2024

filed of view

אבטחה מובטחת

תמר שושנה הולצר

325984011

סמינר דרכי חנה, אלעד

המנחה- המו' רחל טוג

תאריך ההגשה- תמוז, תשפ"ד. (יולי 2024)

תוכן

1. הצעת פרויקט

נבדק ואושר ע"י משרד החינוך

פרטים של מגיש ההצעה -

סמל מוסד: 711903

שם מכללה : סמינר דרכי חנה

שם סטודנט : תמר שושנה הולצר

תעודת-זהות: 325984011

שם פרויקט: אבטחה מובטחת

פרטי הפרויקט-

תיאור הפרויקט:

בהתאם לנתוני IMS Research נכון לשנת 2013 שוק מצלמות המעקב צפוי לגדול ב-1.5 פעמים או יותר בחמש השנים שבאות אח"כ.

מכאן ניתן לראות כי שוק מצלמות המעקב העולמי גדל במהירות. הסיבה לכך היא שמצלמות המעקב משמשות ליותר מאשר רק מניעה ופתרון פשע או ניהול תעבורה. הם נחוצים כעת עבור פסי ייצור או צפייה באסונות טבע.

יתרה מכך, כיום אפשר לא רק לצפות בתמונות אלא ניתן גם לחלץ מהן את הנתונים הדרושים. יחד עם הצמיחה של שוק מצלמות המעקב, העניין במיקום יעיל של מצלמה גובר גם הוא, אם מיקום המצלמה אינו יעיל, אפילו אם מצלמות רבות יותקנו התוצאה יכולה שלא להשביע רצון, מלבד העובדה שהעלויות תהיינה יקרות בלי שום פרופורציה לצורך האמיתי.

בעיית מיקום המצלמה האופטימלית- נקראת לפעמים בעיית פריסת רשת המצלמה, מוגדרת כאיך למקם את המצלמות בצורה נאותה כדי למקסם את הכיסוי תחת תנאים מסוימים.

האפליקציה שאני מפתחת מקבלת כקלט שירטוט של מבנה תוך סימון האזורים האסורים לצילום. האפליקציה מחשבת וקובעת היכן למקם מצלמות האבטחה בצורה אופטימלית: כך שהאזור יכוסה כנדרש, תוך חסימת אזורים אסורים לצילום במינימום מצלמות או מינימום מחיר. ולבסוף היא מציגה את התוצאה למשתמש.

המטרה שלי בפיתוח האפליקציה לפריסת מצלמות אבטחה בתוך מבנה הינה לחסוך למשתמשים הוצאות יקרות על כמות רבה יותר של מצלמות אבטחה רק מכיוון שהם לא ידעו כיצד לחשב את כמות המצלמות הנדרשות להם. וכן לחסוך למתקיני המצלמות את החישוב המורכב בשביל היעוץ ללקוחות בכמה מצלמות להשתמש וכן היכן להתקין אותן.

<u>הגדרת הבעיה האלגוריתמית:</u>

פרישת מצלמות אבטחה במבנה בצורה אופטימלית, תוך התחשבות באילוצים שונים כדוגמת: טווח

הראיה, אזורים אסורים לצילום, כניסות ויציאות.

האלגוריתם יכריע את קביעת המיקום האופטימלי של מצלמות כדי למזער כתמים עיוורים ופערים

בכיסוי. תוך שמירה על אילוץ האזורים האסורים לצילום.

אילוצי הכיסוי יהיו כיסוי כל שטח המבנה תוך התחשבות באזורים האסורים לצילום כגון: שירותים,

מקלחות וכדו' , וכן תוך התחשבות בהגדרת שדה הראייה (FOV) של כל מצלמה.

הקלט שיתקבל יהיה תיאור של המבנה שבו ירצו לפרוש מצלמות מעקב. וכן פירוט של אזורים

אסורים בצילום ואזורים בסיכון גבוה.

פתרון אופטימלי יהיה פתרון כזה שבו מספר המצלמות שבשימוש הכולל יהיה מינימלי או שהעלות

הכוללת של מצלמות תהיה מינימלית.

הפתרון המוצע:

הפתרון המוצע נותן מענה לאתגר של כיסוי שטח יעיל באמצעות מצלמות אבטחה. הגישה לבעיה מתבצעת באמצעות אלגוריתם דו-שלבי, שמטרתו לייעל את מיקום המצלמה תוך צמצום המורכבות החישובית.

שלב ראשון - מתמקד במציאת מיקום המצלמה האופטימלי בתוך אזור ברזולוציה נמוכה, המיוצג על ידי מספר קטן של נקודות רשת. האלגוריתם מחשבת את אזורי הכיסוי לכל המצלמות (FOV) ומשתמש בתכנות מספרים בינאריים (BIP), המאפשר קבלת החלטות מדויקת על ידי הקצאת ערכים בינאריים (0 או 1) לציון נוכחות או היעדר מצלמה בכל נקודת רשת. שלב זה נועד לזהות את מיקומי המצלמה הטובים ביותר בשטח קטן יותר.

שלב שני - כולל חידוד מיקום המצלמה בתרחיש ברזולוציה גבוהה, המיוצג על-ידי

מספר גדול יותר של נקודות רשת. נעשה שימוש באלגוריתם קירוב הנקרא טיפוס על גבעות (Climbing-Hill) כדי לשפר באופן איטרטיבי את תצורת המצלמה. החל מהפתרון שהושג בשלב 1 האלגוריתם משנה בהדרגה את מיקומי המצלמה כדי למצוא סידור טוב יותר שממקם את הכיסוי תוך מזעור כתמים עיוורים. טיפוס גבעות משמש לחיפוש יעיל אחר פתרון משופר.
הגישה הדו-שלבית מציעה מספר יתרונות. ראשית, היא מאפשרת שטח כיסוי רחב יותר בשלב הרזולוציה הגבוהה בהתבסס על הפתרון האופטימלי בעולם שהושג בשלב הרזולוציה הנמוכה. זה עוזר להבטיח שמיקום המצלמה יתאים לדרישות הספציפיות של האזור תוך הימנעות ממלכודות אופטימיזציה מקומיות. שנית, הגישה מתייחסת לדאגות של מורכבות חישובית, מה שהופך אותה למתאימה ליצירת מודלים ואופטימיזציה של מיקום המצלמה בתרחישים בעולם האמיתי.

לסיכום, האלגוריתם הדו-שלבי המוצע נותן מענה לאתגר של כיסוי שטח יעיל במצלמות אבטחה. הוא משתמש בטכניקות BIP וטיפוס על גבעות כדי לייעל את מיקום המצלמה
בתרחישים ברזולוציה נמוכה וגבוהה, ומציע איזון בין איכות כיסוי ויעילות חישובית.

רקע תיאורטי בתחום הפרויקט:

בעיית כיסוי שטח יעיל באמצעות מצלמות אבטחה היא נושא שנלמד היטב בתחום המעקב והניטור. אפשרויות פתרון שונות הוצעו ויושמו בתרחישים בעולם האמיתי. להלן כמה גישות ידועות:

האלגוריתם החמדני: גישה אינטואיטיבית לפתרון בעיות . האלגוריתם החמדן הוא אסטרטגיה פתרון בעיות פופולרית במדעי המחשב. הוא מבוסס על גישה פשוטה ואינטואיטיבית: בכל צעד, האלגוריתם בוחר את האפשרות שנראית הטובה ביותר באותו הרגע, בלי לקחת בחשבון את ההשפעה שלה על המשך הפתרון.

דוגמה: נניח שאנחנו רוצים למצוא את המסלול הקצר ביותר בין שתי ערים. האלגוריתם החמדן יבחר בכל צעד את העיר הקרובה ביותר לעיר הנוכחית, עד שיגיע ליעד.
יתרונות:

- פשוט וקל להבנה: ניתן להבין את האלגוריתם החמדן בקלות רבה.
- יעיל: אלגוריתמים חמדנים רבים פועלים במהירות יחסית, מה שהופך אותם לשימושיים בפתרון בעיות גדולות ומורכבות.
- רב-תכליתי: ניתן להשתמש באלגוריתמים חמדנים לפתרון מגוון רחב של בעיות, החל ממציאת מסלולים קצרים ועד לבחירת תיק השקעות אופטימלי.

חסרונות:

- לא תמיד מוצא את הפתרון הטוב ביותר: האלגוריתם החמדן מתמקד בבחירות מקומיות "טובות", בלי להסתכל על התמונה הגדולה. כתוצאה מכך, הוא לא תמיד מוצא את הפתרון האופטימלי הכולל.
- רגיש לשינויים בנתונים: אלגוריתמים חמדנים יכולים להיות רגישים מאוד לשינויים בנתוני הבעיה. שינוי קטן בנתונים יכול להוביל לשינוי משמעותי בפתרון.

דוגמאות לשימוש באלגוריתם חמדן:

- בעיית הסוכן הנוסע: מציאת המסלול הקצר ביותר שעובר דרך כל הערים בגרף.
- בעיית תרמיל הגב: בחירת פריטים בעלי ערך מרבי שניתן להכניס לתרמיל בעל קיבולת מוגבלת.
- אלגוריתם דייקסטרה: מציאת המסלול הקצר ביותר בין צומת נתון לכל שאר הצמתים בגרף מכוון.

לסיכום: האלגוריתם החמדן הוא כלי שימושי וחזק לפתרון בעיות. הוא פשוט להבנה, יעיל ורב-תכליתי. עם זאת, חשוב לזכור שהוא לא תמיד מוצא את הפתרון הטוב ביותר, ויכול להיות רגיש לשינויים בנתונים.

שיטת חיפוש מקומי: שיטת החיפוש המקומי הוא אלגוריתם פופולארי לפתרון בעיות פתרון. הוא מבוסס על הרעיון של חיפוש איטרטיבי אחר פתרון טוב מקומי, תוך שימוש בפונקציית מטרה שמגדירה את איכות הפתרון.

האלגוריתם פועל באופן הבא:

1. אתחול: מתחילים עם פתרון התחלתי אקראי או הֶוּרִיסְטִי (**הֶוּרִיסְטִי-** (Heuristic) הוא כלל אצבע או שיטה פשוטה יחסית לפתרון בעיות מורכבות. הֶוּרִיסְטִיקות אינן מבטיחות למצוא את הפתרון הטוב ביותר, אך הן יכולות לספק פתרון טוב מספיק בזמן סביר).
2. שיפור מקומי: בודקים את כל השכנים של הפתרון הנוכחי (פתרונות אפשריים להגיע באמצעות שינוי בודד) ומחפשים פתרון טוב יותר.
3. חזרה: אם נמצא פתרון טוב יותר, עוברים אליו. אם לא, חוזרים על שלב 2 עד שלא ניתן למצוא שיפור מקומי נוסף.

פונקציית המטרה היא פונקציה שמקבלת פתרון ומחזירה ערך שמייצג את איכות הפתרון. פונקציית המטרה צריכה להיות מוגדרת כך שפתרון טוב יותר יקבל ערך גבוה יותר.

היתרונות של שיטת החיפוש המקומי:

- קלה להבנה וליישום: האלגוריתם קל הבנה וניתן ליישם אותו בצורה פשוטה.
- יעילה: האלגוריתם יכול למצוא פתרונות טובים בזמן סביר.
- גמישה: ניתן להשתמש בשיטת החיפוש המקומי לפתרון מגוון רחב של בעיות פרשות.

החסרונות של שיטת החיפוש המקומי:

- עלולה להיתקע באופטימום מקומי: האלגוריתם להיתקע בפתרון מקומי ללא הפתרון הגלובלי הטוב ביותר.
- היא אינה מובטחת למצוא את הפתרון הגלובלי הטוב ביותר: האלגוריתם אינה מובטחת למצוא את הפתרון הטוב ביותר בכל מקרה.

דוגמה:

אנחנו רוצים למצוא את הדרך הקצרה בין שתי נקודות על המפה. אנו יכולים להשתמש בשיטת החיפוש המקומי כדי לפתור בעיה זו.
הפתרון יכול להתחיל להיות כל דרך אקראית בין שתי הנקודות.
שכנים של פתרון הם כל הדרכים ניתנים אליהן באמצעות שינוי בודד בדרך. לדוגמה, אם אנו משנים את כיוון הנסיעה באחד הקטעים של הדרך, אנו מקבלים שכן חדש.
פונקציית יכולה להיות אורך הדרך.
האלגוריתם ימשיך לחפש שכנים טובים יותר עד שלא ניתן למצוא שיפור מקומי נוסף.
הפתרון הסופי יהיה הדרך הקצרה ביותר בין שתי הנקודות.

לסיכום: שיטת החיפוש המקומי היא כלי רב עוצמה לפתרון בעיות פרשה. היא פשוטה להבנה וליישום, יעילה וגמישה. עם זאת, חשוב לזכור שהיא עלולה להיתקע באופטימום מקומי ולכן הפתרון אינו מובטח שיהיה הפתרון הגלובלי הטוב ביותר.

מיקום מבוסס רשת: היא שיטה לבעיות כיסוי/פרישה ובינהן הצבת מצלמות אבטחה מאזור ספציפי. השיטה היא לחלק את השטח לרשת של תאים ולהציב מצלמות במיקומים אסטרטגיים בתוך כל תא. שיטה זו מבטיחה חלוקה קבועה ואחידה של מצלמות.

יתרונות:

- פשוטה וקלה לישום- הגישה מבוסס הרשת הינה פשוטה להבנה ולישום. חלוקת החלל לרשת הופכת את התכנון והצבת המצלמה אינטואיטיבית.
- כיסוי אחיד- המצלמות מפוזרות באופן שווה על פני הרשת, ומספקות רמת אבטחה בסיסית לכל המרחב המנוטר. דבר המבטיח שאף אזורים מרכזיים לא נותרים לחלוטין ללא כיסוי.

חסרונות:

- מתעלם מחשיבות- האלגוריתם מתיחס לכל האזורים אותו דבר בלי התייחסות לאזורים קריטיים ואסורים לצילום.

לא יעיל במרחבים לא סדירים- בניינים וחדרים הם לעתים רחוקות ריבועים מושלמים. רשת קפדנית עלולה להשאיר פינות מוזרות ללא פיקוח או לבזבז מצלמות במקומות מיותרים.

לסיכום: אלגוריתם המיקום מבוסס הרשת מציע דרך פשוטה ויעילה להפצת מצלמות אבטחה. עם זאת, אלגוריתם המיקום המבוסס על רשת מעניק עדיפות לפשטות על פני אופטימיזציה. למרות שהוא מציע רמת כיסוי בסיסית, הוא עלול להיות לא יעיל ועלול להחמיץ אזורים מכריעים או לבזבז משאבים.

Art gallery problem- הינה בעיה דומה לבעיה המוצעת. בעיית גלרית האומנות או בעית המוזאון הינה בעיית נראות נחקרת היטב בגיאומטיה חישובית. בעית הגלריה לאומנות הינה השאלה על ידי כמה שומרים באותה משמרת תיהיה בגלריה מכוסה על ידי מספר מינימלי של שומרים.

הבעיה ידועה כבעית NP - כלומר אין אלגוריתם יעיל ידוע לפתור אותו עבור כל המקרים.

* למתמטיקאים יש פתרונות מוכחים לצורות מצולעים ספציפיות, כמו משולשים שבהם תמיד מספיקים רק 3 שומרים.

* יש גם גישות היוריסטיות (כמו ניחושים מושכלים) שיכולות למצוא פתרונות טובים לצורות מורכבות, אבל אולי הן לא תמיד יהיו המספר המינימלי המוחלט של שומרים.

דיאגרמות Voronoi: נקראת גם Voronoi tessellation .

גישת הדיאגרמה- מחלקת מרחב לאזורים המבוססים על קבוצת נקודות. לכל נקודה בסט יש תא Voronoi מתאים. בכדי להשתמש בגישה לפרישת מצלמות אבטחה בונים דיאגרמת Voronoi באמצעות מיקומי המצלמה המתוכננים כנקודות. כל תא Voronoi מייצג את האזור המכוסה בעיקר על ידי מצלמה ספציפית.

* תאים קטנים יותר מציינים אזורים מתחת לעין הפקוחה של מצלמה בודדת, בעוד שלתאים גדולים יותר עשוי להיות כיסוי ממספר מצלמות או פוטנציאל נקודות עיוורות.

יתרונות:

- זיהוי פערי כיסוי- הדיאגרמה עוזרת לאתר אזורים שנמצאים בסיכון ללא פיקוח, ומאפשרת מיקום אסטרטגי של מצלמה כדי לחסל שטחים עיוורים.
- אופטימיזציה של הקצאת משאבים- על ידי ניתוח אזורי כיסוי, אפשרי להבטיח שימוש מיטבי במצלמות אבטחה, הימנעות מחפיפה מיותרת ומקסום אבטחה כללית.

חסרונות:

- דיאגרמת Voronoi מניחה שדה ראיה אחיד של המצלמה. במציאות, גורמים כמו זוויות מצלמה וחסימות עשויים לדרוש התייחסות נוספת.
- ניתוח סטטי- התרשים מייצג את מיקום המצלמה המתוכנן. אם מצלמות מוזזות מאוחר יותר, אזורי הכיסוי משתנים, מה שמצריך הערכה מחדש.

לסיכום: באופן כללי, דיאגרמות Voronoi מציעות נקודת התחלה חשובה למיקום אסטרטגי של מצלמה. על ידי ניתוח אזורי הכיסוי, אך יש צורך לייעל את אמצעי האבטחה ולהבטיח שלא יישארו אזורים חיוניים פגיעים.

Triangulation Delaunay : בדומה לדיאגרמות Voronoi, Triangulation Delaunay מחלק את השטח למשולשים על סמך מיקומי המצלמות. כל משולש מייצג אזור מעקב. שיטה זו מציעה מיקום מצלמה אופטימלי יותר, שכן המשולשים בנויים על סמך הקרבה היחסית של מיקומי המצלמה. עם זאת, הפתרון המתקבל ע"י אלגוריתם זה ידרוש מספר גדול יותר של מצלמות.

לאחר חקר ממושך ועמוק שסבב במאמרים מרובים ושיטות פתירה שונות בעלות אותו אופי או פתרון שונה לגמרי.
סיכמתי כי האלגוריתם ה- Two-Phase הינו האלגוריתם המדויק ביותר לבעיה של הפרויקט שלי.
האלגוריתם מחשב תחילה פתרון ברזולוציה נמוכה המשתמש באלגוריתם BIP- Binary Integer Programing ולאחר מכן משתמש באלגוריתם היוריסטי בשיטת החיפוש המקומי על ידי האלגוריתם Hill-Climbing .
עליו פירטתי בחלק של תיאור הבעיה האלגוריתמית.

<u>הליכים עיקריים בפתרון בעיה בטכנולוגיות הנדסה מתקדמות :</u>

בנית שרת:

כתיבת האלגוריתם לשימוש בקלט שהתקבל הכולל את האלגוריתם הדו שלבי:

שלב ראשון-

*כתיבת חישוב הצילום של המצלמות (FOV)

*הגדרת האילוצים לפתרון הבעיה.

*כתיבת ושימוש באלגוריתם BIP לחישוב מיקומי המצלמות ברזולוציה הנמוכה.

שלב שני-

*כתיבת הפונקציה שתשתמש באלגוריתם Hill-climbing

* כתיבת הפונקציה הראשית שתשלח את הקלט המתקבל לאלגוריתם.

בניית צד הלקוח:

כתיבת הפלטפורמה לשימוש באלגוריתם:

*יצירת העיצוב לאפליקציה.

*כתיבת הקוד לריצת האפליקציה ופעולתה השוטפת.

תיאור פרוטוקולי תקשורת:

http

פיתוחים עתידיים:

בניית מצלמות אבטחה שיוכלו להתריע על אדם שמתנהג בצורה חשודה.

המלצות על חנויות להזמנת המצלמות לפי הצורך המסוים למבנה הנוכחי.

תיאור טכנולוגיות הנדסה :

צד שרת: Python – בגלל מורכבות האלגוריתם של הפרויקט שלי החלטתי להשתמש בשפת python הכוללת מבנים המיועדים לאפשר ביטוי של תכניות מורכבות בדרך קצרה וברורה. וכן עקב הצורך בשימוש בספריות בצורה פשוטה וברורה.

חיסרון משמעותי ב Python: Python היא שפת תכנות פרשנית, מה שאומר שהקוד מפורש בזמן ריצה. תהליך הפרשנות גורם להאטה משמעותית בהשוואה לשפות כמו C# שמתקבלות לקוד מכונה בזמן קומפילציה.

לאחר שקלול הנתונים החלטתי כי החיסרון של זמן הריצה אינו משמעותי לפרויקט שלי מכיוון שהאופטימיזציה של האלגוריתם מתמקד בפרישה אופטימלית של מצלמות מבלי להתייחס לזמן הריצה הארוך בשביל לתת את המיקום האופטימלי ביותר למצלות האבטחה.

צד לקוח: React- בחרתי להשתמש ב React בגלל הפשטות וחופש הפעולה שהוא מספק איתו והמבנה הגמיש שלו.

מסד נתונים :

Sql server – לשימוש במסד נתונים לפרויקט שלי חופשתי מסד נתונים טבלאי, נח לשימוש ותומך בשפה העברית.

פרטים פורמליים-

לוחות זמנים :

נובמבר-דצמבר- איסוף נתונים וחקר עצמי על הפרויקט.

ינואר- אפיון ועיצוב המערכת.

פברואר-מרץ- פיתוח אלגוריתם יעיל ככל האפשר לבעיה.

אפריל- הכנת צד לקוח

מאי-בדיקת המערכת והכנת ספר פרויקט.

יוני- הגשת פרויקט סופי.

מנחות פרויקט : המו' רחל טוג.

חתימת הסטודנט :

חתימת רכז המגמה:

2. מבוא

2.1 הרקע לפרויקט

בעולם שמסביבנו תופעת הגניבות, השודים והפריצות קיימות ויתכן אף שבמצב עליה.

רבים מבעלי העסקים ובעלי נכסים מודאגים מהתרחשות בסגנון זה העלולה לאיים על שלום הנפש והרכוש שבבית ובעסק. ולכן מעוניינים בהתקנת מצלמות אבטחה שיהוו מניע מרתיע עבור בעלי כוונות פליליות ביחס לפריצה וכן בשביל להוות אמצעי לבקרה ותיעוד לצורך התייעלות גם לאחר מעשה.

ישנן סיבות רבות נוספות שיכולות לגרום לאנשים להתקין מצלמות אבטחה בביתם או במשרד שלהם, כגון:

ניטור מרחוק – בעזרת שימוש במכשירים ניידים בעלי הבית יכולים לפקוח עין על רכושם גם כשהם לא נמצאים בבית ובעצם לשמור ולהשגיח על הבית מכל מקום ובכל עת.

ניטור קשישים ילדים והגנה על חיות מחמד - מצלמות אבטחה יכולות להיות שימושיות לניטור ילדים ובני משפחה קשישים. כדי להבטיח את שלומם ורווחתם. וכן ניתן לפקוח עין על חיות המחמד שנשארות בבית בזמן שאתם רחוקים מהם ולוודא כי הם בטוחים ומאובטחים.

ניטור עובדי שירות - ניתן להשתמש במצלמות האבטחה על מנת לפקוח עין על נותני שירות כגון עוזרות בית, בייביסיטר או גנן, על מנת להימנע מאי נעימות בעתיד.

בקיצור התקנת מצלמות אבטחה נותנת תחושת שלווה ורוגע מה שגורם שצריכת מצלמות אבטחה הולכת וגוברת, ובכדי למקסם את השימוש במצלמות אבטחה רצוי ומומלץ להתקין אותן מיקומים הטובים ביותר בכדי לחסוך בהוצאות בד בבד עם הגנה מספקת.

הפרויקט שלי נקרא: field of view , זהו פרויקט שכשמו כן הוא – מחשב שדה ראיה.

הפרויקט עוסק בתחום כיסוי מבנים על ידי מצלמות אבטחה בצורה אופטימלית- יעילה ביותר כך שהמבנה יכוסה בצורה מירבית עם כמה שפחות מצלמות אבטחה.

2.2 תהליך המחקר

בתחום התקנת מצלמות האבטחה, ניתן למצוא המלצות כלליות רבות היכן להתקין את מצלמות האבטחה ואלו מצלמות לקנות - שמוגשות מטעם מוכרים ויבואנים רבים.

וכן ישנם אתרים המדמים מצלמות אבטחה מכמה סוגים ואת שטח הכיסוי שלהם על פי גובה ההתקנה של המצלמה.

כל אלו הן תוצאות נחמדות בשביל מתקין מצלמות אבטחה או בעל נכס כלשהו המעוניין להתקין צלמות אבטחה בעצמו, והן יכולות להועיל במעט.

אך המטרה שלי בפרויקט שאני יוצרת ומפתחת היא לקחת את ההמלצות שלב אחד אחר-כך ולייעץ למתייעץ עצה כזו שתתאים לו אישית למבנה שלו ותאמר לו בפירוש לכמה מצלמות אבטחה הוא יזדקק והיכן והציב אותן.

2.3 סקירת ספרות

וויקיפדיה -בכדי להבין את בעיות כיסוי ובעיות NP

https://he.wikipedia.org/wiki/%D7%91%D7%A2%D7%99%D7%99%D7%AA_%D7%9B%D7%99%D7%A1%D7%95%D7%99_%D7%A7%D7%95%D7%93%D7%A7%D7%95%D7%93%D7%99%D7%9D

https://he.wikipedia.org/wiki/NP_(%D7%9E%D7%97%D7%9C%D7%A7%D7%AA_%D7%A1%D7%99%D7%91%D7%95%D7%9B%D7%99%D7%95%D7%AA)

האלגוריתם עליו חשבתי להתבסס אך לבסוף פסלתי עקב חוסר התאמה-

https://www.sciencedirect.com/science/article/pii/S1570866712000196

Two-Phase- לאחר חקר מרובה הגעתי למאמר הזה ועליו התבססתי בכדי ליישם את האלגוריתם.

https://www.hindawi.com/journals/sp/2016/4801784/#model-and-solution

חיפושים רבים בגוגל

https://www.google.com/search?q=Binary+nteger+Programmng&safe=active&sca_esv=ae2d012acb0463ba&rlz=1C1GCEU_iwIL979IL979&sxsrf=ACQVn08w1O9D3Czh8csZK-D-Pk4Q1uDvLQ%3A1706800342802&ei=1rS7ZYvaMNWnptQPucW7gAk&udm=&ved=0ahUKEwiLnYrhtoqEAxXVk4kEHbniDpAQ4dUDCBA&uact=5&oq=Binary+nteger+Programmng&gs_lp=Egxnd3Mtd2l6LXNlcnAiGEJpbmFyeSB

udGVnZXIgUHJvZ3JhbW1uZ0jK0QFQAFj8zgFwAHgAkAEAmAEAoAEAqgEAuAEDyAEA-AEBqAIA4gMEGAAgQQ&sclient=gws-wiz-serp

שיחות רבות עם chat GPT

https://chat.openai.com/share/eddeb09e-a718-4ea9-9118-3cd34060fee2

2.4 אתגרים מרכזיים

2.4.1 הבעיה איתה התמודדתי

בפרויקט זה הוצבו לפני מספר אתגרים:

- חיפוש אחר אלגוריתם תואם בעיה- בעיית פרישת מצלמות אבטחה בתוך מבנה הינה בעיית כיסוי מסוג NP -בעיות שקשה לאמוד את הסיבוכיות החישובית האבסולוטית שלהן. ולכן משתמשים האלגוריתמי קירוב לשם פתרון הבעיות. מה שגרם לי לבעיה בחיפוש פתרון אופטימלי.
- יישום האלגוריתם הנבחר- לאחר מציאת האלגוריתם ובחירתו כאלגוריתם המתאים ביותר לפתרון הבעיה. עמד פני אתגר גדול מאד והוא לקחת את האלגוריתם וליישם אותו בשפת תכנות שיוכל לרוץ לתת תוצאה מתאימה.
- מציאת נתונים ליצירת מסד נתונים – אתגר גדול נוסף היה למצוא מצלמות בעלות הפרמטרים הדרושים בכדי להשתמש בהן ליצירת מסד הנתונים שבו ישתמש האלגוריתם.

2.4.2 הסיבות לבחירת הנושא

לפני מספר שנים כשלמדתי בבית הספר היסודי באחת השנים התקינו מצלמות אבטחה בבית הספר, ומאז בכל מקום שנתקלתי במצלמות אבטחה נהניתי לבדוק איך הן מותקנות ואילו שטחים הן אינן מכסות ולכן כאשר חיפשתי פרויקט שמכיל אלגוריתם מייד חשבתי על הרעיון הזה.

בנוסף לכך כשחיפשתי רעיון שעליו אני אבנה את הפרויקט גמר שלי, רציתי להשתפשף בהבנת אלגוריתמים והתעסקות איתם מה שגרם לי למצוא את הבעיה הנוכחית של כיסוי מבנה על ידי מצלמות אבטחה בצורה אופטימלית העוסקת בבעיות כיסוי.

2.4.3 הצורך עליו עונה הפרויקט

הפרויקט מסייע לאנשים המעוניינים להתקין מצלמות אבטחה לבדם ולחסוך עלויות או שמעוניינים לקבל שירות התקנה בלי יעוץ ולבדוק בעצמם את כמות מצלמות האבטחה הנדרשות לביתם או למשרדם ואת מיקומם.

2.4.4 דרכי פתרון שנבדקו

* אלגוריתם חמדני- בתחילה בניתי אלגוריתם חמדני שמוצא את הפינות של החדר, מציב שם מצלמה, מכסה את השטח המגולה לעיני המצלמות וכך פועל כל עוד השטח אינו מכוסה כולו על ידי מצלמות אבטחה. אך למעשה גיליתי כי אלגוריתם זה אינו תואם את הנתונים מכיוון שאת האלגוריתם הזה בניתי על סמך חיפוש ראשוני על טווח המעלות שקיימות למצלמות אבטחה והתוצאה הייתה שטווח המעלות של מצלמות האבטחה הוא בין 30 מעלות ל-80 מעלות מה שכמובן אינו נכון מכיוון שישנן בעלות טווח ראיה גדול הרבה יותר (160, 180, 270 מעלות ואף יותר), לכן שבתי לחפש אחר אלגוריתם מתאים.

* במהלך החיפושים פגשתי פרויקט ב GitHub שעסק בכיסוי חדר על ידי מצלמות אבטחה, אך הוא השתמש באלגוריתמי בינה מלאכותית ואני שרציתי להתעסק עם אלגוריתמים קיוויתי למצוא אלגוריתם מתאים יותר.

* לאחר חיפושים נוספים מצאתי מאמר שמתעסק בכיסוי שטח על ידי חיישנים, האלגוריתם המוצע חיפש נתיב כיסוי מ s ל- t עם מכשולים בדרך באמצעות cw-voronoi. אך גם את המאמר הזה עזבתי לנפשו מכיוון שהאלגוריתם שהוצע על ידו יועד למציאת זן קרנפים שכמעט ונכחדו מין העולם ולכן מטרתו היתה לכסות את שטח המחייה שלהם על ידי מצלמות אך לא בצורה מרושתת שמתאימה לשמש כמצלמות אבטחה, וכן התייחסות האלגוריתם היתה בעיקר לתוואי השטח כגון הרים וגבעות מה שאינו תואם שטח של מבנה. ועל כן המשכתי בחיפוש

3. יעדים ומטרות

מטרות-

- רכישת ידע ומיומנות בשפת python .
- רכישת ידע והכרה נרחבת בשפת react.
- השתפשפות בהבנת אלגוריתמים גם מורכבים.
- יכולת לבנות פרויקט שלם כולל כל השלבים משלב התכנון עד לשלב הגמירה.
- האתר יעוצב בצורה ידידותית ונוחה כך שתמיד תרצו להשתמש בו שוב ושוב.
- יצירת ממשק נח וברור תוך שמירה על עיצוב אחיד של מסכי האתר.

יעדים-

- לדעת להתמודד עם בעיות באופן עצמאי.
- תכנון המערכת תוך שימת דגש על כתיבה נכונה, מאורגנת ומקצועית של הקוד.
- חקירת פרמטרים להתקנת מצלמות אבטחה והשפעתם על שטח הראיה של המצלמה.
- שימוש בפרמטרים אלו למציאת כמות המצלמות הנצרכת ומיקומן.

4. אתגרים

בתחילה חשבתי שהתקנת מצלמות אבטחה הינה פשוטה אך ככל שהעמקתי בעניין של התקנת מצלמות אבטחה בצורה יעילה גיליתי יותר ויותר פרמטרים במצלמה, בהתקנתה ובכל דבר מסביב המשפיעים על הראיה ועל התוצאות מה שגורם לכל העניין של התקנתה המצלמות והמחשבה כיצד להתקין אותן להיות דבר מורכב לכשעצמו.

בנוסף לכך מציאת אלגוריתם שיתאים לצרכים של הפרויקט שלי או חשיבה על אלגוריתם לבד הייתה מורכבת למדי מכיוון שאין פרויקטים מוקדמים שפרשו מצלמות אבטחה במבנה בצורה אופטימלית.

וכן המאמרים שנגעו בעניין הזה היו מאמרים אקדמיים בשפת האנגלית מה שגרם לכך ששלב המחקר של הפרויקט נמשך זמן רב מה שהשאיר לשלב הביצוע המעשי זמן קצוב.

בנוסף לכך המרת האלגוריתם לשפה תכנותית היווה אתגר ממשי.

וכן בתחילה השתמשתי בספרית Scipy בכדי למצוא פתרון אופטימלי של בעיית האופטימיזציה שיצרתי. ורק כשהרצתי אותה לאחר כתיבת כל הקוד כולו התברר שהספרייה אינה תומכת בצורה ישירה ומספקת בבעיות אופטימיזציה בינארית מה שגרם לחוסר פתרון. והכריח אותי להסב את הקוד כולו כך שיתאים לשימוש ספרייה אחרת שתומכת בצורה ברורה בפתרון בעיות אופטימיזציה בינארית.

5. מדדי הצלחה

במידה והאלגוריתם ייתן לי תוצאה של מספר מצלמות ומיקום הגיוני למבנה פשוט, אני אחשיב את זה כהצלחה.

6. רקע תאורטי

בעיות כיסוי הן קבוצה רחבה של בעיות אופטימיזציה במדעי המחשב ,בהן המטרה היא למצוא קבוצה מינימלית של "כיסויים" שתכסה קבוצה נתונה של "איברים."

בעיות כיסוי רבות הן **בעיות-NP שלמות** ,כלומר קשה מאוד למצוא להן פתרון אופטימלי בזמן סביר. כתוצאה מכך, מפתחים אלגוריתמים מקורבים שמטרתם למצוא פתרון "טוב מספיק" בזמן סביר.

ישנם סוגים רבים של בעיות כיסוי, ישנם כמה סוגים עיקריים מוכרים לבעיות כיסוי:

- בעיית כיסוי קבוצות :בבעיה זו ,ניתנת קבוצה של קבוצות משנה ,והמטרה היא למצוא קבוצה מינימלית של קבוצות משנה אלה שתכסה את כל האלמנטים בקבוצה המקורית.
- בעיית כיסוי קודקודים בגרף :בבעיה זו ,נתון גרף ,והמטרה היא למצוא קבוצה מינימלית של קודקודים שתכסה את כל הקשתות בגרף.
- בעיית כיסוי מתקנים :בבעיה זו ,ניתנת קבוצה של מתקנים ,והמטרה היא למצוא קבוצה מינימלית של מתקנים שתכסה את כל האזורים שצריכים להיות מכוסים.

בעיית כיסוי שטח יעיל באמצעות מצלמות אבטחה היא נושא שנלמד היטב בתחום המעקב והניטור. אפשרויות פתרון שונות הוצעו ויושמו בתרחישים בעולם האמיתי.

7. מצב קיים

- האלגוריתם החמדני.
- שיטת חיפוש מקומי.
- מיקום מבוסס רשת.
- Art gallery problem.
- דיאגרמות Voronoi.
- Triangulation Delaunay .

8. ניתוח חלופות מערכתי

האלגוריתם החמדני: גישה אינטואיטיבית לפתרון בעיות . האלגוריתם החמדן הוא אסטרטגיה פתרון בעיות פופולרית במדעי המחשב. הוא מבוסס על גישה פשוטה ואינטואיטיבית: בכל צעד, האלגוריתם בוחר את האפשרות שנראית הטובה ביותר באותו הרגע, בלי לקחת בחשבון את ההשפעה שלה על המשך הפתרון.

דוגמה: נניח שאנחנו רוצים למצוא את המסלול הקצר ביותר בין שתי ערים. האלגוריתם החמדן יבחר בכל צעד את העיר הקרובה ביותר לעיר הנוכחית, עד שיגיע ליעד.
יתרונות:

- פשוט וקל להבנה: ניתן להבין את האלגוריתם החמדן בקלות רבה.
- יעיל: אלגוריתמים חמדנים רבים פועלים במהירות יחסית, מה שהופך אותם לשימושיים בפתרון בעיות גדולות ומורכבות.
- רב-תכליתי: ניתן להשתמש באלגוריתמים חמדנים לפתרון מגוון רחב של בעיות, החל ממציאת מסלולים קצרים ועד לבחירת תיק השקעות אופטימלי.

חסרונות:

- לא תמיד מוצא את הפתרון הטוב ביותר: האלגוריתם החמדן מתמקד בבחירות מקומיות "טובות", בלי להסתכל על התמונה הגדולה. כתוצאה מכך, הוא לא תמיד מוצא את הפתרון האופטימלי הכולל.
- רגיש לשינויים בנתונים: אלגוריתמים חמדנים יכולים להיות רגישים מאוד לשינויים בנתוני הבעיה. שינוי קטן בנתונים יכול להוביל לשינוי משמעותי בפתרון.

דוגמאות לשימוש באלגוריתם חמדן:

- בעיית הסוכן הנוסע: מציאת המסלול הקצר ביותר שעובר דרך כל הערים בגרף.
- בעיית תרמיל הגב: בחירת פריטים בעלי ערך מרבי שניתן להכניס לתרמיל בעל קיבולת מוגבלת.
- אלגוריתם דייקסטרה: מציאת המסלול הקצר ביותר בין צומת נתון לכל שאר הצמתים בגרף מכוון.

לסיכום: האלגוריתם החמדן הוא כלי שימושי וחזק לפתרון בעיות. הוא פשוט להבנה, יעיל ורב-תכליתי. עם זאת, חשוב לזכור שהוא לא תמיד מוצא את הפתרון הטוב

ביותר, ויכול להיות רגיש לשינויים בנתונים.

שיטת חיפוש מקומי: שיטת החיפוש המקומי הוא אלגוריתם פופולארי לפתרון בעיות פתרון. הוא מבוסס על הרעיון של חיפוש איטרטיבי אחר פתרון טוב מקומי, תוך שימוש בפונקציית מטרה שמגדירה את איכות הפתרון.

האלגוריתם פועל באופן הבא:

- אתחול: מתחילים עם פתרון התחלתי אקראי או הֶוּרִיסְטִי (**הֶוּרִיסְטִי-** (Heuristic) הוא כלל אצבע או שיטה פשוטה יחסית לפתרון בעיות מורכבות. הֶוּרִיסְטִיקות אינן מבטיחות למצוא את הפתרון הטוב ביותר, אך הן יכולות לספק פתרון טוב מספיק בזמן סביר).
- שיפור מקומי: בודקים את כל השכנים של הפתרון הנוכחי (פתרונות אפשריים להגיע באמצעות שינוי בודד) ומחפשים פתרון טוב יותר.
- חזרה: אם נמצא פתרון טוב יותר, עוברים אליו. אם לא, חוזרים על שלב 2 עד שלא ניתן למצוא שיפור מקומי נוסף.

פונקציית המטרה היא פונקציה שמקבלת פתרון ומחזירה ערך שמייצג את איכות הפתרון. פונקציית המטרה צריכה להיות מוגדרת כך שפתרון טוב יותר יקבל ערך גבוה יותר.

היתרונות של שיטת החיפוש המקומי:

- קלה להבנה וליישום: האלגוריתם קל הבנה וניתן ליישם אותו בצורה פשוטה.
- יעילה: האלגוריתם יכול למצוא פתרונות טובים בזמן סביר.
- גמישה: ניתן להשתמש בשיטת החיפוש המקומי לפתרון מגוון רחב של בעיות פרשות.

החסרונות של שיטת החיפוש המקומי:

- עלולה להיתקע באופטימום מקומי: האלגוריתם להיתקע בפתרון מקומי ללא הפתרון הגלובלי הטוב ביותר.
- היא אינה מובטחת למצוא את הפתרון הגלובלי הטוב ביותר: האלגוריתם אינה מובטחת למצוא את הפתרון הטוב ביותר בכל מקרה.

דוגמה:

אנחנו רוצים למצוא את הדרך הקצרה בין שתי נקודות על המפה. אנו יכולים להשתמש בשיטת החיפוש המקומי כדי לפתור בעיה זו.
הפתרון יכול להתחיל להיות כל דרך אקראית בין שתי הנקודות.

שכנים של פתרון הם כל הדרכים ניתנים אליהן באמצעות שינוי בודד בדרך. לדוגמה, אם אנו משנים את כיוון הנסיעה באחד הקטעים של הדרך, אנו מקבלים שכן חדש. פונקציית יכולה להיות אורך הדרך.
האלגוריתם ימשיך לחפש שכנים טובים יותר עד שלא ניתן למצוא שיפור מקומי נוסף. הפתרון הסופי יהיה הדרך הקצרה ביותר בין שתי הנקודות.

לסיכום: שיטת החיפוש המקומי היא כלי רב עוצמה לפתרון בעיות פרשה. היא פשוטה להבנה וליישום, יעילה וגמישה. עם זאת, חשוב לזכור שהיא עלולה להיתקע באופטימום מקומי ולכן הפתרון אינו מובטח שיהיה הפתרון הגלובלי הטוב ביותר.

מיקום מבוסס רשת: היא שיטה לבעיות כיסוי/פרישה ובינהן הצבת מצלמות אבטחה מאזור ספציפי. השיטה היא לחלק את השטח לרשת של תאים ולהציב מצלמות במיקומים אסטרטגיים בתוך כל תא. שיטה זו מבטיחה חלוקה קבועה ואחידה של מצלמות.

יתרונות:

- פשוטה וקלה לישום- הגישה מבוסס הרשת הינה פשוטה להבנה ולישום. חלוקת החלל לרשת הופכת את התכנון והצבת המצלמה אינטואיטיבית.
- כיסוי אחיד- המצלמות מפוזרות באופן שווה על פני הרשת, ומספקות רמת אבטחה בסיסית לכל המרחב המנוטר. דבר המבטיח שאף אזורים מרכזיים לא נותרים לחלוטין ללא כיסוי.

חסרונות:

- מתעלם מחשיבות- האלגוריתם מתיחס לכל האזורים אותו דבר בלי התייחסות לאזורים קריטיים ואסורים לצילום.
  לא יעיל במרחבים לא סדירים- בניינים וחדרים הם לעתים רחוקות ריבועים מושלמים. רשת קפדנית עלולה להשאיר פינות מוזרות ללא פיקוח או לבזבז מצלמות במקומות מיותרים.

לסיכום: אלגוריתם המיקום מבוסס הרשת מציע דרך פשוטה ויעילה להפצת מצלמות אבטחה. עם זאת, אלגוריתם המיקום המבוסס על רשת מעניק עדיפות לפשטות על פני אופטימיזציה. למרות שהוא מציע רמת כיסוי בסיסית, הוא עלול להיות לא יעיל ועלול להחמיץ אזורים מכריעים או לבזבז משאבים.

Art gallery problem- הינה בעיה דומה לבעיה המוצעת. בעיית גלרית האומנות או בעית המוזאון הינה בעיית נראות נחקרת היטב בגיאומטיה חישובית. בעית הגלריה לאומנות הינה השאלה על ידי כמה שומרים באותה משמרת תיהיה בגלריה מכוסה על ידי מספר מינימלי של שומרים.
הבעיה ידועה כבעית NP - כלומר אין אלגוריתם יעיל ידוע לפתור אותו עבור כל המקרים.
* למתמטיקאים יש פתרונות מוכחים לצורות מצולעים ספציפיות, כמו משולשים שבהם

תמיד מספיקים רק 3 שומרים.
* יש גם גישות היוריסטיות (כמו ניחושים מושכלים) שיכולות למצוא פתרונות טובים לצורות מורכבות, אבל אולי הן לא תמיד יהיו המספר המינימלי המוחלט של שומרים.

דיאגרמות Voronoi: נקראת גם Voronoi tessellation .
גישת הדיאגרמה- מחלקת מרחב לאזורים המבוססים על קבוצת נקודות. לכל נקודה בסט יש תא Voronoi מתאים. בכדי להשתמש בגישה לפרישת מצלמות אבטחה בונים דיאגרמת Voronoi באמצעות מיקומי המצלמה המתוכננים כנקודות. כל תא Voronoi מייצג את האזור המכוסה בעיקר על ידי מצלמה ספציפית.
* תאים קטנים יותר מציינים אזורים מתחת לעין הפקוחה של מצלמה בודדת, בעוד שלתאים גדולים יותר עשוי להיות כיסוי ממספר מצלמות או פוטנציאל נקודות עיוורות.

יתרונות:
- זיהוי פערי כיסוי- הדיאגרמה עוזרת לאתר אזורים שנמצאים בסיכון ללא פיקוח, ומאפשרת מיקום אסטרטגי של מצלמה כדי לחסל שטחים עיוורים.
- אופטימיזציה של הקצאת משאבים- על ידי ניתוח אזורי כיסוי, אפשרי להבטיח שימוש מיטבי במצלמות אבטחה, הימנעות מחפיפה מיותרת ומקסום אבטחה כללית.

חסרונות:
- דיאגרמת Voronoi מניחה שדה ראיה אחיד של המצלמה. במציאות, גורמים כמו זוויות מצלמה וחסימות עשויים לדרוש התייחסות נוספת.
- ניתוח סטטי- התרשים מייצג את מיקום המצלמה המתוכנן. אם מצלמות מוזזות מאוחר יותר, אזורי הכיסוי משתנים, מה שמצריך הערכה מחדש.

לסיכום: באופן כללי, דיאגרמות Voronoi מציעות נקודת התחלה חשובה למיקום אסטרטגי של מצלמה. על ידי ניתוח אזורי הכיסוי, אך יש צורך לייעל את אמצעי האבטחה ולהבטיח שלא יישארו אזורים חיוניים פגיעים.

Triangulation Delaunay : בדומה לדיאגרמות Voronoi, Triangulation Delaunay מחלק את השטח למשולשים על סמך מיקומי המצלמות. כל משולש מייצג אזור מעקב. שיטה זו מציעה מיקום מצלמה אופטימלי יותר, שכן המשולשים בנויים על סמך הקרבה היחסית של מיקומי המצלמה. עם זאת, הפתרון המתקבל ע"י אלגוריתם זה ידרוש מספר גדול יותר של מצלמות.

9. תיאור החלופה הנבחרת והנימוקים לבחירתה

לאחר חקר ממושך ועמוק שסבב במאמרים מרובים ושיטות פתירה שונות בעלות אותו אופי או פתרון שונה לגמרי.
סיכמתי כי האלגוריתם ה- Two-Phase הינו האלגוריתם המדויק ביותר לבעיה של הפרויקט שלי.
האלגוריתם Two-Phase פועל בצורה מונחית חדרים ומורכב משני שלבים:
בשלב הראשון האלגוריתם לוקח את הקורדינאטות של החדרים ואת נתוני המצלמות לפיהן הוא מחשבת את שדה הראיה של כל מצלמה. האלגוריתם בונה בעיית אופטימיזציה בינארית (Binary Integer Problem), ופותר אותה על ידי ספריה קיימת לבעיות אופטימיזציה עם משתנים בינאריים.
בשלב השני האלגוריתם לוקח את התוצאה שהתקבלה בשלב ראשון ומאפטם אותה על ידי אלגוריתם קירוב- Hill Climbing .
יתרונות:

- האלגוריתם מדויק ונותן תוצאה הקרובה ביותר למצב האופטימלי.
- האלגוריתם מתחשב בשטח הכיסוי המדויק של המצלמה.

החסרונות:

- האלגוריתם אינו מתחשב במכשולים.
- האלגוריתם ארוך ומלא איטרציות ולכן לוקח זמן רב עד לקבלת הפתרון.

10. אפיון המערכת

10.1 ניתוח דרישות המערכת

סביבת פיתוח:

חומרה: מעבד i5 RAM 8GB

עמדת פיתוח: מחשב Intel

מערכת ההפעלה: Windows 10

שפות תוכנה: צד שרת: Python | צד לקוח: react, html, CSS

כלי תוכנה לפיתוח המערכת: צד שרת: PyCharm | צד לקוח: VScode

מסד נתונים: SQL Server

חיבור לרשת: אין צורך

תוספים טכנולוגים: אין

10.2 מודול המערכת

נושאים שבאחריות המערכת-

- האפליקציה מאפשרת הרשמה והתחברות למערכת
- קליטת נתונים- המערכת מאפשרת למשתמש להעלות תמונה ולבחור את קודקודי החדרים שברצונו לכסות במצלמות (בסדר עוקב) וכן המשתמש יבחר את גובה החדר שלו. לאחר מכן המשתמש ילחץ על כפתור לשליחת הנתונים לאלגוריתם הראשי שיבנה מהנתונים בעיית אופטימיזציה יפתור אותה ואת התוצאה ישפר עם אלגוריתם קירוב.
- צפייה בהיסטורית הבקשות שלו.

נושאים שאינם באחריות המערכת-

- המערכת עוסקת בפריסת מצלמות אבטחה בחדרים בלי מכשולים.
- האפליקציה מקבלת קורדינאטות של חדרים בצורה רודפת, כלומר פינה רודפת פינה בסדר עוקב לפי כיוון השעון או נגד כיוון השעון.

10.3 אפיון פונקציונלי

* המערכת מציבה מצלמות אבטחה במיקומים המתאימים לפי קודקודי המבנה שמתקבלים כקלט.

* המערכת מציגה עבודות קודמות.

10.4 ביצועים עיקריים

* המערכת קולטת מהמשתמש רשימה של חדרים כשלכל חדר ישנם המיקומי הX וה-Y של קודקודי החדר.

* האלגוריתם מקבל את הקלט מהמשתמש ומחזיר את כמות המצלמות הנדרשות ו את המיקום שלהן.

10.5 אילוצים

המערכת צריכה להתמודד עם חדרים עם קירות אלכסוניים או יותר מארבע פינות.

11. תיאור הארכיטקטורה

11.1 הארכיטקטורה של הפתרון המוצע בפורמט של Top-Down level design

מודל 3 השכבות:

הפתרון מבוסס על מודל 3 שכבות קלאסי, המפריד בין 3 רכיבים עיקריים:

שכבת הצגה (Presentation Layer):

אחראית על אינטראקציה עם המשתמש.

ממומשת באמצעות ממשק משתמש גרפי (GUI) מבוסס דפדפן (Web Application).

יצרתי אתר אינטרנט המפותח ב-React .

שכבת הלוגיקה העסקית (Business Logic Layer):

מכילה את פונקציונליות הליבה של המערכת.

אחראית על עיבוד נתונים, ביצוע חישובים של המערכת.

יצרתי שרת API המפותח ב-Python.

שכבת נתונים (Data Access Layer):

אחראית על גישה לאחסון הנתונים.

מתקשרת עם מסד נתונים ומנהלת את קריאת, כתיבה ועדכון נתונים.

השתמשתי במסד הנתונים SqlServer המאחסן את כל נתוני המערכת.

.

ישנם יתרונות לחלוקת המערכת לפי מודל שלושת השכבות:

מודולריות: מאפשר פיתוח ותחזוקה קלים יותר של המערכת.

גמישות: ניתן לשנות או להחליף רכיבים בודדים מבלי להשפיע על שאר המערכת.

תחזוקה: מקל על איתור שגיאות ותיקונים.

אבטחה: שיפור אבטחת המערכת על ידי הפרדת רכיבים רגישים.

11.2 תיאור הרכיבים בפתרון

הארכיטקטורה של הפתרון המוצע:

פירוט על כל אחד מהרכיבים

פונקציות שרת - צד שרת Python:

השרת מכיל את האלגוריתם הראשי של הפרויקט. והוא נכתב באמצעות שפת תכנות Python, האלגוריתם מומש ומבוצע על ידי השרת. Python היא שפת תכנות פופולרית המאפשרת כתיבת קוד בצורה קריאה ומובנת, ומצוינת לפיתוח אלגוריתמים.

שרת אינטרנט Web API- HTTP:

Web API (Application Programming Interface) הוא פרוטוקול התקשורת בין צד השרת לצד הלקוח שבודק, מאפשר ומספק גישה לשירותים ולמידע מהשרת. פרוטוקול זה מגדיר את הפורמט שבו נשלחות הבקשות והתגובות בין הלקוח והשרת.

Web API מספק את היכולת לשתף נתונים ותפקידים שונים באמצעות האינטרנט, כגון קבלת ושליחת מידע, ביצוע פעולות, עדכון מצבים ועוד. פופולריותו של Web API גדלה מאוד בשנים האחרונות בזכות הפופולריות של אפליקציות התומכות בעיקר בגישה למידע ושירותים דרך האינטרנט.

שפת התקשורת הנפוצה ביותר ב Web API היא HTTP (Hypertext Transfer Protocol). HTTP מגדיר כיצד מתנהלת התקשורת בין הלקוח והשרת. בשימוש עם HTTP, הלקוחות שולחים בקשות לשרת ומקבלים תגובות מצד השרת בפורמט תקני כמו JSON או XML

ממשק לקוח - צד לקוח React:

צד הלקוח של הפרויקט מכיל את ממשק המשתמש שבו המשתמשים מתקשרים עם האפליקציה. על מנת לקשר את הממשק לשרת ולאפשר תקשורת חוזרת, את צד הלקוח כתבתי ב- React .

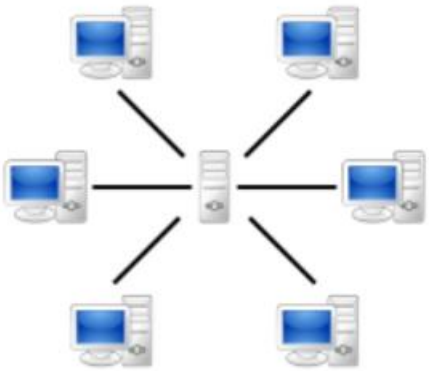
React היא ספריית פיתוח פופולרית לצד הלקוח, המשמשת לבניית ממשק משתמש (UI) מודרני ואינטראקטיבי באפליקציות אינטרנט. היא פותחה על ידי חברת Facebook והיא תואמת באופן מושלם לפרדיגמת תצוגה ב--MVC (Model-View-Controller).

11.3 ארכיטקטורת רשת

המודל מחלק את המשימות או העומס העבודה בין השרת והלקוח.
השרת הוא תוכנה פסיבית שמאזינה לרשת ומחכה לקבל בקשות מהלקוח.
הלקוח, שהוא ממשק המשתמש, מופעל על ידי המשתמש ופונה לשרת כאשר הוא זקוק למידע או שירותים מהשרת.

מודל שרת-לקוח הוא אחת מתצורות התקשורת הנפוצות ביותר ברשתות מחשבים, והוא מאפשר יעילות והבטחה בתקשורת בין המערכות השונות. במודל זה, השרת מספק שירותים ומשאבים, והלקוחות מבצעים בקשות ופניות אל השרת לקבלת המידע או השירותים שהם זקוקים להם.

להלן תיאור לצורת מודל שרת לקוח:



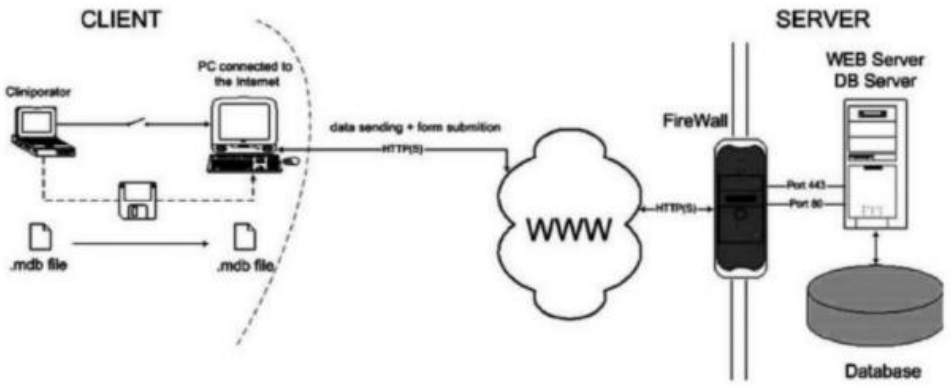
11.4 תיאור פרוטוקולי תקשורת

המערכת מורכבת משרת IIS המריץ את האתר בסביבת ה- Server,

מסד נתונים - DataBase של Server SQL.

ממשק משתמש בצד הלקוח: דפדפן אינטרנט כלשהו:

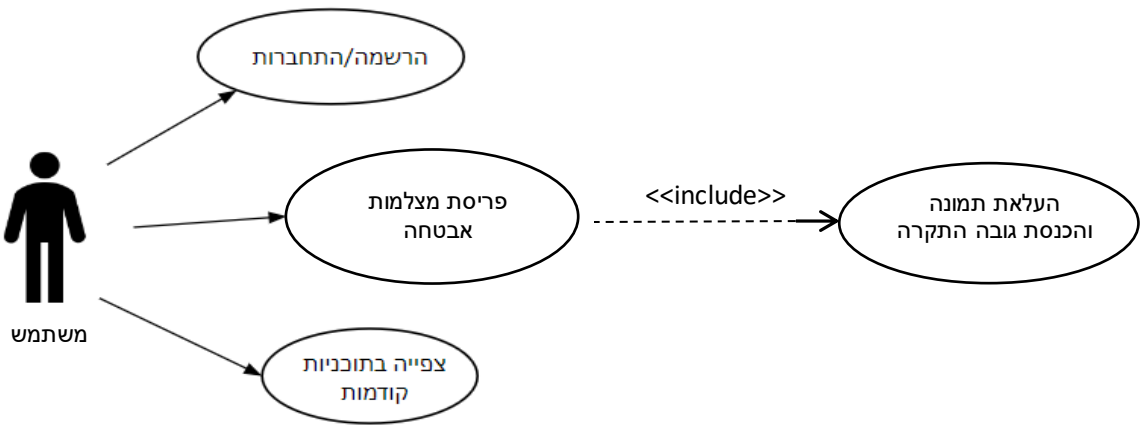
internet explorer, chrome, Firefox



11.5 שרת-לקוח

כפי שנכתב לעיל טכנולוגיית התקשורת בין השרת ללקוח היא ע"י – WebApi HTTPS.

12. ניתוח ותרשים UML / Use cases של המערכת המוצעת



12.1 תיאור ה-UC העיקריים של המערכת

המשתמש

- הרשמה
- כניסה
- פרישת מצלמות אבטחה
- צפייה בעבודות קודמות

מערכת

- רישום משתמש
- אימות המשתמש
- פרישת מצלמות האבטחה
- שליחת פרטי העבודות הקודמות

12.2 הצגת use case עבור כל הפונקציות העיקריות בפרויקט.

שם מקרה השימוש-	רישום
תיאור:	רישום למערכת
שחקנים:	משתמש חדש
תנאי מקדים:	מייל תקין
הזנק:	מייל או סיסמה אינם תקינים
מסלול עיקרי:	a. השתמש מכניס מייל וסיסמא. b. המערכת בודקת בבסיס נתונים האם המשתמש כבר קיים במערכת ובמידה והוא לא קיים- המערכת תכניס אותו לבסיס הנתונים.
תנאי סיום:	רישום עבר בהצלחה

תדירות:	פעם אחת

שם מקרה השימוש-	**פרישת מצלמות אבטחה**
תיאור:	קביעת מיקומי מצלמות האבטחה במבנה
תנאי מקדים:	משתמש לחץ על הכפתור לפרישת המצלמות
שחקנים:	משתמש קיים
הזנק:	משתמש שלח רשימת חדרים
מסלול עיקרי:	1. המשתמש מעלה תמונה או שרטוט של המבנה בו הוא רוצה להציב מצלמות.

	2. המשתמש לוחץ על קורדינאטות של החדר ולאחר מכן על כפתור הוספת חדר וכך הוא בוחר חדר לפרישת מצלמות אבטחה בו במידה והוא רוצה להוסיף עוד חדר הוא יחזור על הפעולה 2. 3. המשתמש לוחץ על כפתור התחלת ביצוע האלגוריתם. 4. המערכת מפעילה את האלגוריתם עד לקבלת תוצאה. 5. המערכת מציגה את התוצאה למשתמש.
תנאי סיום:	האלגוריתם גמר את החישוב
תדירות:	פעם במספר שנים

שם מקרה השימוש-	**צפייה בהיסטוריה**
תיאור:	צפייה בשרטוטים קודמים ופתרונם
תנאי מקדים:	משתמש רשום
שחקנים:	משתמש קיים
הזנק:	משתמש לוחץ על כפתור צפייה בהיסטוריה
מסלול עיקרי:	1. המשתמש לוחץ על כפתור צפייה בהיסטוריית עבודות שלו. 2. המערכת פונה לטבלת ההיסטוריה של המשתמש ,שולף את הנתונים מציג אותם למשתמש
תנאי סיום:	פרטי העבודות הקודמות שעשה מוצגים על המסך
תדירות:	פעם בשבועיים

שם מקרה השימוש-	התחברות
תיאור:	התחברות למערכת
שחקנים:	משתמש קיים
תנאי מקדים:	משתמש רשום
הזנק:	מייל או סיסמה אינם תקינים
מסלול עיקרי:	1. השתמש מכניס מייל וסיסמא. 2. המערכת בודקת בבסיס נתונים האם המשתמש כבר קיים במערכת ובמידה והוא קיים- המערכת תעבור לדף האלגוריתם פרישת מצלמות אבטחה.
תנאי סיום:	ההתחברות עברה בהצלחה
תדירות:	פעם בשבועיים

12.3 מבני נתונים

Dictionary -מילון -באפליקציה השתמשתי במבנה נתונים dictionary השומר נתונים בפרמט <key, value> לצורך אחסון מידע זמני על משתני הכמות שהגדרנו בפרויקט. כגון: מספר סוגי המצלמות, מספר הזוויות האופקיות הניתנות להתקנה וכדומה.

List- רשימות מקושרות- השתמשתי רבות ברשימות מקושרות בפרויקט שלי לאחסון נקודות אפשריות להתקנת מצלמות אבטחה, רשימה של נקודות לכיסוי ועוד.

Tuple- רשימה בלתי משתנה- tuple מהווה רשימה בלתי משתנה (immutable list) מרגע יצירת רשימה כזו, לא ניתן עוד לשנות את תכנה בשום צורה – אך ניתן לגשת אליו למטרת קריאה. השתמשתי בה בעיקר לשמירת נקודות על הציר עם X ו Y כמו ברשימת מיקומי המצלמות האפשריים או רשימת הנקודות לכיסוי.

Matrix- מטריצה- מערך רב מימדי, השתמשתי בו כדי לאתחל את המשתנה V שהוא משתנה בינארי המייצג האם בנקודה K חשופה ונראית על יד המצלמה שבמיקום i בזוית אופקית j בזוית אנכית d ובגובה e ובזווית ראיה t .

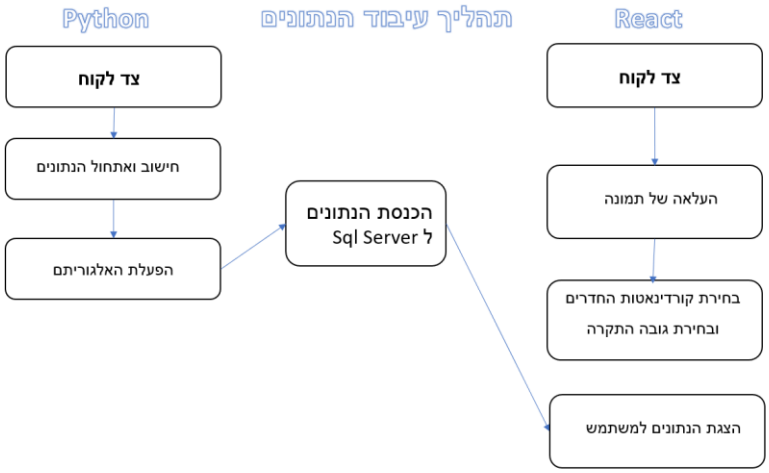
12.4 הקשרים בין היחידות השונות

הפרויקט שבצד השרת מתחיל ביחידה שנקראת MainAlgorithm שהיא אחראית על ניהול הפונקציות, היחידה מקבלת מהממשק רשימה של אובייקטים של חדר שמכילים את הקורדינאטות של כל חדר, וכן את הגובה של החדר/המבנה הרצוי. ביחידה זו האלגוריתם גם מחשב פרמטרים שונים הנצרכים על ידי היחידות השונות כמו כמות המצלמות הקיימות כמות המיקומים האפשריים של המצלמות וכו' וזאת הן על ידי חישוביים מקומיים והן על ידי פונקציות חיצוניות נוספות.

היחידה הבאה היא בניית בעיית האופטימיזציה הבינארית binary integer programming ופתרונה, היחידה קוראת ליחידה הנוספת שאחראית על בניית האילוצים ופונקציית המטרה ושולחת את התוצאה לפתרון הבעיה על ידי הפותר pulp .

היחידה הנוספת היא זו שאחראית על יצירת אילוצי בעיית האופטימיזציה ופונקציית המטרה. ומחזירה אותם ליחדה הקדמת.

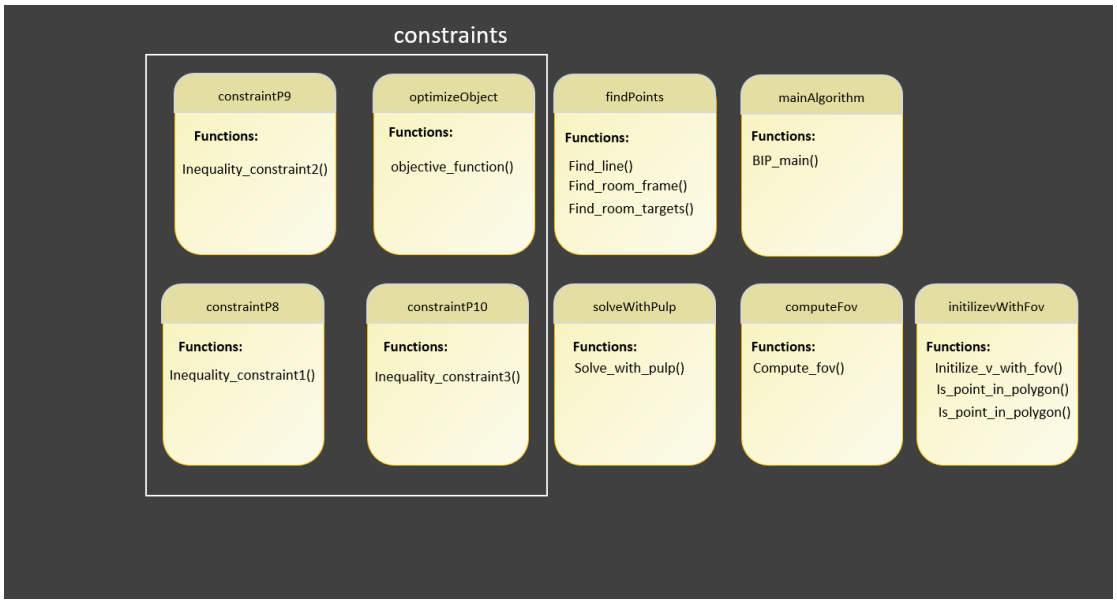
12.5 עץ מודולים



12.9 תרשים של כל המחלקות בפרויקט

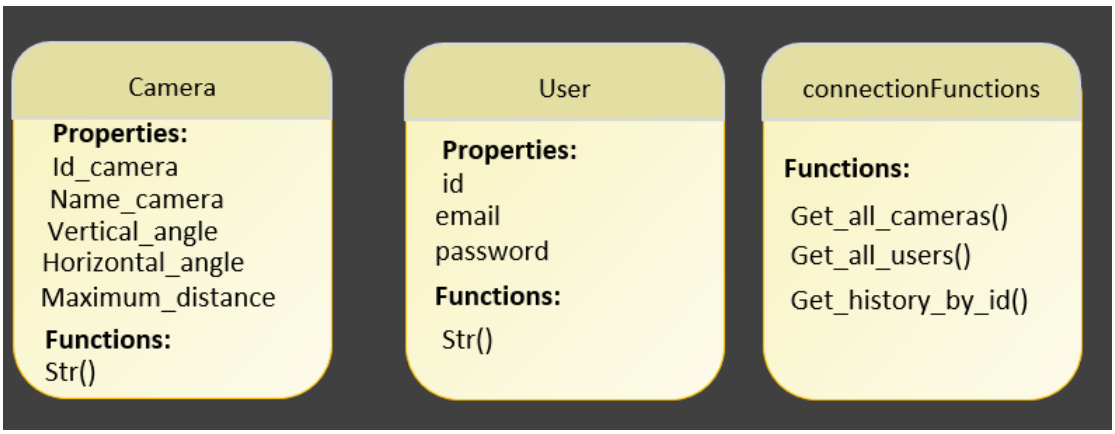
algorithm

מחלקה זו אחראית על מימוש האלגוריתם למציאת מספר ומיקומי מצלמות האבטחה הנצרכות.



models

מחלקה זו אחראית על שליפת הנתונים מה Sql Server



13 רכיבי ממשק

.internet explorer, chrome, Firefox

14 תיכנון המערכת

שרת ולקוח.

14.1 ארכיטקטורת המערכת

WebApi

14.2 תיכנון מפורט

Server (שרת) = Python

Client (לקוח) = React

14.3 חלופות המערכת

שרת- Java, C#, C++ אלו שפות שמתאימות לחישובים מתמטיים כמו שנצרך בפרויקט שלית למעשה בגלל מורכבות האלגוריתם של הפרויקט שלי החלטתי להשתמש בשפת Python הכוללת מבנים המיועדים לאפשר ביטוי של תכניות מורכבות בדרך קצרה וברורה. וכן עקב הצורך בשימוש בספריות בצורה פשוטה וברורה.

לקוח- Angular, בחרתי להשתמש ב React בגלל הפשטות וחופש הפעולה שהוא מספק איתו, המבנה הגמיש שלו והקלות ללמוד אותה.

15 תיאור התוכנה

15.1 סביבת עבודה

PyCharm 2023.3, visual studio code

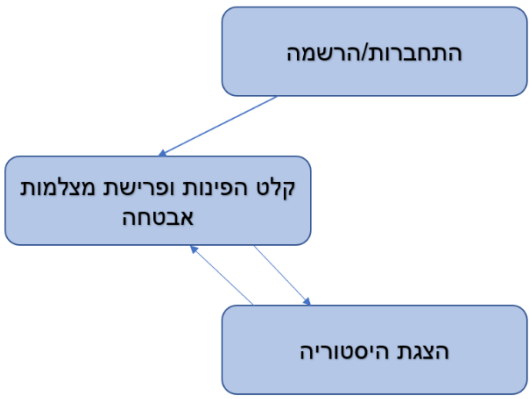
15.2 שפות תכנות

Python, React כמו כן שימוש בטכנולוגיית WebApi .

16 תיאור מסכים

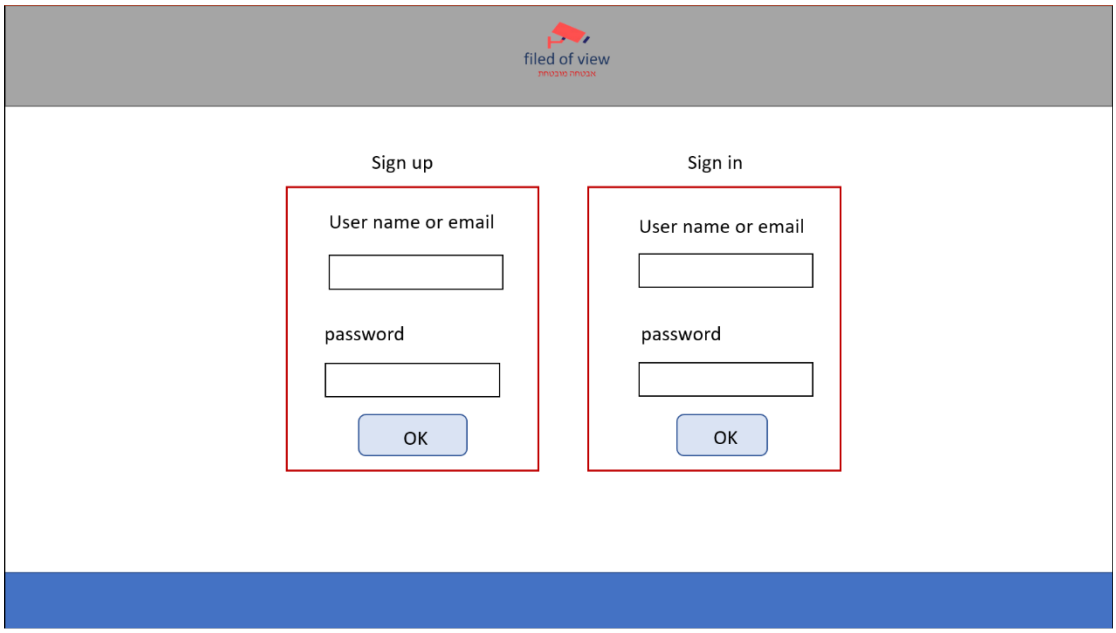
- מסך התחברות או הרשמה
- מסך בחירת חדרים והצגת הפיתרון
- מסך הצגת מסכים קודמים

17 תרשים מסכים



18 פירוט ה מסכים

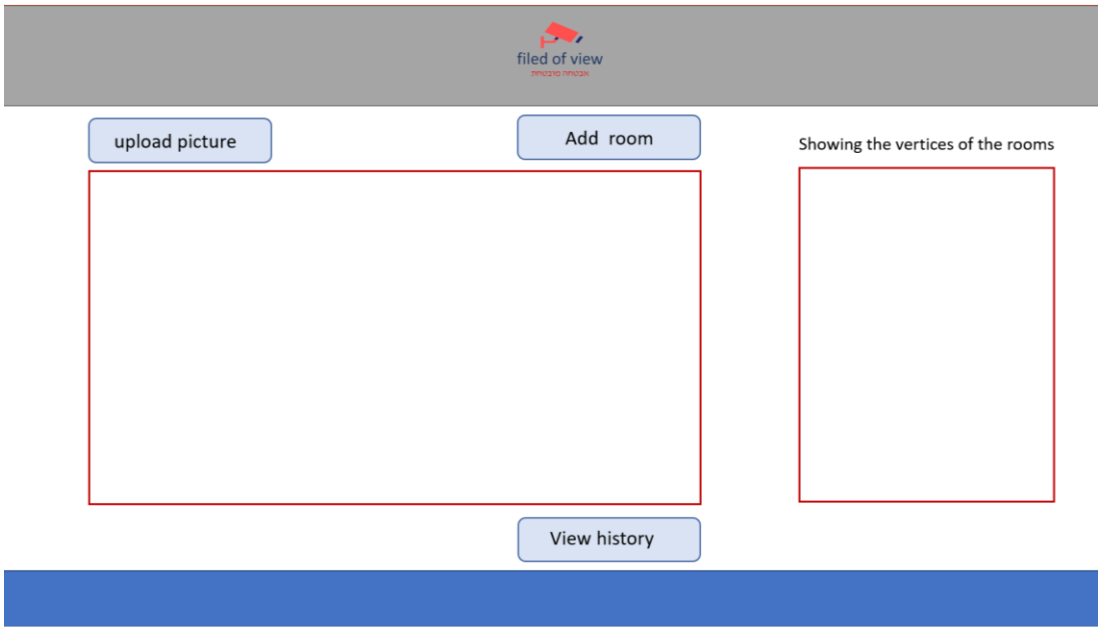
מסך התחברות/הרשמה-



זהו מסך הפתיחה, בו בכניסה למערכת המשתמש יזין את כתובת המייל שלו ואת הסיסמא במקום המתאים: להרשמה- אם הוא אינו רשום במערכת והתחברות- במידה והוא קיים במערכת.

בלחיצה על כפתור השליחה (OK) במידה ושם המשתמש והסיסמא מתאימים אנחנו נעבור למסך הבא.

מסך פרישת מצלמות אבטחה



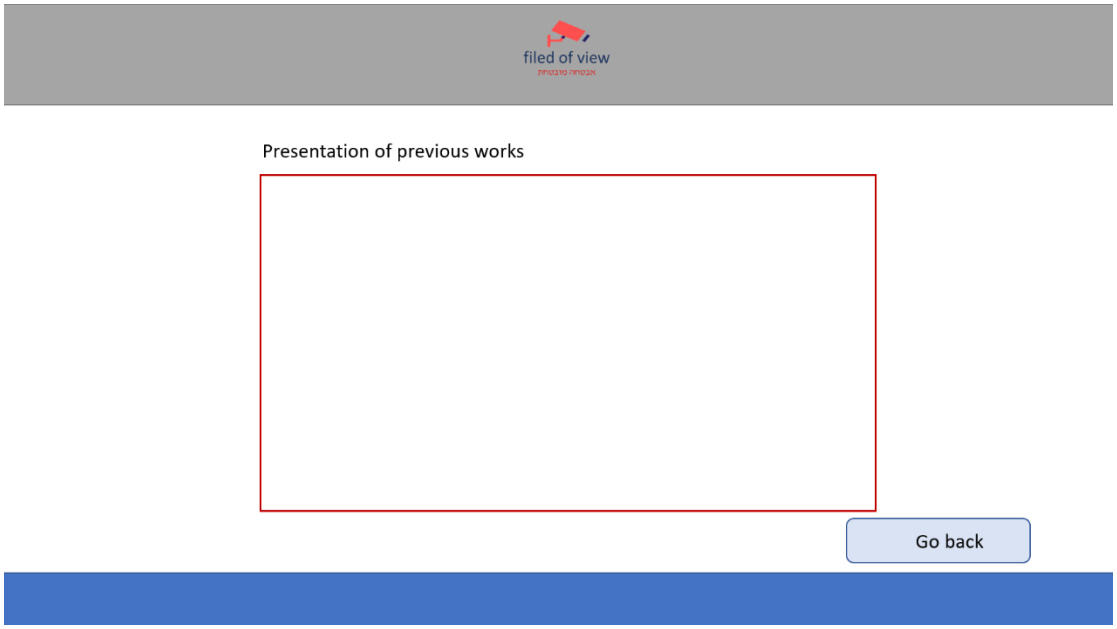
במסך הנוכחי המשתמש יעלה תמונה של שרטוט אדריכלי שיוצג באזור תצוגה השמאלי.

מעליו ישנו כפתור להעלאת תמונה וכן כפתור להוספת חדר לפרישתו במצלמות.

ובכל פעם שיבחרו חדר נוסף יתווסף לאזור תצוגה השמאלי אובייקט לרשימה עם קודקודי החדר.

הכפתור התחתון מוביל אותנו לתצוגת היסטורית העבודות שבמסך:

מסך היסטורית עבודות



זהו מסך שבו יוצגו כל או חלק מהמבנים הקודמים שהמשתמש ביקש לפרוש בהן מצלמות אבטחה.

ישנו האזור האמצעי שבו יוצגו העבודות.

ויש כפתור חזרה שיחזיר את המשתמש לעמוד הקודם.

19 קוד התוכנית

הפונקציה הראשית- mainAlgorithm

זוהי הפונקציה הראשית של האלגוריתם והיא מפעילה את שאר החלקים של האלגוריתם עד לקבלת תוצאה סופית.

אתחול משתני הספירה שמונים את כמות מיקומי המצלמות האפשריים, את מספר נקודות הכיסוי , את מספר הזוויות האנכיות שיתכן ויתקינו את המצלמה, את מספר הזוויות האופקיות, את מספר הגבהים, ואת מספר סוגי המצלמות שמתבטא בעיקר בגודל בזווית ראיה של המצלמה.

בכל אלו נשתמש בהמשך בכדי לעבור על כל האפשרויות למיקומי המצלמה והשדה ראייה שלהם.

```
import solve_with_pulp
import cameras.retrieving_the_cameras_data as dataFuncs
from my_coverage_algorithm.BIP_by_pulp import find_room2
from my_coverage_algorithm import Initialize_the_quantity_variable as
init

def BIP_main(list_of_tuples_with_the_xy_cordinates = [(0, 0), (9, 0),
(9, 9), (0, 9)], heightOfRoomChosenByUser = 2.50):#x,v,y,nc, nhd,
nvd, ne, na,nt

    #המצלמות נתוני שליפת
    dataCameras, numOfCameras = dataFuncs.get_all_cameras()
    pointInWalls =
find_room2.find_room_frame(list_of_tuples_with_the_xy_cordinates)  #
למצלמות אפשריים מיקומים

    # number of target positions- מספר עמדות יעד
    NT, listOfTargets= init.sum_the_target_position(pointInWalls)

    # number of camera positions- מספר עמדות המצלמה
    NC = init.sum_the_cameras_positions(pointInWalls)


    # number of vertical orientations- מספר כיוונים אנכיים
    numOfVerticalOriens = []
    NvD = 1
    if dataCameras != None:
        for i in range(len(dataCameras)):
            theVer = dataCameras[i]
            numOfVerticalOriens.append(theVer.vertical_angle)
    if (numOfVerticalOriens != []):
        numOfVerticalOriens.sort(reverse=True)
        NvD = numOfVerticalOriens.pop()

    # number of heights- מספר גבהים
    heightOfRoomChosenByUser *= 100
    NE = 0
    if (heightOfRoomChosenByUser > 280):
        NE = 1
    else:
        NE = 280 - heightOfRoomChosenByUser
    NE = int(NE)

    # number of camera types- מספר סוגי מצלמות
    NA = numOfCameras

    # number of horizontal orientations- מספר כיוונים אופקיים.
    numOfHorizontalOriens = []
    NhD = 1
    if dataCameras != None:
        for i in range(len(dataCameras)):
```

```
            theVer = dataCameras[i]
            numOfHorizontalOriens.append(theVer.vertical_angle)
    if (numOfHorizontalOriens != []):
        numOfHorizontalOriens.sort()
        NhD = numOfHorizontalOriens.pop()
    NhD =
int(init.sum_of_horizontal_orientations(list_of_tuples_with_the_xy_co
rdinates, NhD))


    # given minimal coverage rate- מינימלי כיסוי שיעור בהינתן
    CVR = 0.9


   sol = solve_with_pulp.solve_with_pulp(NC=NC, NhD=NhD, NvD=NvD,
NE=NE, NA=NA, NT=NT, CVR=CVR, listOfTargetPositions=listOfTargets)

BIP_main()
```

פונקצית המטרה

בכדי לפתור את הבעיה, האלגוריתם בונה בעיית אופטימיזציה בינארית כאשר פונקציית המטרה הינה ייצוג של מספר המצלמות הנדרש על ידי פונקצית המטרה לבעיית מינימום

```
import pulp


def objective_function(prob, x, NC, NhD, NvD, NE, NA):
    prob += pulp.lpSum(x[(i, j, d, e, t)]
                       for i in range(NC)
                       for j in range(NhD)
                       for d in range(NvD)
                       for e in range(NE)
                       for t in range(NA)), "Minimize number of
cameras"
```

פונקציות האילוצים

פונקציות אלו אחראיות לוודא שכל נקודות המטרה (השטח לכיסוי) אכן מתכסות על ידי מצלמה כלשהי, שמספר המצלמות אינו רב יתר על המידה וכן שנקודות הכיסוי אינן מכוסות על ידי מצלמות אבטחה רבות.

```
# Inequality constraint function
import pulp

# אם לפחות אחת במצלמה מכוסה מטרה שכל מבטיח y[k]היא 1.
```

```
def inequality_constraint1(prob, x, y, v, NC, NhD, NvD, NE, NA, NT):
    for k in range(NT):
        prob += pulp.lpSum(v[i][j][d][e][t][k] * x[i, j, d, e, t] for
i in range(NC) for j in range(NhD)
                           for d in range(NvD) for e in range(NE) for
t in range(NA)) >= y[k]
# Inequality constraint function
import numpy
import pulp

#המצלמות ממספר יותר ידי על תכוסה לא מטרה שאף מבטיח.
def inequality_constraint2(prob, x, y, v, NC, NT, NhD, NvD, NE, NA):
    for k in range(NT):
        prob += pulp.lpSum(v[i][j][d][e][t][k] * x[i, j, d, e, t] for
i in range(NC) for j in range(NhD)
                           for d in range(NvD) for e in range(NE) for
t in range(NA)) <= NC * y[k]

import pulp

#כהלכה מיושם המינימלי הכיסוי שיעור שאילוץ מבטיח.
def inequality_constraint3(prob, y, NA, NT, CVR):
            prob += pulp.lpSum(y[k] for k in range(NT)) >= NT * CVR
```

פונקציית הפתרון- solveWithPulp

הפונקציה הזו אחראית על שליחת המשתנים לפונקציות לבניית בעיית אופטימיזציה ולבסוף לשליחתם לפתרון על ידי פותר בעיות אופטימיזציה עם משתנים בינאריים מספריית Pulp

```
import pulp
import constraints.constraintP8 as con8
import constraints.constraintP9 as con9
import constraints.constraintP10 as con10
import optimize_object_pulp
from my_coverage_algorithm import initilize_v_with_fov
from my_coverage_algorithm.BIP_by_pulp import find_room2
#from flask import Flask, request, jsonify
#from flask_cors import CORS
# import cameras.retrieving_the_cameras_data as dataFuncs
# from my_coverage_algorithm.BIP_by_pulp import find_room2
# from cameras import Camera
#
# app = Flask(__name__)
# CORS(app)

def solve_with_pulp(NC, NhD, NvD, NE, NA, NT, CVR,
listOfTargetPositions):
```

```
    # Define the problem
    print("open prob")
    prob = pulp.LpProblem("Camera_Optimization", pulp.LpMinimize)

    dictionaryOfTheCountersParam = {
        "NC": NC,
        "NhD": NhD,
        "NvD": NvD,
        "NE": NE,
        "NA": NA,
        "NT": NT,
        "CVR": CVR
    }

    print("params")
    # Assume V is a predefined 6-dimensional array that represents
visibility
    v =
initilize_v_with_fov.initilize_v_with_fov(dictionaryOfTheCountersPara
m, listOfTargetPositions)

    # Decision variables
    x = pulp.LpVariable.dicts("x", ((i, j, d, e, t) for i in
range(NC) for j in range(NhD)
                                    for d in range(NvD) for e in
range(NE) for t in range(NA)), cat='Binary')
    y = pulp.LpVariable.dicts("y", range(NT), cat='Binary')

    # Define the objective function
    optimize_object_pulp.objective_function(prob, x, NC, NhD, NvD,
NE, NA)
    print("constraints")
    # Add constraints
    con8.inequality_constraint1(prob, x, y, v, NC, NhD, NvD, NE, NA,
NT)
    print("con1")
    con9.inequality_constraint2(prob, x, y, v, NC, NT, NhD, NvD, NE,
NA)
    print("con2")
    con10.inequality_constraint3(prob, y, NA, NT, CVR)
    print("solving")
    # Solve the problem
    prob.solve()

    # Print the results
    print("Status:", pulp.LpStatus[prob.status])
    for v in prob.variables():
        print(v.name, "=", v.varValue)

    print("Optimal number of cameras:", pulp.value(prob.objective))
```

פונקציית האיתחול של חלק מהמשתני מנייה

```
import math

from my_coverage_algorithm.BIP_by_pulp import find_room2

# <editor-fold desc="אופקית זווית חישוב">
def distance(point1, point2):
    return math.sqrt((point2[0] - point1[0]) ** 2 + (point2[1] -
point1[1]) ** 2)

def angle(p1, p2, p3):
    dx1 = p1[0] - p2[0]
    dy1 = p1[1] - p2[1]
    dx2 = p3[0] - p2[0]
    dy2 = p3[1] - p2[1]
    dot_product = dx1 * dx2 + dy1 * dy2
    magnitude1 = math.sqrt(dx1 ** 2 + dy1 ** 2)
    magnitude2 = math.sqrt(dx2 ** 2 + dy2 ** 2)
    return math.degrees(math.acos(dot_product / (magnitude1 *
magnitude2)))

def largest_angle(coordinates):
    largest = 0
    for i in range(len(coordinates)):
        p1 = coordinates[i]
        p2 = coordinates[(i + 1) % len(coordinates)]
        p3 = coordinates[(i + 2) % len(coordinates)]
        ang = angle(p1, p2, p3)
        if ang > largest:
            largest = ang
    return largest

def sum_of_horizontal_orientations(coordinates,
min_horizontal_angle_of_camera):
    large_angle_of_corner = largest_angle(coordinates)
    if large_angle_of_corner > min_horizontal_angle_of_camera:
        return large_angle_of_corner/min_horizontal_angle_of_camera
    else:
        return 1
# </editor-fold>

#החדר מתאר שעל הנקודות את מקבלת - מצלמות להצבת נקודות לסכימת פונקצית
def sum_the_cameras_positions(pointInWalls):
    children_len = [len(child) for child in pointInWalls]  #סכימת
הנקודות
    camPosi = sum(children_len)
    return camPosi
```

```
#החדר לכיסוי הנקודות את מקבלת - הכיסוי נקודות לסכימת פונקציה
def sum_the_target_position(pointInWalls):
    dirOfTheTargetPoints = find_room2.find_room_targets(pointInWalls)
    numOfTheTargetPoints = 0
    listOfTargetPositions = []
    #לרשימה הנקודות המרת
    for key, tuple_list in dirOfTheTargetPoints.items():
        #print(f"Key: {key}")
        for tuple_item in tuple_list:
            numOfTheTargetPoints += 1
            listOfTargetPositions.append(tuple_item)
  #  print(listOfTargetPositions)
    return numOfTheTargetPoints, listOfTargetPositions



# # Example usage:
# coordinates = [(0, 0), (4, 0), (4, 3), (0, 5)]  # Example
coordinates of the room's corners
# p =
sum_the_target_position(find_room2.find_room_frame(coordinates))
# print(type(p))
# print("Largest angle:", sum_of_horizontal_orientations(coordinates,
20))
```

פונקצית האיתחול של המשתנה V

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
from my_coverage_algorithm.BIP_by_pulp import find_room2
from my_coverage_algorithm import FOV

# ה- FOV של הטווח בתוך שנמצאות הנקודות לחישוב פונקציה
def is_point_in_polygon(point, polygon):
    path = Path(polygon)
    return path.contains_point(point)

# הFOV ידי על מכוסות נקודות האילו המסומנת בוליאנית מטריצה יוצר
def generate_boolean_matrix(target_matrix, fov_vertices):
    rows, cols = target_matrix.shape
    bool_matrix = np.zeros((rows, cols), dtype=bool)

    for i in range(rows):
        for j in range(cols):
            point = (j, i)  # (x, y)
            if is_point_in_polygon(point, fov_vertices):
```

```
                bool_matrix[i, j] = True

    return bool_matrix
def initilize_v_with_fov(dictionary_of_the_counters_paramerters,
list_of_target_psitions):
    # <editor-fold desc="ולמחוק לדינאמי לשנות FOV-מ איתחול">
    #המצלמה פרטי את שתשלוף פונקציה לכתוב
    print(1)
    epsilon = 7  # height (m)
    theta1 = 80  # horizontal angle of view (degrees) של הזווית גודל
המצלמה!!!!
    theta2 = 60  # vertical angle of view (degrees) - Adjusted to
ensure the angle sum is < 90
    psi = 20  # vertical angle of the camera (degrees)
    phi = 45  # horizontal angle of the camera (degrees) של הכיוון
המצלמה!!!!
    T = 60  # maximum recognition distance (m)//מהמשתמש לקבל
    x0 = 0  # camera installation x-coordinate (m)
    y0 = 0  # camera installation y-coordinate (m)

# <editor-fold desc="לתוך וכו המיקומים כמות של הערכים כל שליפת
משתנים">
NC = dictionary_of_the_counters_paramerters['NC']
NhD = dictionary_of_the_counters_paramerters['NhD']
NvD = dictionary_of_the_counters_paramerters['NvD']
NE = dictionary_of_the_counters_paramerters['NE']
NA = dictionary_of_the_counters_paramerters['NA']
NT = dictionary_of_the_counters_paramerters['NT']
CVR = dictionary_of_the_counters_paramerters['CVR']
# </editor-fold>

#באפסים V איתחול
v = np.zeros((NC, NhD, NvD, NE, NA, NT))

#list_of_target_psitions = (0.0, 0.0), (20, 6), (4, 4)
NT = list_of_target_psitions.__len__()

# <editor-fold desc="הנקודה האם וסימון נקודה לכל האפשרויות כל על מעבר
המצלמה כל י"ע נצפית">
for i in range(NC):  # להתקנה הנקודות לכל
    for j in range(NhD):  # האופקיות הזוויות לכל
        for d in range(NvD):  # האנכיות הזוויות ולכל
            for h in range(NE):  # ההתקנה של הגבהים לכל
                for t in range(NA):  # מצלמה לכל ובודק
                    fov_vertices = FOV.compute_fov(epsilon, theta1,
theta2, psi, phi, T, x0, y0) # מחשב את הקורדינאטות של הFOV
                    for k in range(NT):  # כיסוי הנקודות כל על עובר
                        if k >= list_of_target_psitions.__len__():
                            continue
                        elif type(list_of_target_psitions) != list:
                            return None
```

```
                                break
                            else:
                                v[i, j, d, h, t, k] =
is_point_in_polygon(list_of_target_psitions[k], fov_vertices)
print(v)
# </editor-fold>

return v
def is_point_in_polygon(point, polygon):
    path = Path(polygon)
    return path.contains_point(point)


def generate_boolean_matrix(target_matrix, fov_vertices):
    rows, cols = target_matrix.shape
    bool_matrix = np.zeros((rows, cols), dtype=bool)

    for i in range(rows):
        for j in range(cols):
            point = (j, i)  # (x, y)
            if is_point_in_polygon(point, fov_vertices):
                bool_matrix[i, j] = True

    return bool_matrix
```

פונקצית field of view -computFov

הפונקציה הזאת מחשבת את שדה הראיה של מצלמה לפי נתוני ההתקנה שלה

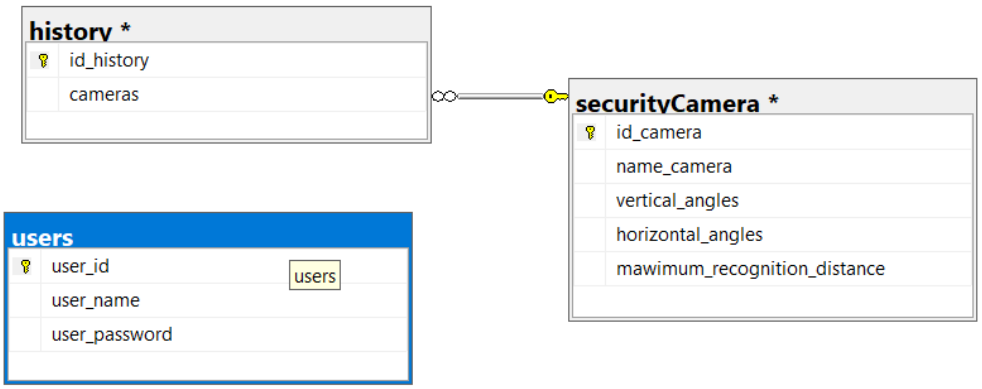
```
import time

import numpy as np

# FOV
def compute_fov(epsilon, theta1, theta2, psi, phi, T, x0, y0):
    print("fov")
    # Step 0: Compute tau and check conditions
    tau = epsilon / np.cos(np.radians(theta2 + psi))
    if (theta2 + psi) >= 90 or tau > T:
        raise ValueError("FOV cannot be computed with the given
parameters.")

    # Step 1: Compute the initial coordinates of the FOV vertices
    # שלב 1: חשב את הקואורדינטות הראשוניות של קודקודי ה-FOV
    h = epsilon
    tan_theta1_half = np.tan(np.radians(theta1 / 2))
    tan_psi = np.tan(np.radians(psi))
    cos_psi = np.cos(np.radians(psi))
    cos_theta2_psi = np.cos(np.radians(theta2 + psi))
    tan_theta2_psi = np.tan(np.radians(theta2 + psi))
```

```
# Vertex at the lower left (near the camera)
# התחתונה השמאלית בפינה קודקוד (המצלמה ליד)
p1_x = h * tan_psi
p1_y = (h / cos_psi) * tan_theta1_half

# Vertex at the lower right (near the camera)
# התחתונה הימנית בפינה קודקוד (המצלמה ליד)
p2_x = h * tan_psi
p2_y = (h / cos_psi) * -tan_theta1_half

# Vertex at the upper right (far from the camera)
# למעלה ימין בצד קודקוד (מהמצלמה רחוק)
p3_x = h * tan_theta2_psi
p3_y = (h / cos_theta2_psi) * tan_theta1_half

# Vertex at the upper left (far from the camera)
# העליונה השמאלית בפינה קודקוד (מהמצלמה רחוק)
p4_x = h * tan_theta2_psi
p4_y = (h / cos_theta2_psi) * (-tan_theta1_half)

# Step 2: Rotate the coordinates by angle phi
# שלב 2: זווית לפי הקואורדינטות את סובב phi
def rotate(x, y, angle):
    rad = np.radians(angle)
    x_new = x * np.cos(rad) - y * np.sin(rad)
    y_new = x * np.sin(rad) + y * np.cos(rad)
    return x_new, y_new

p1_x, p1_y = rotate(p1_x, p1_y, phi)
p2_x, p2_y = rotate(p2_x, p2_y, phi)
p3_x, p3_y = rotate(p3_x, p3_y, phi)
p4_x, p4_y = rotate(p4_x, p4_y, phi)

# Step 3: Translate to the actual installation coordinates
# שלב 3: בפועל ההתקנה לקואורדינטות תרגם
p1_x += x0
p1_y += y0

p2_x += x0
p2_y += y0

p3_x += x0
p3_y += y0

p4_x += x0
p4_y += y0

# Return the coordinates of the FOV vertices
# ה-FOV קודקודי של הקואורדינטות את החזר
return (p1_x, p1_y), (p2_x, p2_y), (p4_x, p4_y), (p3_x, p3_y)
```

[redacted]

20 תיאור מסד הנתונים

20.1 תרשים טבלאות

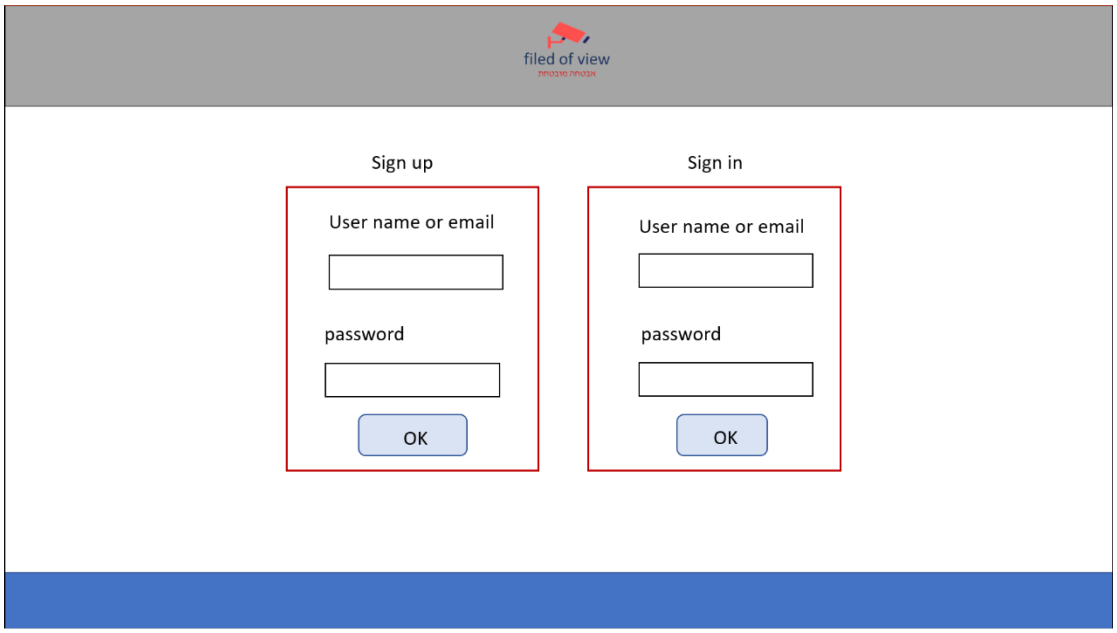


21 מדריך למשתמש

בעל בית ומתקין יקר!

ברוך הבא למדריך שלנו לשימוש באפליקצית field of view

ראשית עליך להיות רשום במערכת אם אינך רשום הירשם כעת ב sign up ובאם הינך רשום כבר התחבר למערכת על ידי הקשת כתובת המייל והסיסמא שלך בsign in

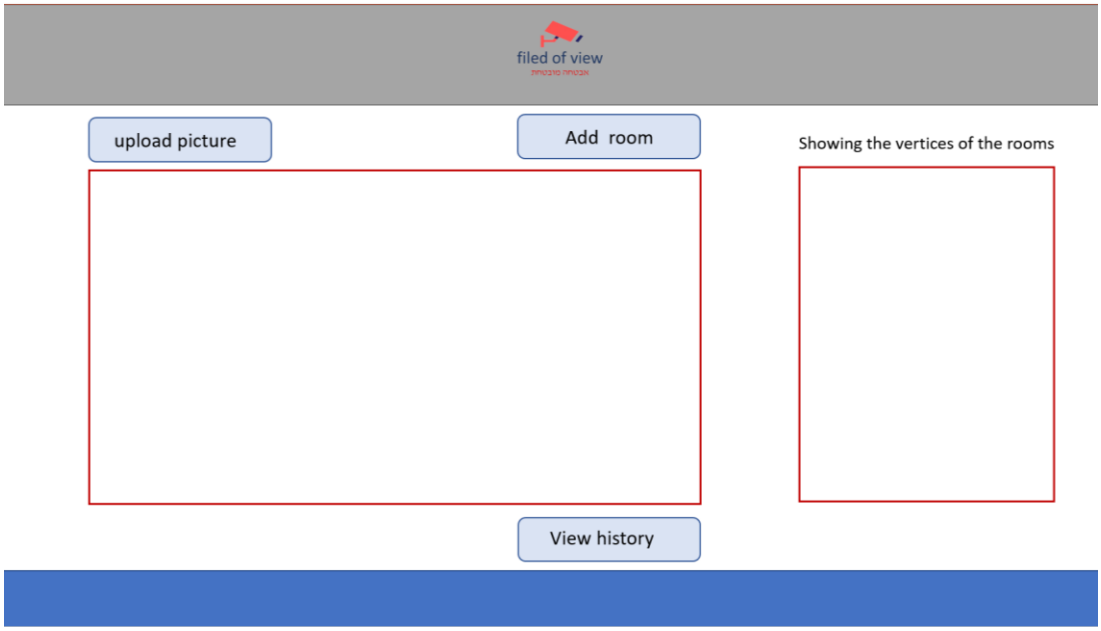


כעת העלה את השירטוט של המבנה שהיננך רוצה להתקין בו את המצלמות -על יד הכפתור upload picture.

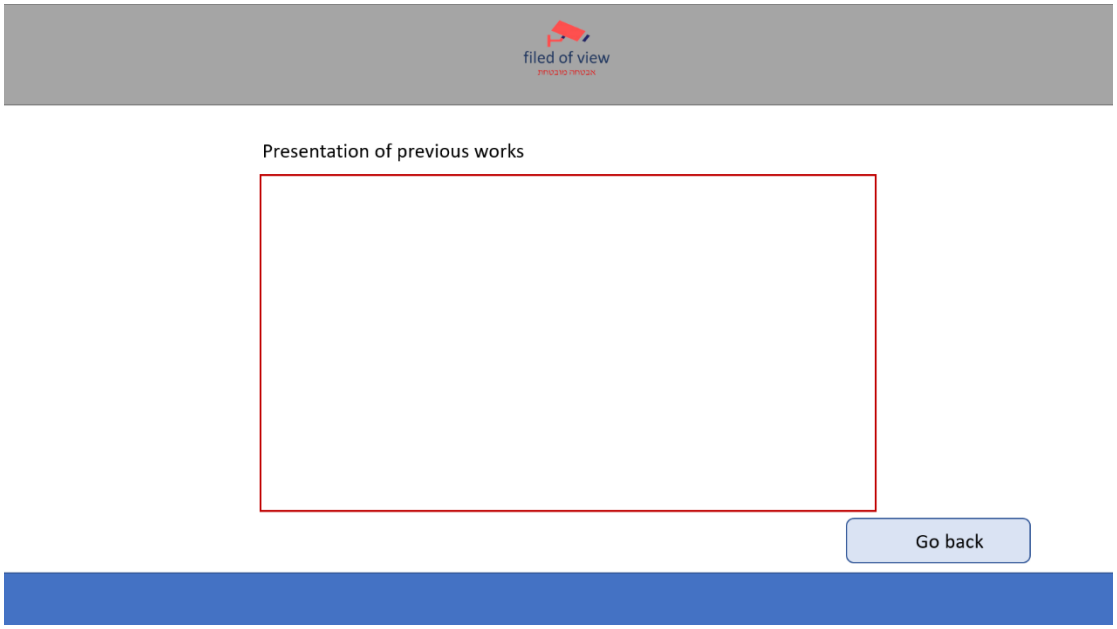
ולחץ לכל חדר שהינך רוצה לכסות עם מצלמות על הכפתור add room ולאחר מכן על כל פינות החדר. אם הינך רוצה לכסות חדרים נוספים חזור על הפעולה.

ולבסוף לחץ על כפתור הסיום.

לאחר מספר דקות יוצג בפניך התשובה.



אם את כבר לקוח ותיק והינך מעוניין לראות את כל העבודות האחרונות לחץ על הכפתור view history



מקווים שנהנית!

22 בדיקות והערכה

הפרויקט שלי השיג את מטרותיו להציב מצלמות אבטחה במקומות היעילים

23 ניתוח יעילות

התוצאה האופטימלית של הפרויקט שלי מתייחסת דווקא למיקום אופטימלי ולאו דווקא לזמן ריצה אופטימלי, ולכן זמן הריצה של הפרויקט שלי אינו אופטימלי כלל.

ניתוח זמן ריצה:

24 אבטחת מידע

ההתחברות למערכת נעשית באמצעות שם משתמש וסיסמא, שם המשתמש הוא יחיד,

וסיסמא שאורכה 8 תווים לפחות עם אותיות קטנות וגדולות באנגלית וללא תווים מיוחדים.

25 מסקנות

דבר ראשון וקודם לכל הבנתי שהדבר החשוב ביותר שחייבים להתחיל איתו זמן רב לפני תחילת המחשבה על הפרויקט הוא- חקר וכמה שיותר עמוק כולל המחשבות כיצד בדיוק ליישם כל דבר. המסקנה הזאת הגיעה בעקבות כך שבתחילה הבאתי את הרעיון הזה כמשהו קליל שאפשר בצורה פשוטה למדי לבצע אותו אך לאחר שקבילתי אישור על הפרויקט התחלתי לחקור לעומק על הנושא ופתאום גיליתי עד כמה עמוק ורחב העניין וכל מה שחשבתי כקליל התברר כבעל התפלספויות רבות. מה שגרם לתחילת הפרויקט שלי להידחות זמן רב למדי עד סיום שלב החקר.

בנוסף לכך התחדד לי העניין של החשיבות של הכנת תוכנית עבודה מפורטת ועד כמה היא מועילה לעמוד בתאריכי היעד.

26 פיתוחים עתידיים

- שיפור זמן הריצה של הפרויקט.
- התאמת האלגוריתם לחללים בעלי מכשולים.

27 בבליוגרפיה

מקורות לפרויקט אלגוריתמים:

https://www.sciencedirect.com/science/article/pii/S1570866712000196

https://www.sciencedirect.com/science/article/pii/S1570866712000196https://research.google//pubs/pub45757

https://research.google/pubs/pub45757/

https://web.cortland.edu/matresearch/BinaryInteger.pdf?&~nfopt(fileDistorted=4643197527226619)

https://github.com/elaugh9/Finding-the-minimum-number-of-surveillance-cameras-for-room-coverage

https://www.hindawi.com/journals/sp/2016/4801784/#model-and-solution

https://www.google.com/search?q=climbing+method+algorithms+%D7%95%D7%99%D7%A7%D7%99%D7%A4%D7%93%D7%99%D7%94&safe=active&sca_esv=1d6a025a864bdd9f&rlz=1C1GCEU_iwIL979IL979&sxsrf=ACQVn0_ULDR8sgqK9xj2WRcoFPKwSOShMQ%3A1706798537514&ei=ya27Zaj8HbGpptQP2rCF2A0&udm=&ved=0ahUKEwjohJ-EsIqEAxWxIIkEHVpYAdsQ4dUDCBA&uact=5&oq=climbing+method+algorithms+%D7%95%D7%99%D7%A7%D7%99%D7%A4%D7%93%D7%99%D7%94&gs_lp=Egxnd3Mtd2l6LXNlcnAiK2NsaW1iaW5nIG1ldGhvZCBhbGdvcml0aG1zINeV15nXp9eZ16TXk9eZ15RI5h5QlwVY2h1wAHgBkAEAmAEAoAEAqgEAuAEDyAEA-AEB4gMEGAAgQQ&sclient=gws-wiz-serp

/https://pypi.org/project/PuLP

/https://scipy.org

https://developers.google.com/s/results/optimization/?q=binary%20integer%20programming&hl=he%2F%3Fq%3Dbinary%20integer%20programming%20problems&text=binary%20integer%20programming