרות המסמך

1. הכנה למימוש המוצר - דיאגרמת מבנה סטטי תגדיר את המחלקות(אלו הם רכיבי המערכת) שיהיו לנו בקוד, ואת סוג האינטראקציה שביניהם.
2. הגדרת סדר פעולתה של המערכת בזמן ריצה - באמצעות דיאגרמת הרצף.
3. בדיקת נכונות המערכת.

2. ארכיטקטורת הפצה – Deployment Diagrams



3. תרשימי מבנה סטטי – Class Diagrams



פירוט על המחלקות:

הכנת תשתית של מבני הנתונים שדרושים לקוונטיזציה:

1. מחלקת NODE : מייצגת התאמה של תת מחרוזת בינארית לערך אלפבתי ספציפי ומצביע לאיבר הבא.

   הפונקציות:

   בנאי, GETTER, SETEER.

2. מחלקת LIST: מייצגת רשימה מקושרת של NODES, תוך כדי מעבר על המחרוזת הבינארית מוסיפים איברים לרשימה.

   הפונקציות:

   הוספה: מקבלים ערך בינארי וערך אלפבתי, יוצרים ממנו NODE ומוסיפים אותו לרשימה.

   חיפוש של איבר: מקבל ערך בינארי - ומחזיר TRUE כאשר הערך הבינארי הזה נמצא באיבר כלשהו ברשימה המקושרת.

חיפוש מחרוזת: הפונקציה מקבלת מחרוזת ואם המחרוזת נמצאת אז היא מחזירה את הערך האלפבתי של המחרוזת

פונקצית שחרור: דואגת לשחרר את ההקצאות שהוגדרו.

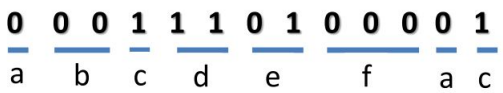
3. מחלקת Quntization: הקוונטיזציה בעצמה:

הפונקציות:

פונקציה אסקי: מקדמת את ה'מונה האסקיי':

יהיה לנו משתנה מסוג STRING שבכל פעם יכיל את הערך האלפבתי שבדיוק קראנו עכשיו, אם לדוגמה נסתכל על הדוגמה שבמצגת של הצעת הפרויקט:

String (bin representation):



symbol	code
a	'0'
b	'00'
c	'1'
d	'11'
e	'01'
f	'000'

11 / 18

בתחילה המונה היה A , אחר כך B .... עד F

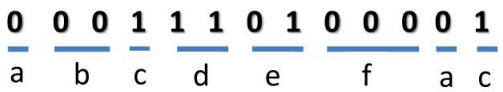
אבל לאחר שנסיים לקרוא עד Z, נעבור לרצפים של AA,AB, AC, AD...AZ ואחר כך BA,BB,BC...BZ וכך הלאה....

הנחה: יש לנו מחרוזת בינארית ארוכה של אפסים ואחדות, אחר כך נממש את השרשור ומחיקת רווחים.

- הפונקציה divAndMatch -
  קלט: STRING של אפסים ואחדות- המחרוזת הבינארית.
  תיאור: זאת הפונקציה שמבצעת קריאות לDIV ולMATCH.
  קודם קוראים לDIV, עם ערך החזרה של DIV קוראים ל MATCH.

  - div
  קלט: מחרוזת STRING הבינארית, אינדקס מאיפה להתחיל לקרוא.
  תיאור הפונקציה:
  לחתוך את תת המחרוזת הבינארית שעוד לא קיימת ברשימה המקושרת, אלא אם כן אנחנו בסוף המחרוזת ואז יש מצב לעשות התאמה חוזרת, כמו בדוגמה בהצעת הפרויקט שבה יש את A ואת C פעמיים בגלל הסוף...

String (bin representation):



symbol	code
a	‘0’
b	‘00’
c	‘1’
d	‘11’
e	‘01’
[illegible]	‘000’

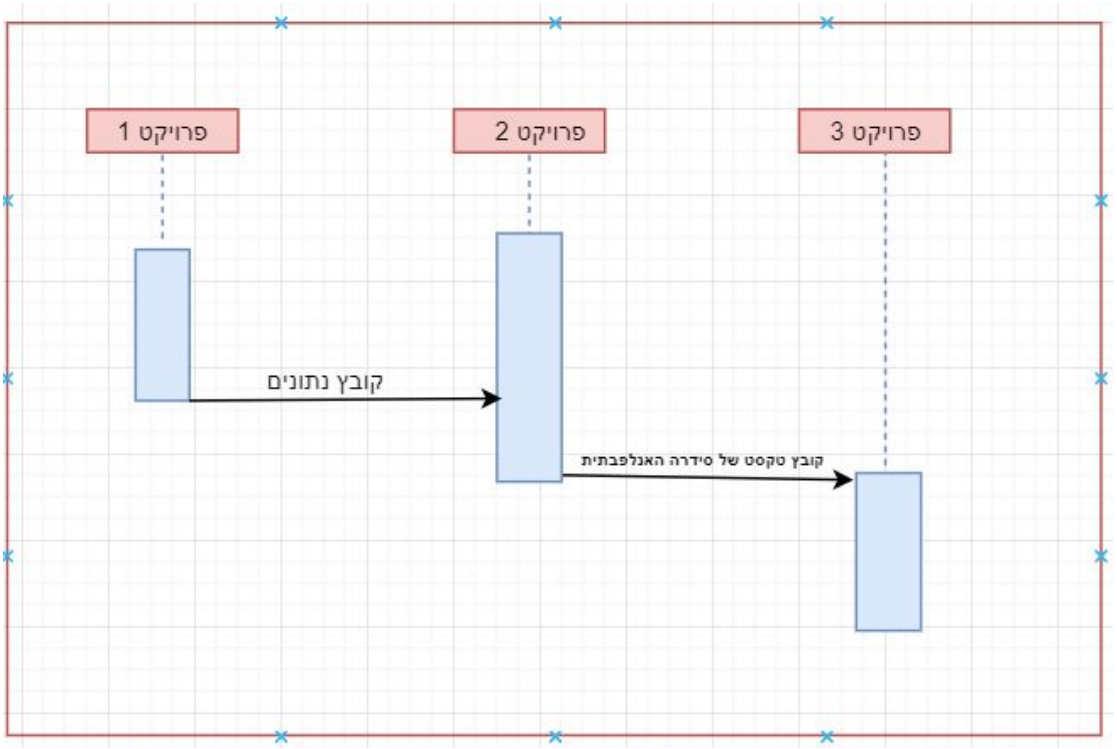
11 / 18

להחזיר את תת המחרוזת הבינארית שיש לקרוא, כלומר הפונקציה תקרא בתחילה תו בינארי בודד ותעבור על הרשימה המקושרת ותחפש אם התו הבודד הזה כבר קיים ברשימה המקושרת, אם כן תקרא תת מחרוזת גדולה יותר, וכך הלאה עד שנקבל תת מחרוזת שעוד לא קיימת ברשימה.
לאחר מעבר על הרשימה המקושרת נקבל תת מחרוזת לה יש לבצע התאמה.

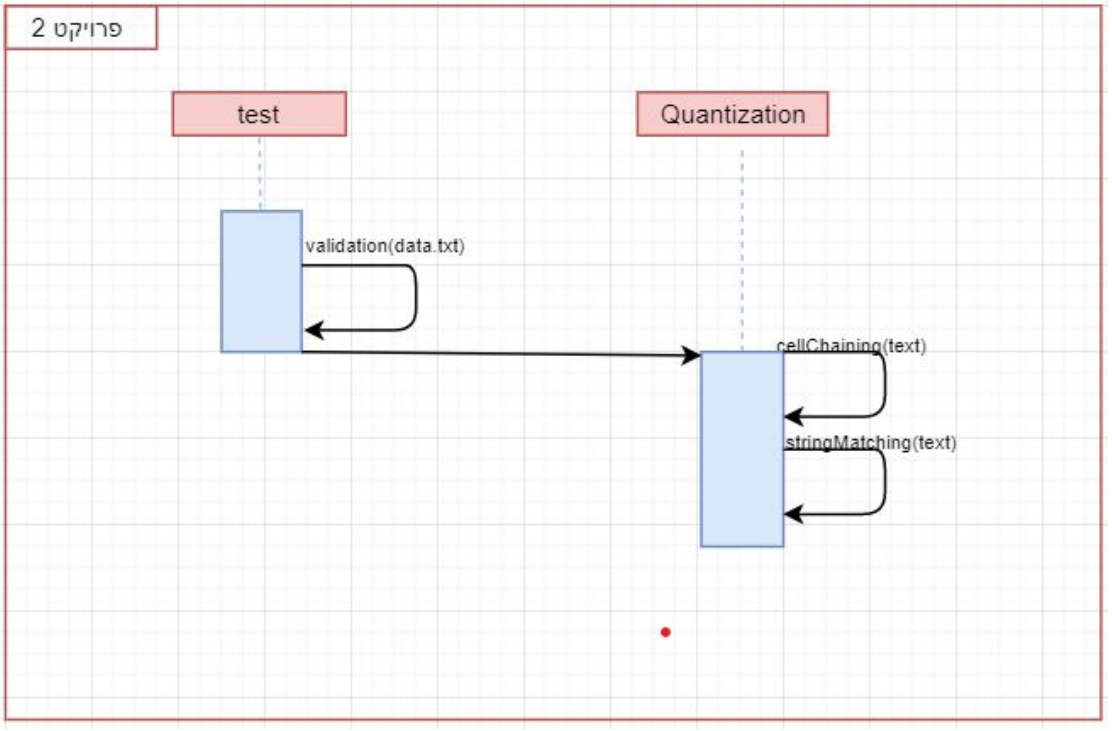
- match
קלט: תת מחרוזת בינארית.
תיאור:בהתאם לערכו של ה"מונה האסקי" אותו אנו מחזיקים נתאים ערך מעל הא"ב למחרוזת שהתקבלה מהפונקציה Div ויש לעדכן את הרשימה ב NODE חדש שנוצר.

4. תרשימי רצף התנהגותי – Sequence Diagrams

<u>כללי</u>:


פרויקט 1
פרויקט 2
פרויקט 3
קובץ נתונים
קובץ טקסט של סידרה האנלפבתית

<u>פרטי</u>:


פרויקט 2
test
Quantization
validation(data.txt)
cellChaining(text)
stringMatching(text)

5. שמירת נתונים - Persistence

הנתונים של המערכת נשמרים בקובץ טקסט (.txt),

יווצר קובץ טקסט המכיל את המחרוזת האלפאבתית.

6. טבלת עקיבות לדרישות

דיאגרמה:	עונה על הדרישה:
7. ארכיטקטורת הפצה – Deployment Diagrams	המערכת תקבל קלט ממסד נתונים ותייצר קובץ הרצה.
8. תרשימי מבנה סטטי – Class Diagrams	פיתוח מודולרי באמצעות שפת C++, OOP.
9. תרשימי רצף התנהגותי – Sequence Diagrams	כללי: אינטראקציה בין צוותי המערכת- פרויקט 2 מקבל קלט מפרויקט 1 ומעביר פלט לפרויקט 3. פרטי: אינטראקציה בין רכיבי המערכת בזמן ריצה- בדיקת חוקיות הקלט, ואלידציה, העברת קובץ נתונים למחלקת המימוש ובדיקת האלגוריתם הממשמ.

7. תוכנית בדיקות ותיעוד

7.1 תוכנית בדיקות

1. <u>תכנון בדיקות יחידה</u>

<u>א. מהי הבדיקה (מטרה וכיסוי)</u>

מטרת בדיקות היחידה היא לבודד כל חלק, ולהראות שכל חלק כזה עובד באופן תקין כאשר הוא עצמאי.
בדיקות היחידה מספקות סיכום חד משמעי שהחלק עומד בתנאים בהם הוא אמור לעמוד.

נבצע בדיקות בכל שלב באלגוריתם,
שלבי האלגוריתם:

1. במחלקה Test מחלקה שבודקת את תקינות המחרוזת שנקראה מהקובץ:

- בפונקציה checkSize- יש לבדוק שגודל הקובץ תואם למטריצה בגודל שבין 250 ל1500 תאים .
(סדר גודל של מטריצה שנע בין בערך 16X16 לבין 40X40).
הפונקציה תחזיר true במקרה והגודל תקין אחרת נקבל false ונשלח הודעה לפרויקט 1 כי זו שגיאת פלט שלהם.
- בפונקציה checkChars- יש לבדוק שהקובץ מכיל רק אפסים ואחדות.

הפונקציות יחזירו true במקרה והגודל / סוג התוים תקין אחרת נקבל false ונשלח הודעה לפרויקט 1 כי זו שגיאת פלט שלהם.

2. במחלקה Quantization נבצע התאמה לתת מחרוזת ערכים אלפאבתים.

המחלקה מכילה מבנה שהוא רשימה שתכיל את כל הערכים האלפאבתים מותאמים לתת מחרוזת בינארית.

- בmain- נקרא מקובץ טקסט (לאחר שתקינותו נבדקה במחלקה Validation) את כל התווים ונשרשר למחרוזת.
- הפונקציה divAndMatch - תקבל את המחרוזת (יש לבדוק שהמחרוזת לא ריקה) ותבצע התאמת ערכים אלפאבתים לתת מחרוזת.

נחזיק רשימה מקושרת בגודל דינאמי שתכיל בתוכה מידע על תתי מחרוזות בינאריות שכבר קראנו.
בכל פעם נעבור על רשימה זו ונבדוק אם כבר קראנו את תת המחרוזת שעליה אנו עובדים כרגע, אם כבר קראנו נמשיך לקרוא עוד תו, כלומר נעבור לתת מחרוזת בינארית גדולה יותר,
וכן יש לדעת מהו הערך האלפאבית הבא הפנוי להתאמה, ואת זה נשמור על ידי 'מונה' אסקי.
תהליך הקוונטיזציה יהיה ע"י שימוש בשתי פונקציות:

div - לאחר מעבר על הרשימה המקושרת נקבל תת מחרוזת לה יש לבצע התאמה.

match - בהתאם לערכו של ה"מונה האסקי" אותו אנו מחזיקים נתאים ערך מעל הא"ב. יש להוסיף ערך למבנה הנתונים List ונשתמש בפונקציות להוספת Node.

בעצם במחלקה זו נבצע בדיקות כך שבהנתן קובץ טקסט תקין אכן נבנה מילון (בתהליך דינמי) ונשמר בקובץ טקסט.

3. במבנה List יש רשימה מקושרת של Node ושתי פונקציות:
   1. insert- תקבל את הNode שדרוש להוסיף.
      יש לבדוק:
      - אם ה Node= Null נשלח הודעה מתאימה.
      - במקרה שהרשימה ריקה - נוסיף לראש את הNode.
      - במקרה שהרשימה לא ריקה - נחפש אם הNode שנרצה להוסיף קיים כבר. אם קיים לא נוסיף. אחרת, נוסיף לסוף.
   2. searchNode - תקבל ערך אלפביתי והרשימה ותחזיר true / false אם הערך קיים או לא.
      יש לבדוק:
      - אם הערך האלפביתי null נשלח הודעה מתאימה.
      אחרת הפונקציה תבצע חיפוש הערך ותחזיר true / false בהתאם.
      - אם הרשימה ריקה.

   שרשור הנתונים במטריצה למחרוזת בינארית ארוכה - בדיקה שהמחרוזת המתקבלת לא מכילה רווחים.
   3. **searchStr** - תקבל מחרוזת בינרית ואם היא קיימת אז היא תחזיר אותה בצורה אלפבתית
4. התאמה לתת מחרוזת ערכים אלפאבתים

נחזיק רשימה מקושרת בגודל דינאמי שמכילה בתוכה מידע על תתי מחרוזות בינאריות שכבר קראנו
בכל פעם נעבור על מערך זה ונבדוק אם כבר קראנו את תת המחרוזת שעליה אנו עובדים כרגע, אם כבר קראנו נמשיך לקרוא עוד תו, כלומר נעבור לתת מחרוזת בינארית גדולה יותר,
וכן יש לדעת מהו הערך האלפאבית הבא הפנוי להתאמה, ואת זה נשמור על ידי 'מונה' אסקי.

ב. <u>אופן הפיתוח ועל ידי מי</u>

ג. תדירות שימוש

כאשר מסיימים לכתוב קוד עבור חלק מסוים נבצע עבורו את בדיקת היחידה.

2. <u>תכנון בדיקות מערכתיות</u>

א. <u>מהי הבדיקה (מטרה וכיסוי)</u>

לוודא את תאימות המערכת לאפיון ולוודא את פעילותה התקינה של המערכת.
יש לתת את הדעת על השאלות הבאות:

- האם המערכת השיגה את המטרות והיעדים שהוגדרו לה באפיון.
- האם המערכת פתרה את הבעיות שהיה עליה לפתור
- האם המערכת כוללת את כל התכולות שסוכמו בסקר/י האפיון.

נבדוק באופן כללי שהפרויקט מקבל קובץ טקסט המכיל מידע לגבי פרוסה מפרויקט 1, מבצע את כל השלבים ומצליח להעביר לפרויקט 3 פלט של מבנה נתונים המכיל ערכים אלפאבתים.

ב. <u>אופן הפיתוח ועל ידי מי</u>

ג. תדירות השימוש

כאשר כל החלקים שבפרויקט מוכנים, וכן כל חלק נבדק בבדיקות היחידה נבצע את בדיקות המערכת.

3. <u>תכנון בדיקות שמישות</u>

מטרת בדיקה זו הנה להעריך האם ה- MMI יעיל, אסוציאטיבי, קל ונוח לשימוש, כך שיוכל להשתלב בקלות בתהליך עבודת המשתמשים ולא יכביד עליהם בשל בעיות עיצוב של ממשק לא נוח או לא ברור.

פן נוסף בבדיקות שימושיות הוא בדיקות סובלנות (טולרנטיות) של המערכת: בדיקות אלו ממחישות את מידת התאמתה של המערכת לעבודתו של המשתמש. מערכת עמוסה בתהליכים, מסובכת ועם תגובות לא ידידותיות לעבודת המשתמש ובמיוחד לטעויותיו, תיזנח ע"י המשתמש לטובת מערכות קיימות אחרות או לחילופין תקטין את שימושו בה.

4. <u>מנגנון המעקב אחרי תקלות והשימוש בו</u>

7.2 תוכנית תיעוד

מדריך לפרוייקט 3:

קובץ טקסט שמכיל בתוכו ערכים אלפאביתים שהתקבלו ע"י תהליך קוונטיזציה מקובץ הנתונים של פרוייקט 1- המרכזת תקינות כל DIE לקובץ הנתונים.

סיכום סקר תיכון תוכנה - SDS

תוכן

שם הפרויקט	פרוייקט 2
לקוחות	מר גיא
מועד ומקום הפגישה	בנין הדר, הר חוצבים
שמות משתתפים	אילה לרדו, ברכה רוטשטיין,איילת רפפורט, שרה עייאש, ד"ר לשם

נקודות שעלו במפגש

1.	שאלות לגבי האלגוריתם בו נשתמש
2.	גודל המטריצה
3.	אופן גישה לקלט מפרויקט 1

טבלת משימות הנובעות מהמפגש

נושא	פעולה נדרשת	באחריות
אופן גישה לקלט מפרויקט 1	צריך לקבוע עם פרויקט 1 בין 2 האופציות הבאות: את קובץ הטקסט (שמהווה קלט עבורנו) הם שומרות במסד הנתונים ואנחנו נפנה לשם כאשר נקבל עדכון לגבי זה שיש קלט למערכת, או שבשביל שהכל יהיה רציף לגמרי, שלא ישמרו את קובץ הטקסט אלא יעבירו לנו אותו ישירות...	