**<Leprecoin>**

מסמך עיצוב

<אילן נוה>

<גרסה 1>

<תאריך>

**היסטוריית גרסאות המסמך**

|  |  |  |
| --- | --- | --- |
| **תאריך** | גרסה | **תקציר השינויים** |
| 28.5.19 | 1 |  |
|  |  |  |
|  |  |  |
|  |  |  |



**1. הקדמה**

החלק הזה ישמש להצגה כללית של מסמך העיצוב. הסבר כללי, וקישור למסמכים קודמים כמו מסמך האפיון עליו אנו מתבססים

1.1 מטרה

*המטרה של המסמך היא להסביר על הפרויקט, על עיצובו ועל איך הפרוטוקול בוא עובד.*

1.2 המוצר

* *המוצר נקרא Leprecoin והוא מטבע דיגיטלי מאובזר קריפטוגרפי*
* *המוצר מאפשר העברת כספים בטוחה וחוקית מסוג המטבע על ידי ממשק ארנק*

1.3 קישור למסמכים קודמים

*ראה מסמך אפיון*

*ראה מסמך חקר*

1.4 הגדרות

*מטבע דיגיטלי ( (Cryptocurrency- המטבע הדיגיטלי מונפק על ידי גוף פרטי כלשהו שמפתח טכנולוגיה שאינה מאפשרת לשכפל אותו. המטבע בדרך כלל אינו שייך לגוף שהנפיק אותו אלא פתוח לכולם. למעשה מדובר בפרוטוקול (תוכנה), שרץ במקביל על אלפי מחשבים ונרשם על ידיהם. מטבע דיגיטלי קריפטוגרפי הוא מטבע שאבטחתו נעשת באמצעות מפתחות פרטיים וציבוריים.*

*בלוק צ'יין - היא תפיסה טכנולוגית במחשוב המאפשרת פעילות עסקית מאובטחת באינטרנט ואימות של טרנזקציות עסקיות בין צדדים שונים ללא צורך בישות ניהול מרכזית. הבלוקצ'יין היא בעצם שרשרת של בלוקים כשכל בלוק מקושר לקודמו ולכל בלוק יש קוד hash יחודי.*

*ביטקוין - הביטקוין הוא מטבע שאינו תלוי בגוף מרכזי. במקום זאת, ביטקוין הוא רשומה בקובץ ציבורי המכונה "שרשרת בלוקים" (בלוקצ'יין), ניתן לנהל אותו או להשתמש בו בעזרת תוכנה המכונה "ארנק". מערכת הביטקוין היא מערכת המבטיחה שהתשלומים יתבצעו באופן תקין כך שסכום שהועבר בתשלום אכן נגרע מארנק המקור והתווסף לארנק היעד.*

*כרייה – הוספת בלוק לשרשרת הבלוקים על ידי תהליך מסובך הכולל חישובים מתמטיים ודורש יכולות מחשוב גדולות*

*מפתח פרטי- מפתח פרטי השייך למי שיצר אותו ורק הוא יכול להשתמש בוא על מנת לחתום מידע.*

*מפתח ציבורי- לכל מפתח פרטי יש מפתח ציבורי שמטרתו שיהיה ידוע לכולם, בעזרת המפתח הציבורי אפשר לאמת את החותם על חתימות דיגיטלות. אי אפשר לחשב את המפתח הפרטי מהציבורי*

*חתימה דיגיטלית- מידע אשר עבר עלי שימוש במפתח פרטי (נחתם על ידו) נקרא חתימה. בעזרת מפתח הציבורי אפשר לאמת אם החותם הוא בעל המפתח הפרטי שמתאים לציבורי.*

*P2P- בניגוד לתקשורת הנפוצה כיום- בין שרת ללקוח. p2p זאת שיטת תקשורת שלא תלויה בשרת שמנהל אותה אלא כל המשתמשים במערכת מקימים אותה. מטבע המשתמש ברשת כזאת נקרא מטבע מאובזר*

1. ארכיטקטורת המערכת

חלק זה כולל את תיאור מבנה המערכת ופירוט המודולים השונים בה

2.1 מבט על

המערכת מחולקת לשלושה חלקים עיקריים: למבנה הנתונים של הבלוקצ'יין, לארנק המשתמש הכולל ממשק גרפי ולמימוש התקשורת באמצעות node.

הבלוקצ'יין היינו שרשרת הבלוקים שמכילה בתוכה את כל העסקאות שנעשו במטבע. בעזרת הפרוטוקול תקשורת לכל המשתמשים ברשת יש אותו עותק של בלוקצ'יין ובכך הם יכולים לאמת עסקאות ולאשר עסקאות חוקיות.

הבלוקצ'יין עצמו הוא רשימה של עצמים מסוג בלוק כאשר כל בלוק מכיל את הhash של קודמו, מידע נוסף (יורחב בהמשך) ואת רשימת העסקאות של אותו הבלוק. כל הבלוקצ'יין מכיל כך, את כל העסקאות שנעשו עם המטבע. הבלוקצ'יין עטוף במחלקה שמקשרת אותו למבנה הנתונים, בוא הוא נשמר בטבלאות בשפת sql. השרשרת נשמרת בטבלאות על מנת לאפשר גישה נוכה לכל בלוק לסידור מתאים של הנתונים. כל ארנק שמכיל את הבלוקצ'יין הוא בעצם מכיל את המחלקה שעוטפת אותו עם בסיס הנתונים ובכך יש לו גישה למידע דרך בסיס הנתונים.

בתחילת ההרצה התוכנה יוצרת את עצם מסוג זה פעם אחת ומעבירה אותו לארנק ולnode.

הארנק הינו הדרך להשתמש במטבע. הארנק מאפשר להעביר את המטבע לארנקים אחרים, ולראות את המטבעות שאתה מקבל. הארנק כולל ממשק גרפי על מנת לאפשר למשתמש להשתמש בוא. הארנק מכיל בתוכו את הבלוקצ'יין, עם גישה לבסיס הנתונים, הוא מוצא על פי המפתח הפרטי שבקובץ את הכתובת של במשתמש, על הפי הבלוקצ'יין והכתובת הוא יכלו למצוא את כמות הכסף שיש לו (ראה מבנה עסקה ומסמך חקר). בנוסף לכך יש לו גישה לעצם מסוג node שאחראי על התקשורת ועל אימות עסקאות.

תקשורת מסוג p2p נבנית על כך שכל מחשב (peer או node) משמשים גם כשרת וגם כלקוח. כל node בתקשורת מסוג זה חייב להתנהג על פי אותם חוקים על מנת שהרשת תהיה אמינה, לכן הם נקראים גם peers (=עמיתים).

לכן על מנת לממש זאת התקשורת בנויה על שלוש מחלקות. המחלקה הראשונה היא השרת, והוא מאפשר על ידי ריצה בthread שונה לקבל כל הזמן הודעות ממשתמשים אחרים. תפקידו העיקרי של השרת הוא לאחסן את ההודעות כך שיהיה גישה אליהם. המחלקה השנייה היא הלקוח, הלקוח הוא בעצם שולח את ההודעות לכתובת שמצוינת לו או לכל הnodes שהוא מודע לקיומם. המחלקה השלישית והמרכזית היא המחלקה node שהיא מריצה את השרת, ואת הלקוח כשצריך. לכן, היא אחראית למימוש של כל peer ולתפקוד התקין שלו. בנוסף לכך, המחלקה מחזיקה את העצם של הבלוקצ'יין (העצם עם מבנה הנתונים) ובכך היא יכולה לאמת עסקאות ובלוקים שהיא מקבלת.

2.2 פירוט רכיבי המערכת

**כאן יופיע פירוט לכל רכיב (ניתן לעשות באיטרציות – כל פעם עבור הרכיבים הרלוונטיים, לפני תהליך הפיתוח שלהם)**  
בסעיף זה יש לפרט את המבנה הפנימי של כל מודול/רכיב - כלומר:

* מבנה המחלקות השונות המרכיבות אותו, המאפיינים שלהם (public/private/protected + אילו טיפוסי משתנים כל פונקציה מקבלת ו/או מחזירה) והקשרים ביניהם (מי קורא למי, משתמש במי ולאיזה צורך) – מומלץ לצייר גרף UML (לא חייבים לפרט משתנים/פונקציות פרטיות אבל שיהיה ברור מה קשור למה ובאיזה אופן או לאיזה צורך)
* יש לתאר את ה- data flow / use case של שימושים שונים בכל תת-רכיב או פונקציה (מה קורה אם מתקבלים סוגים שונים של ערכים, איך/מדוע/באילו מקרים הרכיבים מתנהגים בצורה מסוימת או אחרת וכן הלאה)

**המלצות:**

* כדאי לתכנן רכיבים תשתיתיים עבור צרכים גלובליים במערכת - כמו למשל יצירת תקשורת רשתית, כתיבה וקריאה לבסיס נתונים (או קבצים), כתיבת תיעוד (log) הפעולות של המערכת (מקובל להגדיר "רמה" לכל שורה שנכתבת - debug, info, warning, error כדי שיהיה קל לסנן בזמן ריצת התוכנית. בנושא זה – מוזמנים להסתכל על מודול logging בפייתון).
* מומלץ לתכנן את הרכיבים באופן שיאפשר אבסטרקציה ומודולריות – בכדי לאפשר הרחבה עתידית ותחזוקה קלה. אם המימוש הפנימי של רכיב או מודול מסוים משתנה אז שאר הרכיבים/מודולים אינם מושפעים או נפגעים מכך ואין צורך לשנות גם אותם, או כאשר רוצים להוסיף פונקציונליות/רכיב חדש אז אין צורך לשכתב את כל שאר המערכת במיוחד בשבילו.

\*כשכתוב input ו outputבתיאור המחלקות הבאות, הכוונה לא לקלט מהמשתמש ולפלט של תוכנה אלא לinput ולoutput של עסקה. (ראה במחלקות ובמסמך חקר)

\*את התפקידים של הinputs והoutputs ניתן לראות במסמך חקר.

\*הכוונה בפונקציית hash היא בדרך כלל SHA256 למעט, השימוש בRIPEMD160 ביצירת הכתובת של הארנק. (ראה בספרייה pycrypto).

|  |
| --- |
| Block |
| תכונות   * self.number - המספר הסודר של הבלוק * self.nonce – המספר שמחושב עם הערך של הhash על מנת ליצור proof of work * self.prev – ערך הhash של הבלוק הקודם * self.difficulty – ערך הקושי ביצירת proof of work * self.transactions – העסקאות שבבלוק * self.time\_stamp – חותמת הזמן של יצירת הבלוק * self.hash\_code – קוד הhash של הבלוק |
| פעולות   * \_\_init\_\_ - יוצרת את הבלוק, מקבלת את הערכים של כל התכונות של הבלוק. או משתמשת בערכים קבועים.   השימוש בערכים אלו הוא בשביל ליצור בלוק חדש ולא כשמוציאים אותו ממבנה הנתונים   * \_\_str\_\_ - ממירה את הבלוק למחרוזת ומחזירה אותו * hash\_block – מחשבת את ערך הhash של הבלוק * hash\_transactions – מחשבת ומחזירה את ערך הhash של כל העסקאות ביחד * add\_transaction – מקבלת עסקה ומוסיפה אותה לרשימת העסקאות בבלוק * mine\_block – הפעולה כורת את הבלוק על ידי חישוב ערך nonce מתאים * is\_valid\_proof – מחזירה אם יש proof of work תקין על הבלוק * serialize – מחזירה tuple של הבלוק על מנת שיוכל להיכנס בקלות לdatabase * deserialize – פעולה סטאטית שמחזירה עצם בלוק מערך של serialized בלוק ורשימת העסקאות שבו |

|  |
| --- |
| Transaction |
| תכונות   * self.inputs – רשימת הinputs בעסקה * self.outputs – רשימת הoutputs של העסקה * self.transaction\_id – ערך הhash של העסקה |
| פעולות   * \_\_init\_\_ - יוצרת את העסקה. מקבלת רשימה של inputs וoutputs * add\_input – מקבלת input מוסיפה אותו לרשימת הinputs * \_\_str\_\_ - ממירה את העסקה למחרוזת ומחזירה אותה * add\_output – מקבלת output ומוסיפה אותו לרשימת הoutputs * hash\_transaction – מחשבת ומחזירה את ערך הhash של העסקה * is\_valid\_proof – מחזירה אם יש proof of work תקין על הבלוק * deserialize – פעולה סטאטית שמחזירה עצם עסקה מרשימת inputs וoutputs |

|  |
| --- |
| Output |
| תכונות   * self.value – מספר הכסף שמעבירים בעסקה * self.address – כתובת הארנק שאליה מעבירים (לא ip) |
| פעולות   * \_\_init\_\_ - יוצרת את הoutput. מקבלת את התכונות של המחלקה * \_\_str\_\_ - ממירה את הoutput למחרוזת ומחזירה אותה * hash\_output – מחשבת ומחזירה את ערך הhash של הoutput * serialize – מחזירה tuple של הoutput על מנת שיוכל להיכנס בקלות לdatabase * deserialize – פעולה סטאטית שמחזירה עצם output מ- serialized output |

|  |
| --- |
| Input |
| תכונות   * self.transaction\_id – ערך הhash של העסקה שהinput משתמש בה להוכחה * self.output\_index – המספר הסידורי של הoutput להוכחה בעסקה שממנה מוכיחים * self.proof – ההוכחה ליכולת להשתמש בoutput. ההוכחה היא רשימה המכילה בתוכה את החתימה והמפתח הציבורי של החותם. |
| פעולות   * \_\_init\_\_ - יוצרת את הinput. מקבלת את כל התכונות של המחלקה * \_\_str\_\_ - ממירה את הinput למחרוזת ומחזירה אותה * hash\_input – מחשבת את ערך הhash של הinput ומחזירה אותו * add\_transaction – מוסיפה עסקה לרשימת העסקאות בבלוק * mine\_block – הפעולה כורת את הבלוק על ידי חישוב ערך nonce מתאים * is\_valid\_proof – מחזירה אם יש proof of work תקין על הבלוק * serialize – מחזירה tuple של הinputעל מנת שיוכל להיכנס בקלות לdatabase * deserialize – פעולה סטאטית שמחזירה עצם input מערך שלinput serialized |

|  |
| --- |
| UnspentOutput |
| תכונות   * self.output – הoutput הלא מבוזבז * self.transaction\_id – המזהה של העסקה ממנה לקוח הoutput * self.output\_index – המספר הסידורי של הoutput בעסקה ממנה הוא לקוח |

|  |
| --- |
| BlockChain |
| תכונות   * self.chain – רשימת הבלוקים * self.transaction\_pool – רשימת העסקאות שצפויות להיכנס לבלוק החד שייווצר בתהליך הכרייה * self.logger – עצם מסוג logger (ראה ספרייה logging) בשביל הפעולות במחלקה |
| פעולות   * \_\_init\_\_ - יוצרת את הבלוקצ'יין. מקבלת logger ורשימת בלוקים ויוצרת את העצם עם רשימת עסקאות ריקה * add\_new\_block – הפעולה מקבלת את כתובת הארנק של הכורה, כורת בלוק חדש ומוסיפה אותו לשרשרת עם תשלום לכורה * \_\_str\_\_ - ממירה את הבלוק למחרוזת * add\_transaction – מוסיפה עסקה לרשימת העסקאות בשרשרת |

|  |
| --- |
| BlockChainDB |
| המחלקה יורשת מהמחלקה BlockChain  תכונות   * self.logger – עצם מסוג logger בשביל הפעולות במחלקה * self.connection – העצם שאחראי לחיבור של המחלקה למבנה הנתונים (ראה ספרייה sqlite3) * self.cursor – המצביע שנוצר בחיבור, אחראי על ביצוע הפעולות השונות במבנה הנתונים * self.path – מיקום קובץ מבנה הנתונים * עצמים של הטבלאות השונות במבנה הנתונים: טבלת בלוקים, טבלת עסקאות, טבלת inputs וטבלת outputs |
| פעולות   * \_\_init\_\_ - יוצרת את העצם המכיל בתוכו גם את העצם המורש. מקבלת כפרמטרים את עצם הlogger ואת השם של מבנה הנתונים (אם לא מועבר שם אז יש לו ערך קבוע). * create\_connection – יוצר חיבור חדש למבנה הנתונים * close\_connection – סוגר את החיבור למבנה הנתונים * insert\_block – מקבלת בלוק ומכניסה אותו למבנה הנתונים * extract\_block – מקבלת מספר שורה, ומחזירה את הבלוק ממספר שורה זה * extract\_inputs – הפעולה מקבלת את הid של עסקה ומחזירה את כל הinputs השייכים לעסקה זאת * extract\_outputs – הפעולה מקבלת את הid של עסקה ומחזירה את כל הoutputs השייכים לעסקה זאת * extract\_transactions – הפעולה מקבלת מספר בלוק ומחזירה את כל העסקאות השייכות לבלוק זה * extract\_chain – הפונקציה מוציאה ממבנה הנתונים את כל שרשרת הבלוקים * add\_new\_block\_to\_db – הפונקציה מוסיפה לבנה נתונים בלוק חדש שעובר כרייה. הפונקציה מקבלת את הכתובת של הכורה * update\_chain – הפונקציה מעדכנת את רשימת הבלוקים בהתאם למבנה הנתונים * add\_downloaded\_blocks – הפונקציה מקבלת בלוקים שהורדו מהאינטרנט ומוסיפה אותם למבנה הנתונים ולרשימה של הבלוקים |

|  |
| --- |
| Table |
| תכונות   * self.connection – החיבור למבנה הנתונים שאליו שייכת הטבלה * self.curser – המצביע שנוצר בחיבור, אחראי על ביצוע הפעולות השונות במבנה הנתונים * self.table\_name – שם הטבלה * self.structure- מבנה הטבלה |
| פעולות   * \_\_init\_\_ - יוצרת את עצם הטבלה. מקבלת את כל התכונות של המחלקה * create\_table – יוצר את הטבלה אם היא לא קיימת במבנה הנתונים * insert – מקבלת tuple של עצם ומכניסה אותו לטבלה |

|  |
| --- |
| CryptoSet |
| תכונות   * self.private\_key – המפתח הפרטי בסט * self.public\_key – המפתח הציבורי בסט |
| פעולות   * \_\_init\_\_ - יוצרת את הסט. מקבלת את הערך של המפתח הפרטי * sign – מקבלת קוד hash חותמת עליו ומחזירה את החתימה * decrypt– הפעולה מקבלת מידע מוצפן ומפענחת אותו באמצעות המפתח הפרטי * encrypt – פעולה סטאטית המקבלת מפתח ציבורי והודעה להצפין, מצפינה אותה ומחזירה את ההודעה המוצפנת. * verify – פעולה סטאטית המקבלת מפתח ציבורי, קוד hash וחתימה, ומאמתת את החתימה. מחזירה אמת אם היא אמיתי ושקר אחרת. * hash – הפעולה מקבלת הודעה מעבירה עליה פונקצית hash ומחזירה את התוצאה |

|  |
| --- |
| Wallet |
| המחלקה יורשת מהמחלקה CryptoSet  תכונות   * self.address – כתובת הארנק. מתקבלת על ידי העברת פונקציית sha256 על המפתח הציבורי שלו, ועל התוצאה העברת פונקציה ripemd160. * self.block\_chain\_db – עצם מסוג BlockChainDB המכיל את הבלוקצ'יין שהארנק משתמש בוא * self.unspent\_outputs – רשימת הoutputs הלא מבוזבזים. משמשים על מנת לחשב את מאזן הכסף שיש לארנק * self.balance – מאזן הכסף שיש לארנק * self.transactions – כל העסקאות שהעבירו כסף לארנק או שהעביר בהם כסף לארנק אחר * self.logger – עצם מסוג logger בשביל הפעולות במחלקה |
| פעולות   * \_\_init\_\_ - הפעולה יוצרת את הארנק ואת העצם המורש. הערכים שהיא מקבלת הם המפתח הפרטי של הארנק, עצם הבלוקצ'יין והlogger. * new\_wallet – פעולה סטאטית המחזירה עצם מסוג ארנק הנוצר ממפתח ציבורי חדש. הפעולה מקבלת את הערכים של עצם הבלוקצ'יין ואת הlogger.   השימוש בערכים אלו הוא בשביל ליצור בלוק חדש ולא כשמוציאים אותו ממבנה הנתונים   * can\_unlock\_output – הפעולה מקבלת output ומחזירה אמת אם הארנק יכול להשתמש בוא ליצירת input, כלומר לבזבז אותו, ושקר אחרת. * is\_unspent\_output – הפעולה מקבלת את העסקה שהoutput נמצא בא ואת המספר הסידורי שלו בא ומחזירה אמת אם הoutput הוא לא מבוזבז ושקר אחרת * update\_balance – הפעולה מעדכנת את המאזן הכספי של הארנק, בהתאם לשינויים שנעשו בבלוקצ'יין (אם נעשו). * create\_transaction – הפעולה מקבלת כמות כסף לשלוח ואת כתובת הארנק שלוח לה. העסקה מחזירה אמת אם יש לארנק מספיק כסף לעסקה ואת העסקה עצמה. בנוסף לכך, אם העסקה תקינה היא מוסיפה אותה למאגר העסקאות בבלוקצ'יין שלה. אחרת היא מחזירה שקר. * update\_unspent\_outputs – הפעולה מעדכנת את הoutputs הלא מבוזבזים של הארנק, בהתאם לשינויים בבלוקצ'יין (אם נעשו) * update\_transactions– הפעולה מעדכנת את רשימת העסקאות הקשורות בארנק בהתאם לשינויים בבלוקצ'יין (אם נעשו). * find\_address – פעולה סטאטית המחשבת את הכתובת של המחזיק במפתח הציבורי שהיא מקבלת ומחזירה אותה. * equals\_public\_key – הפעולה מקבלת מפתח ציבורי ומחזירה אמת אם הוא זהה (מתאים לאותו מפתח פרטי) למפתח הציבורי של הארנק, ושקר אחרת. |

|  |
| --- |
| NodeServer |
| תכונות   * self.server\_socket – עצם מסוג socket שמשמש את השרת (ראה ספרייה socket) * self.logger – עצם מסוג logger המשמש את הפעולות במחלקה * self.recieved\_messages – רשימת ההודעות שהשרת קיבל והם עוד לא טופלו * self.to\_close – דגל בוליאני המודי האם להפסיק את פעולת השרת * self.readable – רשימת הsockets שמוכנים לקריאה * self.exceptional – רשימת הsockets שאראה שגיאה בקשר איתם. |
| פעולות   * \_\_init\_\_ - הפעולה יוצרת את העצם ואת השרת. הפעולה מקבלת את הlogger. * run – פעולה שמריצה את השרת עד שקופץ הדגל של הסגירה. על מנת תפקוד תקין של התוכנית, הפעולה צריכה לרוץ בthread נפרד מהthread הראשי. * handle\_exception\_sockets – הפעולה מטפלת בsockets שאראה בהם שגיאה. * handle\_read\_sockets – הפעולה מטפלת בsockets שמוכנים לקריאה. * extract\_pack\_length – פעולה סטאטית המקבלת socket של לקוח ומקבלת ממנו את האורך של ההודעה שהוא רוצה להעביר בהתאם לפרוטוקול. הפעולה מחזירה אורך זה. * receive\_message – הפעולה מקבלת socket של לקוח ואת האורך של ההודעה שהוא אמור לקבל ממנו. הפעולה מקבלת ממנו את ההודעה בהתאם לאורכה ומחזירה אותה. * disconnect\_connection – הפעולה מקבלת socket של לקוח ומנתקת אתו באופן תקין את הקשר. * get\_received\_messages– הפעולה מחזירה רשימה של ההודעות שקיבל השרת והם עוד לא טופלו. * remove\_message – הפעולה מקבלת הודעה ומוציאה אותה מרשימת ההודעות שקובלו בשרת. |

2.3 דיון בנושא העיצוב הנבחר

כאן יש לפרט ובעיקר להסביר מדוע בחרתם דווקא בחלוקת הרכיבים/תפקידים הזו ולא אחרת. מה היתרונות שלכם מבחינתה, וגם מה החסרונות שאתם מודעים אליהם. במידה וחשבתם על חלופות אחרות, יש לציין אותן ולהסביר בקצרה כיצד הן שונות מהפתרון שנבחר - ומדוע החלטתם בסופו של דבר שלא לבחור בחלופות האלו.  
כאן יש להתייחס גם לגבי שפת התכנות שבה בחרתם לכתוב כל רכיב, ומדוע דווקא בשפה זו ולא אחרת.

1. עיצוב נתונים ופרוטוקולים

כאן יופיע תיעוד של מבני נתונים שונים / פרוטוקולים המשמשים אותנו במערכת

בסעיף זה יש לפרט את כל סוגי המידע אשר מועבר או נשמר במערכת - בין אם בין רכיבים, בין מודולים או בכל צורה אחרת (כלומר גם אם זה משמש רק לתקשורת פנימית ולא ע"ג הרשת, או למשל אם זה משהו ששומרים לקובץ/בסיס נתונים).

* לכל סוג מידע כזה יש לפרט על השדות אותן הוא מכיל ומאיזה סוג כל שדה, מה טווח הערכים הרלוונטי לגביו וכל הגבלה או מידע נוסף אחר שאתם מוצאים לנכון (למשל: האם מותר שהשדה יהיה ריק ומה המשמעות של זה, אורך או ערך מינימלי/מקסימלי למספר או למחרוזת)
* יש לציין לאיזו מטרה משמש כל מבנה נתונים.
* עבור פרוטוקול תקשורת-נתונים בין רכיבים שונים:
* מה המצבים השונים לכל סוג של תקשורת (למשל אותנטיקציה, הרשמה, שליחת קובץ וכו')
* באילו ייצוגי מידע נעשה שימוש + תרשים זרימה של המצבים השונים שיכולים להיות וסדר השלבים שלהם (למשל כיצד נעביר מידע על מקרים של סיסמה שגויה, משתמש חסום, התחברות מוצלחת וכו')
* מומלץ להגדיר מראש קודים של בקשות/פעולות, תוצאות/תגובות ושגיאות אשר ישותפו בין כך הרכיבים הרלוונטיים
* יש להתייחס לאופי המידע הנשמר (האם הוא טקסטואלי או בינארי) ולייצגו בצורה נוחה והולמת בהתאם
* יש לפרט אודות מבנה בסיס הנתונים (חלוקה לטבלאות, השם והטיפוס של כל עמודה/שדה, קשרים בין טבלאות ואילוצים כלשהם על עמודות)

4. ממשק משתמש

כאן יופיע פירוט ממשקי המשתמשים וכן תופיע הסקיצה עבורם

בסעיף זה יש לפרט את הפונקציונליות של המערכת כפי שהיא מתבטאת עבור משתמש חיצוני. במידה ויש סוגים שונים של משתמשים, יש להתייחס לכולם בהתאם - אילו רכיבים או נתונים רלוונטיים עבורם, וכיצד הם מתקשרים איתם.  
  
בנוסף, יש לצרף המחשות ויזואליות של המסכים השונים, ולהסביר את התוכן שלהם (למשל מה התפקיד של כל שדה או כפתור, מה קשור/תלוי במה - למשל כפתור שמכובה בהתאם לתנאים מסוימים במערכת וכן הלאה) והקשרים ביניהם (איזה מסך מוביל לאיזה מסך ובאילו מקרים)  
  
דוגמה (לא מלאה):  


5. נספחים

**כל דבר שהייתם רוצים להוסיף בנוגע לעיצוב המערכת**