

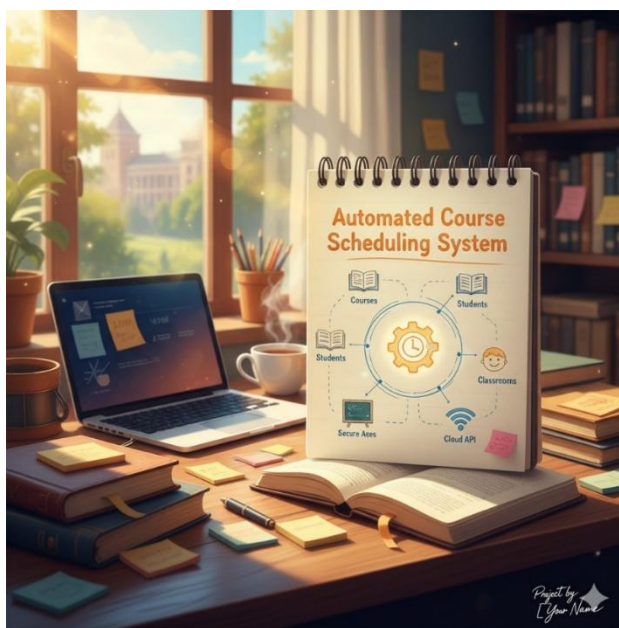


# הצעת פרויקט לעבודת גמר

י"ד הנדסאי תוכנה (שאלון 714918)

בנושא

מערכת לשיבוץ קורסים אקדמיים ע"פ אילוצים



שם הסטודנט: כפיר יוסף יעקובוב

ת"ז: 216232470

מנחה: אילן פרץ

תאריך: 13/1/2026

מכללה: כנפי רוח, קריית נוער ירושלים (סמל מוסד: 140129)

## תוכן עניינים:

1.	תיאור הנושא.....	3
2.	רקע תיאורטי.....	4
3.	תיאור הפרויקט.....	7
4.	הגדרת הבעיה האלגוריתמית.....	10
5.	הליכים עיקריים בפתרון בעיה בטכנולוגיות הנדסה מתקדמות.....	15
6.	הליכים עיקריים בתחום למידת מכונה - לא רלוונטי.....	21
7.	הליכים עיקריים במע' הפעלה/רשתות/תקשורת/אבטחה - לא רלוונטי.....	21
8.	תיאור פרוטוקולי תקשורת.....	21
9.	פיתוחים עתידיים.....	22
10.	תיאור טכנולוגיות הנדסה.....	22
11.	מסד נתונים.....	24
12.	פרטים פורמליים.....	26

## 1. תיאור הנושא

תחום שיבוץ המערכות (Scheduling) במוסדות אקדמיים מהווה אתגר לוגיסטי וחשובי מורכב, המשתיך למשפחת הבעיות של "סיפוק אילוצים" (Constraint Satisfaction Problems - CSP). במוסדות השכלה גבוהה, יצירת לוח זמנים סמסטריאלי אינה מסתכמת רק בשיבוץ טכני של קורסים, אלא דורשת פתרון אופטימיזציה רב-ממדי שנועד לסנכרן בין משאבים מוגבלים לבין דרישות סותרות. האתגרים והבעיות הקיימים בתחום: התהליך הקיים של בניית מערכת שעות מתבצע לעיתים קרובות באופן ידני או חצי-אוטומטי, מה שמוביל למספר כשלים מרכזיים:

- טעויות אנוש והתנגשויות: קושי במעקב אחר מאות אילוצים בו-זמנית גורם לשיבוצים כפולים (התנגשויות כיתות או מרצים) או ליצירת מערכות שאינן בנות-ביצוע עבור הסטודנטים.
- חוסר יעילות וניצול משאבים: ללא אלגוריתם אופטימיזציה, קשה לנצל את חדרי הלימוד ואת שעות ההוראה בצורה מיטבית, מה שמוביל לבזבז משאבים מוסדיים.
- מורכבות חישובית (NP-Hard): הבעיה מאופיינת בגידול מעריכי (אקספוננציאלי) של מרחב הפתרונות ככל שמספר הקורסים והאילוצים עולה. לכן, מציאת הפתרון "המושלם" בזמן סביר היא משימה כמעט בלתי אפשרית ללא שימוש בהיוריסטיקות מתקדמות.
- ריבוי אילוצים סותרים: הצורך לאזן בין אילוצים "קשיחים" (שאסור להפר, כגון מרצה שלא יכול ללמד בשני מקומות במקביל) לבין אילוצים "רכים" (העדפות נוחות של סטודנטים ומרצים) יוצר קונפליקטים שקשה לפתור ללא כלים ממוחשבים חכמים.

המטרה בתחום זה היא לפתח מנגנונים שיאפשרו מעבר משיבוץ ידני ומסורבל לשיבוץ דינמי, הממקסם את שביעות הרצון של כלל הגורמים (סטודנטים, מרצים והנהלה) תוך עמידה קפדנית בכל אילוצי המערכת.

השלכותיו של תכנון לקוי בתחום זה אינן נעצרות ברמה הלוגיסטית בלבד, אלא משפיעות באופן ישיר על איכות ההוראה וההישגים האקדמיים. מערכת שעות שאינה אופטימלית עלולה ליצור חסמים משמעותיים בפני סטודנטים, כגון חוסר יכולת להירשם לקורסי חובה עקב חפיפות בזמנים, מצב המוביל לעיתים קרובות להארכת משך התואר שלא לצורך ולפגיעה בחוויית הלימוד הכוללת. כמו כן, חלונות זמן ארוכים ובלתי סבירים בין הרצאות גורמים לבזבז זמן יקר עבור הסטודנטים והמרצים כאחד, ומפחיתים את האטרקטיביות של המוסד.

מן ההיבט המוסדי, ההסתמכות על שיטות ידניות גובה מחיר יקר במשאבי אנוש. סגלי ההוראה והמזכירות האקדמיות נדרשים להשקיע מאות שעות עבודה בכל תקופת רישום בניסיונות לפתור קונפליקטים נקודתיים, במקום להתמקד בפיתוח פדגוגי ובמחקר. בעידן של טרנספורמציה דיגיטלית, המגמה הרווחת כיום בעולם היא מעבר למערכות חכמות ("Smart Scheduling") המבוססות על אלגוריתמים, אשר מסוגלות לעבד כמויות אדירות של נתונים בזמן אמת.

מערכות אלו הופכות את תהליך השיבוץ לכלי ניהולי אסטרטגי, המאפשר חיסכון בעלויות תפעול (כגון צמצום ימי פעילות של קמפוסים שלמים) ושיפור דרמטי ברמת השירות הניתנת ללומדים ולסגל.

## 2. רקע תיאורטי

בעיית השיבוץ האקדמי נחשבת לאחת הבעיות המרכזיות בעולם האופטימיזציה הדיסקרטית ובתחומי הבינה המלאכותית, מאחר שהיא משלבת כמות גדולה של אילוצים ותלויות. כמו שהוזכר מקודם, בעיה זו שייכת למשפחת בעיות ה- Constraint Satisfaction Problems (CSP), בעיות שבהן נדרש למצוא פתרון חוקי מתוך מרחב חיפוש רחב, כאשר כל פתרון חייב לעמוד באוסף גדול של אילוצים לוגיים, מבניים וזמניים.

בבעיות CSP כל צעד משפיע על צעדים אחרים, ולכן גם החלטה שנראית נכונה בשלב מוקדם יכולה להיות שגויה בהמשך. מאפיין זה יוצר צורך בגישה אלגוריתמית שמאזנת בין עמידה מלאה בכל האילוצים לבין יעילות חישובית, במיוחד כאשר מתמודדים עם כמויות גדולות של קורסים, סטודנטים וזמינות מרצים.

בניגוד לבעיות פשוטות שבהן ניתן לבחור את הפתרון המידי הטוב ביותר, כאן מדובר בבעיה רב-ממדית: כל קורס תלוי בקורסים אחרים, כל מרצה זמין רק בחלק מהשבוע והמערכת כולה צריכה להגיע ללוח זמנים אקדמי תקין, חוקי ויציב.

לכן, לפני בחירת האלגוריתמים המתאימים לפרויקט, נבחנו מגוון גישות שונות - חלקן קלאסיות וחלקן מבוססות היוריסטיקות - כדי להבין אילו מהן מתאימות בצורה הטובה ביותר להתמודד עם סוג כזה של בעיות.

אלגוריתמים שנבחנו אך לא נבחרו:

אלגוריתמים חמדניים - Greedy Algorithms: האלגוריתמים החמדניים מבוססים על בחירה מקומית של האפשרות שנראית כרגע כטובה ביותר, ללא התייחסות למצב הכללי העתידי. למרות יעילותם והפשטות שבהם, הם מתאימים למערכות שבהן כל החלטה עומדת בפני עצמה, או כאלו שבהן התלות בין הצעדים קטנה. בבעיית השיבוץ האקדמי יש תלות הדדית בין עשרות ואף מאות אילוצים: קורסים שאסור שיתנגשו, מרצים שמלמדים על פי אילוצים אישיים, סטודנטים שלומדים ומתכננים את המערכת על פי צרכיהם האישי, תנאי קדם שנוגעים לסדר ההוראה, וכו'...

בחירה "מהירה" שאינה מתחשבת בכל האילוצים עלולה ליצור לוח זמנים חלקיים שלא אפשריים או כזה שמפר עשרות תנאי מערכת. לכן, למרות יעילותם, האלגוריתמים החמדניים נפסלו.

אלגוריתמים גנטיים - Genetic Algorithms: אלגוריתמים גנטיים פועלים לפי עקרונות אבולוציוניים: יצירת אוכלוסיות של פתרונות, ביצוע מוטציות ושחזורים, ובחירה של הפתרונות המתאימים ביותר.

למרות שהם מתאימים היטב למרחבי חיפוש גדולים, הם סובלים ממספר בעיות בהקשר של שיבוץ אקדמי: הם אינם מבטיחים שהפתרון עומד בכל האילוצים הנוקשים. נדרש זמן רב עד להתכנסות לפתרון תקף. יש צורך בפונקציית מטרה מורכבת וגדולה במיוחד, שיכולה להאט את המערכת בצורה משמעותית. גם אחרי זמן רב, האלגוריתם עלול לספק פתרון "כמעט תקין", שאינו עומד בכל הדרישות. בעיית שיבוץ שבו כל אילוץ חייב להתקיים, ללא חריגות, אינה מתאימה לגישה אבולוציונית שמטבעה מקבלת לעיתים סטיות. לכן האלגוריתם הגנטי נפסל.

חיפוש מדומה - Simulated Annealing: Simulated Annealing הוא אלגוריתם שפועל בצורה הסתברותית ומנסה למצוא את הפתרון הכי טוב מתוך הרבה אפשרויות. הוא משתמש ברעיון של "חימום וקירור" - בהתחלה הוא מוכן לנסות גם פתרונות פחות טובים כדי לא להיתקע במקום לא מוצלח, וככל שהאלגוריתם מתקדם הוא מצמצם את הניסיונות האקראיים ומתקרב לפתרון יציב ואיכותי. למרות יתרונותיו בפתרון בעיות אופטימיזציה, השיטה אינה מתאימה לבעיה האקדמית מכמה סיבות: האלגוריתם אינו מבטיח פתרון חוקי, הוא עשוי להסתפק בפתרון טוב אך לא תקף מבחינה לוגית. כאשר יש הרבה אילוצים, האלגוריתם עלול להיתקע בפתרון חלקי ולא להגיע לפתרון הטוב ביותר. תהליך ה"קירור" שלו עשוי להיות איטי, במיוחד כשיש כמות גדולה של קורסים ושיבוצים לבדוק. כדי שהוא יעבוד היטב צריך לבנות פונקציית הערכה מדויקת שמכסה את כל האילוצים, וזה תהליך מסובך במיוחד. לכן, במערכת שיבוץ שבה כל אילוץ חייב להתקיים ללא פשרות. Simulated Annealing אינו מתאים, ולכן נפסל.

לאחר בחינת מגוון גישות, נבחרו שני אלגוריתמים מרכזיים שמשתלבים יחד לפתרון מיטבי של בעיית השיבוץ. **מיון טופולוגי** לסידור על פי תנאי קדם, ו- **Pruning / Branch-and-Bound** עם **Backtracking** and **Bound** להצבה המדויקת בלוח הזמנים.

**מיון טופולוגי (Topological Sort):** מיון טופולוגי הוא אלגוריתם המשמש ליצירת סדר תקין של איברים במערכת שבה קיימות תלותיים וכיווניות בין רכיבים. האלגוריתם מבטיח שכל איבר יופיע לאחר כל האיברים שהוא תלוי בהם, ולכן הוא מתאים במיוחד לפתרון בעיות הכוללות יחסי קדם והיררכיה. תוצאה זו מאפשרת עבודה מסודרת ועקבית עם מבנים מורכבים תוך שמירה על כללי התלות המוגדרים.

**Pruning / Branch-and-Bound עם Backtracking:** לאחר קביעת סדר בסיסי, ניתן להשתמש באלגוריתמי חיפוש רקורסיביים מסוג Backtracking לצורך בחינת שילובים אפשריים תחת מערכת אילוצים. במהלך החיפוש מתבצעות בדיקות שוטפות לעמידה באילוצים שונים, וכל ענף שאינו עומד בתנאים נחתך מוקדם (Pruning), דבר המפחית משמעותית את מרחב החיפוש וזמן החישוב. במקרים מסוימים משולבת גם שיטת Branch-and-Bound, מאפשרת להשוות בין פתרונות חלקיים, לקבוע חסמים, ולהעדיף מסלולי חיפוש מבטיחים יותר על פני אחרים.

להלן טבלה שמסבירה על היתרונות והחסרונות של כל אחד מן האלגוריתמים שנבחנו:

שם האלגוריתם	עיקרון הפעולה	יתרונות	חסרונות
<b>אלגוריתמים חמדניים - Greedy Algorithms</b>	בחירה מקומית של האפשרות הטובה ביותר. בכל צעד נתון, ללא הסתכלות על ההשלכות העתידיות.	- פשטות מימוש. - יעיל ומהיר לריצה.	- בחירה "טובה" כרגע עלולה להוביל למבוי סתום בהמשך. - אינו מתאים לבעיות עם תלות הדדית גבוהה, בין אילוצים.
<b>אלגוריתמים גנטיים - Genetic Algorithms</b>	חיקוי תהליך אבולוציוני: יצירת "אוכלוסיות" פתרונות, ביצוע מוטציות ושיפור הדרגתי של הדורות.	- מתאים מאוד למרחבי חיפוש עצומים. - יכול למצוא פתרונות יצירתיים לבעיות אופטימיזציה.	- אין הבטחה לפתרון חוקי: האלגוריתם עלול להחזיר פתרון "כמעט טוב" שמפר אילוץ קשיח. (למשל, התנגשות קורסי חובה) - זמן התכנסות ארוך.
<b>חיסול מדומה - Simulated Annealing</b>	חיפוש הסתברות המשלב "חימום" וקירור" כדי לצאת ממינימום מקומי ולהתכנס לפתרון יציב.	טוב, במניעת היתקעות בפתרונות לא אופטימליים.	- בדומה לאלגוריתמים גנטיים, עשוי להסתפק בפתרון שאינו תקף לוגית ב-100%. - דורש חישוב מורכב של פונקציית הערכה ותהליך "הקירור" איטי.
<b>מיון טופולוגי ו- Pruning עם Backtracking Branch-and-Bound</b>	חיפוש רקורסיבי שיטתי: הצבת משאבים וחזרה לאחור. אם נתקלים במבוי סתום, תוך "גיזום" ענפים שגויים מראש.	- מבטיח פתרון חוקי: לא משאיר קצוות פרומים. - גמישות מלאה בטיפול באילוצים מורכבים. - הגיזום חוסך זמן החישוב משמעותי.	במקרה הגרוע ביותר, זמן הריצה עלול להיות ארוך.

לסיכום, הבחירה לשלב בין מיון טופולוגי לבין **Backtracking** עם מנגנוני **Pruning** נובעת מהצורך להתמודד עם בעיית שיבוץ מורכבת בעלת ריבוי אילוצים, יחסי תלות בין קורסים, מגבלות מערכת ומשתמשים שונים. מיון טופולוגי מאפשר ליצור סדר היררכי-לוגי של הקורסים והמשאבים, כך שהמערכת מתחשבת מראש בתלויות מבניות ובכללים שאינם ניתנים להפרה.

לעומתו, מנגנון ה-Backtracking מספק גמישות מלאה להתאמת השיבוץ בפועל - תוך בחירת מסלולים אפשריים בלבד, בעוד ש-Pruning מונע בחינה של פתרונות שגויים עוד בשלבי החיפוש המוקדמים. שילוב זה מאפשר לאזן בין דיוק מקסימלי בעמידה בכל האילוצים לבין יעילות חישובית, ומבטיח שהמערכת לא רק מוצאת פתרון, אלא מוצאת את הפתרון הסביר והעקבי ביותר מתוך מרחב אפשרויות עצום.

באופן זה, שילוב האלגוריתמים מאפשר בניית מערכת שיבוץ אקדמית אמינה וגמישה, הנותנת מענה מלא לצרכים השונים של כלל המשתמשים, תוך שמירה על יעילות חישובית ועמידה בכל האילוצים המורכבים.

### 3. תיאור הפרויקט

מטרת המערכת ואופן פעולתה היא לבצע שיבוץ אוטומטי, חכם ואופטימלי של קורסים במוסד אקדמי, תוך התחשבות באילוצים מגוונים של מרצים, סטודנטים ומנהלי מערכת. המטרה היא לספק פתרון טכנולוגי מתקדם שיחליף את התהליך הידני והמסורבל של בניית מערכת שעות, ויאפשר ניהול יעיל, גמיש ודינמי של מערך הלימודים בכל סמסטר. המערכת תאפשר למשתמשים להזין אילוצים, לנהל נתונים ולהפעיל אלגוריתם שיבוץ מתקדם אשר יחשב מערכת שעות מאוזנת, ללא התנגשויות בין קורסים, מרצים או קבוצות לימוד. האלגוריתם ישאף להשיג אופטימיזציה מירבית וסיפוק הרצון של כלל המשתמשים.

חווית המשתמש והתאמה אישית המערכת פותחה כיישום אינטרנט הנגיש מכל דפדפן, על מנת לאפשר זמינות מלאה ונוחות מרבית למשתמשים ללא צורך בהתקנה מקומית. השימוש במערכת מדמה "דיאלוג" בין הסטודנט/המרצה לבין המערכת: לדוגמה, סטודנט יכול להגדיר אילוצים אישיים כגון עבודה בימים מסוימים (למשל ימים א' ו-ג' משעה 16:00) או להגדיר מגבלת שעות לימוד יומיות (למשל לא יותר מ-8 שעות ברצף). בהתאם לקלטים אלו, המערכת תחשב עבורו מערכת שעות מותאמת אישית, תוך שמירה על מגבלות כלליות כמו זמינות מרצים, תפוסת כיתות ומועדי קורסי חובה.

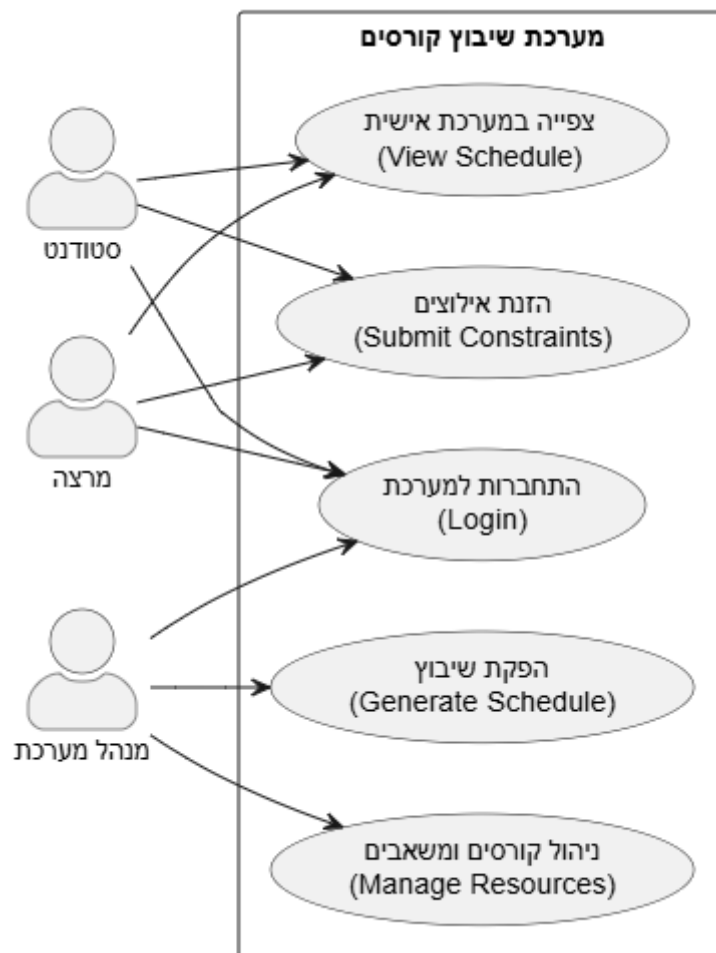
להלן תיאור השחקנים במערכת (Actors), המערכת משרתת שלושה סוגי משתמשים עיקריים, כאשר לכל אחד ממשק ייעודי המותאם לצרכיו:

מנהל המערכת (Admin): המשתמש בעל ההרשאות הגבוהות ביותר. המנהל אחראי על הזנת נתוני התשתית (רשימת הקורסים, כיתות לימוד, שיוך מרצים לקורסים). תפקידו המרכזי הוא להגדיר את פרמטרים של האלגוריתם, להפעיל את תהליך השיבוץ האוטומטי, ולבצע בקרת איכות על התוצאה הסופית לפני פרסומה.

מרצה (Lecturer): משתמש המזין את אילוצי ההוראה שלו. המרצה נכנס למערכת כדי להגדיר ימים ושעות בהם הוא אינו זמין להוראה, וכן להגדיר העדפות (כגון הימנעות משעות רצופות רבות מדי). לאחר השיבוץ, המרצה מקבל את מערכת השעות האישית שלו.

סטודנט (Student): משתמש הקצה העיקרי שנהנה מהגמישות של המערכת. הסטודנט מתחבר כדי לבחור קורסי בחירה ולהזין אילוצים אישיים (עבודה, מגורים רחוקים וכו'). המערכת מציגה לו את האפשרויות הרלוונטיות ביותר עבורו ומאפשרת לו לצפות בלוח הזמנים הסופי ובציונים.

להלן תרשים Use-Case המתאר את הליך הפרויקט:



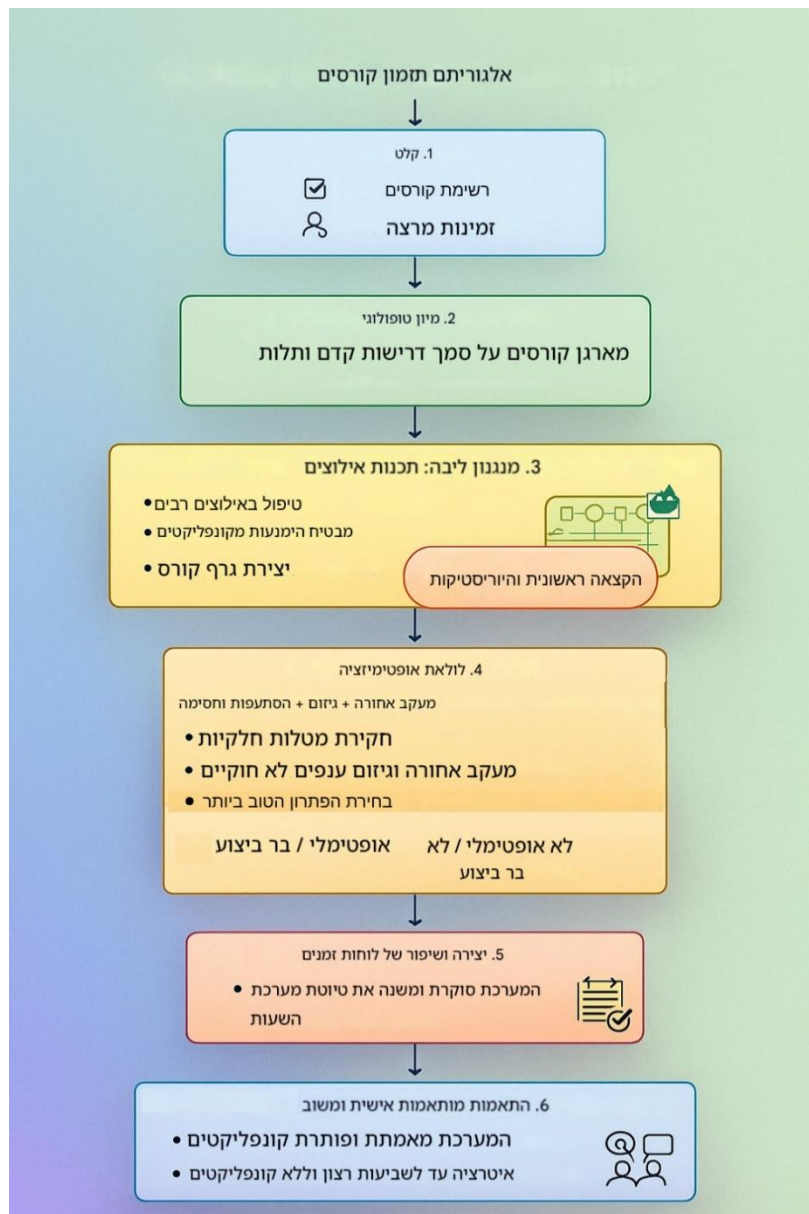
פירוט מקרי שימוש עיקריים (Use Cases) להלן תיאור הפעולות המרכזיות שהמשתמשים מבצעים במערכת:

- **התחברות למערכת (Login):** כל משתמש (סטודנט, מרצה או מנהל) נדרש להזדהות באמצעות שם משתמש וסיסמה בכניסה למערכת. המערכת מאמתת את הפרטים מול מסד הנתונים, ובמידה והם תקינים, היא טוענת את הממשק המתאים לפי רמת ההרשאות של המשתמש (למשל, מנהל יראה אפשרויות ניהול שסטודנט לא יראה).



- **הזנת אילוצים (Submit Constraints):** הסטודנט או המרצה ממלאים טופס מקוון ובו הם מציינים ימים ושעות חסומים. המערכת בודקת את תקינות הקלט ושומרת את האילוצים בבסיס הנתונים.
- **ניהול קורסים ומשאבים (Manage Resources):** מנהל המערכת מוסיף, עורך או מוחק קורסים מהמאגר, ומגדיר לכל קורס את דרישות הקדם שלו ואת המרצה האחראי.
- **הפקת שיבוץ (Generate Schedule):** פעולה המבוצעת על ידי המנהל. בלחיצת כפתור, המערכת מפעילה את מנוע השיבוץ (האלגוריתם), אשר מעבד את כל הנתונים והאילוצים ומפיק טיוטה של מערכת שעות.
- **צפייה במערכת אישית (View Schedule):** כל משתמש (סטודנט/מרצה) יכול לצפות בלוח הזמנים השבועי שלו, המיוצג ויזואלית בטבלה ברורה, ולראות את המיקום והשעה של כל הרצאה.

להלן תרשים המתאר את "זרימת העבודה של האלגוריתם":



תרשים זה מתאר את זרימת העבודה של אלגוריתם השיבוץ, החל משלב קליטת הנתונים הגולמיים (קורסים וזמינות מרצים) וארגונם הלוגי באמצעות מיון טופולוגי. בהמשך, המערכת מפעילה מנוע אילוצים המשלב לולאת אופטימיזציה וגזוז (Pruning) כדי לסנן פתרונות לא חוקיים ולמצוא את השיבוץ היעיל ביותר. התהליך מסתיים ביצירת לוח זמנים סופי וביצוע התאמות אישיות למניעת קונפליקטים, עד לקבלת תוצר מוגמר ותקין.

## 4. הגדרת הבעיה האלגוריתמית

הבעיה המרכזית שהמערכת נועדה לפתור היא **בעיית שיבוץ עם אופטימיזציה רב-ממדית** של קורסים במוסד אקדמי. את בעיית השיבוץ אני מתאר בצורה של גרף מכוון ללא מעגלים (גמ"ל) מדובר בבעיה קומבינטורית מורכבת שבה יש לשבץ קבוצה גדולה של משאבים (מרצים, אילוצים וסטודנטים) על פני ציר זמן מוגדר, תחת מערכת ענפה של אילוצים. הקושי המרכזי אינו מסתכם רק במציאת שיבוץ "חוקי" (כזה שמונע התנגשויות פיזיות), אלא במציאת שיבוץ אופטימלי הממקסם את פונקציית המטרה - שביעות הרצון של כלל הגורמים במערכת.

האילוצים בבעיה זו מתחלקים לשניים:

אילוצים "קשיחים": אילוצים שחובה לקיים כדי שהמערכת תהיה תקפה, כגון מניעת מצב שבו מרצה משובץ לשני שיעורים בו-זמנית, מניעת חפיפה בין קורסי חובה באותו סמסטר.

אילוצים "רכים": העדפות אישיות ואופטימיזציה, כגון ניסיון להיענות לבקשות סטודנטים לימי חופש מסוימים, צמצום "חלונות" במערכת, והתחשבות במגבלות שעות עבודה של מרצים.

מבחינה חישובית, מדובר בבעיית NP-Hard. המשמעות היא שמרחב הפתרונות האפשריים גדל באופן מעריכי (אקספוננציאלי) ככל שמספר הקורסים והאילוצים עולה, ובלתי אפשרי לסרוק את כל האפשרויות בזמן סביר כדי למצוא את הפתרון המושלם. לכן, האתגר האלגוריתמי דורש שימוש בפתרון חכם (כגון אלגוריתם יוריסטי) המצמצם את מרחב החיפוש וממקד את המערכת רק באפשרויות בעלות פוטנציאל גבוה להצלחה. מטרת האלגוריתם היא למצוא "נקודת שיווי משקל" אופטימלית בין כל האילוצים המתנגשים, כך שיינתן פתרון יעיל ומאוזן שלא יפגע ביציבות המערכת הכללית.

בשביל לפתור את בעיה זו בחרתי בשני שלבים שיפתרו לי את הבעיה בצורה היעילה ביותר: **מיון טופולוגי (Topological Sort)**: השלב הראשון בפתרון הוא יצירת סדר תקין של קורסים בהתאם לתנאי קדם. באמצעות מיון טופולוגי מתקבל סדר היררכי הגיוני של הקורסים - כך לדוגמה "מבוא לתכנות" מופיע לפני "אלגוריתמים". השלב הזה מבטיח שכל רצף לימודי עומד בכללי והדרישות האקדמיות

**Pruning / Branch-and-Bound עם Backtracking** : לאחר יצירת הסדר הבסיסי, המערכת

מפעילה מנגנון חיפוש רקורסיבי שמציב קורסים בלוח הזמנים.

האלגוריתם מבצע בדיקות בזמן אמת:

זמינות מרצה, עמידה בתנאי קדם, מגבלות עומס ועוד. בכל מצב שבו אפשרות לא עומדת באילוצים, היא נגזרת מיד (Pruning), וכך נחסך זמן חישוב משמעותי. במידת הצורך משולבת גם שיטת Branch-and-Bound, המאפשרת לתעדף פתרונות ולהתקדם במסלול החיפוש הטוב ביותר.

השלבים הבאים מתארים את תהליך העבודה המרכזי לפתרון בעיית השיבוץ האקדמי במערכת, משלב איסוף האילוצים ועד ליצירת מערכת שעות מלאה ומותאמת אישית לכל משתמש. מהלך זה יוצר תשתית חישובית מאורגנת, הנשענת על עקרונות הנדסה, תכנון מערכות ויישום אלגוריתמים מתקדמים, אשר יחד מאפשרים פתרון יעיל ומדויק לבעיה מורכבת זו.

שלב 1: הגדרת הבעיה וניתוח הדרישות - בשלב זה מתבצעת הבנה עמוקה של מכלול האילוצים המערכתיים: אילוצי מרצים (זמינות, מגבלות הוראה), אילוצי סטודנטים, מבנה קורסים ויחסי תלות אקדמיים (דרישות קדם, מעבדות, תרגולים), וכן מגבלות מערכתיות כגון שעות לימוד וכו'...

המטרה היא לתרגם מציאות אקדמית מורכבת לסט של אילוצים לוגיים ברורים, שמספקים בסיס לפתרון חישובי עקבי.

שלב 2: תרגום הבעיה למודל מבוסס גרף - לאחר איסוף הנתונים, הבעיה ממופה למודל גרפי המסדיר את יחסי התלות בין הקורסים. כל קורס מיוצג כקודקוד בגרף, וקשתות מייצגות יחסי קדם. בנייה זו מאפשרת לזהות בקלות מבנים היררכיים, למנוע יצירת מעגלים אינסופיים, ולבסס את תהליך השיבוץ על עקרונות מסודרים.

שלב 3: הפקת סדר לוגי באמצעות מיון טופולוגי - המערכת מפעילה מיון טופולוגי על הגרף כדי לקבל סדר עקבי שבו ניתן לשבץ את הקורסים. שלב זה מוודא היעדר סתירות בין תנאי הקדם, ומספק מסגרת התקדמות ברורה לתהליך השיבוץ. התוצאה היא סדר היררכי של קורסים שעליו מבוססים שלבי החיפוש הבאים - וכך האלגוריתם מתחיל מנקודת מוצא תקפה ויציבה.

שלב 4: בניית מודל חיפוש וגיבוש גבולות הפתרון - בהתבסס על הסדר הטופולוגי, נבנה מודל חיפוש המגדיר את מרחב האפשרויות: לאיזה סמסטר ניתן לשבץ כל קורס, כיצד מוצלב המידע עם זמינות מרצים, ומהם התנאים שמגדירים פתרון תקף. בשלב זה הבעיה ממוסגרת כך שניתן יהיה להפעיל עליה Backtracking בצורה יעילה ומבוקרת.

שלב 5: חיפוש פתרון באמצעות Backtracking מושכל - זהו השלב שבו מתבצע החיפוש בפועל. האלגוריתם מנסה לשבץ קורסים לפי המיון הטופולוגי, ובכל צומת בוחן את האפשרויות התקפות.

אם נתקל באילוץ שאינו מאפשר התקדמות (כגון התנגשות בזמנים, מרצה לא זמין, או חריגה ממבנה אקדמי), האלגוריתם חוזר לאחור (Backtracking) ובוחן חלופה אחרת. תהליך זה מאפשר חיפוש מדויק בתוך מרחב פתרונות גדול, תוך שמירה על עמידה מלאה בחוקים שהוגדרו.

שלב 6: צמצום המרחב על ידי Pruning - במקביל לחיפוש, המערכת מפעילה מנגנוני Pruning שמטרתם לעצור מסלולי חיפוש שלא יכולים להוביל לפתרון תקף. חיתוך זה מבוסס על זיהוי מוקדם של סתירות: אילוץי זמן שאינם ניתנים ליישוב, קורסים שאי אפשר למקם באף סמסטר מותר, עומסי זמן אצל מרצים, או רצפים שמפרים את ההיררכיה הטופולוגית. כך נחסך זמן חישוב יקר ונמנעת בחינה של מסלולים לא ריאליים מראש.

שלב 7: הפקת שיבוץ כללי ובקרת איכות - לאחר תהליך החיפוש, מופק לוח זמנים אקדמי מלא הכולל את כלל הקורסים, הסמסטרים, המרצים, השעות והחדרים. בשלב זה מתבצעת בקרת תקינות פנימית: בדיקת תקפות תנאי קדם, התאמת זמינות מרצים, מניעת התנגשויות, ושמירה על מבנה אקדמי זהה למדיניות המוסד. מנהל המערכת יכול לעיין בתוצאה, לבצע התאמות ידניות ולהגדיר גרסאות חלופיות במידת הצורך.

שלב 8: יצירת מערכות אישיות לסטודנטים - המערכת מאפשרת יצירת מערכת שעות מותאמת אישית לכל סטודנט, לפי אילוץיו והעדפותיו. בשלב זה מופעל תהליך התאמה חלקית המבוסס על תוצאות השיבוץ הכללי, תוך בדיקות עקביות נוספות. כך מתקבלת מערכת שעות אישית נטולת התנגשויות, המשמרת את ההיגיון האקדמי ואת מגבלות הלימוד שהוגדרו.

חשוב לזכור שהשילוב בין מיון טופולוגי לבין Backtracking עם Pruning נבחר כיוון שהוא מספק מענה מאוזן בין בנייה לוגית מוקדמת לבין חיפוש פתרון גמיש ומדויק.

המיון הטופולוגי מספק שלד היררכי קשיח המונע מראש סתירות ומאפשר התקדמות בטוחה.

ה - Backtracking מאפשר גמישות מלאה בהתאמת הפתרון לצרכים מורכבים ומשתנים.

ה - Pruning מצמצם את מרחב החיפוש ומונע בזבז זמן על מסלולים לא אפשריים.

יחד, תהליך זה יוצר מערכת שיבוץ אקדמית יציבה, עקבית, יעילה ומותאמת לכלל המשתמשים, גם בסביבה עתירת אילוצים ונתונים.

## פסאודו-קוד:

### מיון טופולוגי - Topological Sort:

Pseudocode – Topological Sort (using DFS)

Algorithm TopologicalSort(Graph G):

Input:  $G = (V, E)$  where  $V$  = set of courses,  $E$  = prerequisite relations

Output: Ordered list of courses (topological order)

```

visited ← empty set
stack ← empty stack
For each node v in V:
    If v not in visited:
        DFS(v)
Return stack in reverse order

Procedure DFS(node):
    Mark node as visited
    For each neighbor in node.adjacent:
        If neighbor not in visited:
            DFS(neighbor)
Push node onto stack

```

הסבר:

1. עובר על כל הקורסים בגרף אחד-אחד.
2. בודק אם כבר ביקר בקורס. אם לא - מתחיל עליו חיפוש עומק (DFS).
3. בתוך DFS הוא מסמן את הקורס כ"נעול" / ביקרתי בו.
4. עובר על כל הקורסים שתלויים בו (כל קורס שמגיע אחרי בתור).
5. אם יש קורס תלוי שעדיין לא ביקרו בו הוא נכנס אליו רקורסיבית וממשיך להעמיק.
6. רק אחרי שסיים לעבור על כל הקורסים שתלויים בו, הוא מוסיף את הקורס הנוכחי למחסנית.
7. בסוף, אחרי שכל ה-DFS מסתיים, הופכים את המחסנית, וזה נותן את סדר הקורסים החוקי (קודם תנאי הקדם, ואז הקורסים שתלויים בהם).

### :Backtracking and Pruning

Pseudocode – Course Scheduling with Backtracking and Pruning

Algorithm ScheduleCourses(courses, constraints):

```

bestSolution ← null
bestScore ← -∞
currentSchedule ← empty schedule

```

Sort courses by priority or number of constraints

Backtrack(0)

Procedure Backtrack(index):

If index == number of courses:

score ← Evaluate(currentSchedule)

If score > bestScore:

bestScore ← score

bestSolution ← copy(currentSchedule)

Return

course ← courses[index]

For each possibleSlot in AvailableSlots(course):

If IsValid(course, possibleSlot, constraints):

Assign(course, possibleSlot)

Backtrack(index + 1)

Unassign(course)

Else:

Continue // Pruning – skip invalid option

הסבר:

Backtracking - לקדמה

1. מכין מבנה נתונים ריק שבו ייבנה השיבוץ הנוכחי.

2. ממיין את הקורסים לפי עדיפויות (למשל: אילוצים, רמת חשיבות, כמות הגבלות).

3. מתחיל תהליך רקורסיבי של שיבוץ - קורס אחרי קורס.

מה קורה בתוך Backtracking :

1. בודק אם סיימנו לשבץ את כל הקורסים:

○ אם כן - מחשב ציון לשיבוץ.

○ אם הציון טוב יותר מהפתרון הכי טוב שנמצא עד עכשיו הוא שומר אותו כפתרון הטוב

ביותר.

2. לוקח את הקורס הבא בתור לפי האינדקס.
3. עובר על כל המשבצות האפשריות שהקורס יכול להיכנס אליהן (ימים, שעות, וכו').
4. לכל משבצת בודק אם אפשר לשבץ את הקורס שם לפי האילוצים:
  - אם המשבצת חוקית אז הוא משבץ את הקורס.
  - נכנס רקורסיבית לקורס הבא.
  - כשחוזרים מהרקורסיה - מסיר את השיבוץ כדי לנסות אפשרות אחרת.
5. אם המשבצת לא חוקית (התנגשות, אי זמינות, כפילות וכו') מדלג עליה מיד (Pruning) בלי לנסות לחפש פתרון דרכה.

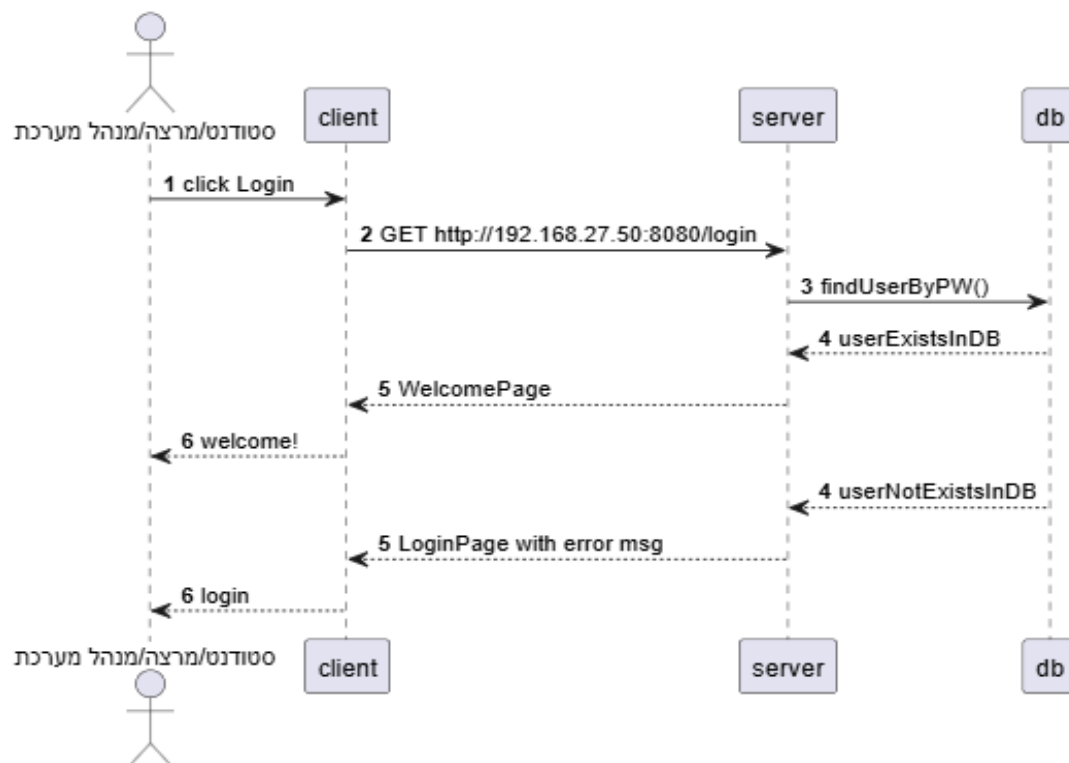
## 5. הליכים עיקריים בפתרון בעיה בטכנולוגיות הנדסה מתקדמות

### תרשימי Sequence:

להלן תרשימי Sequence שירחיבו מעט על הפעולות המרכזיות שהפרויקט עושה. פעולות הפרויקט

כתובות בתמצות בתרשימים ה - Use-cases שמוצג בסעיף 3.

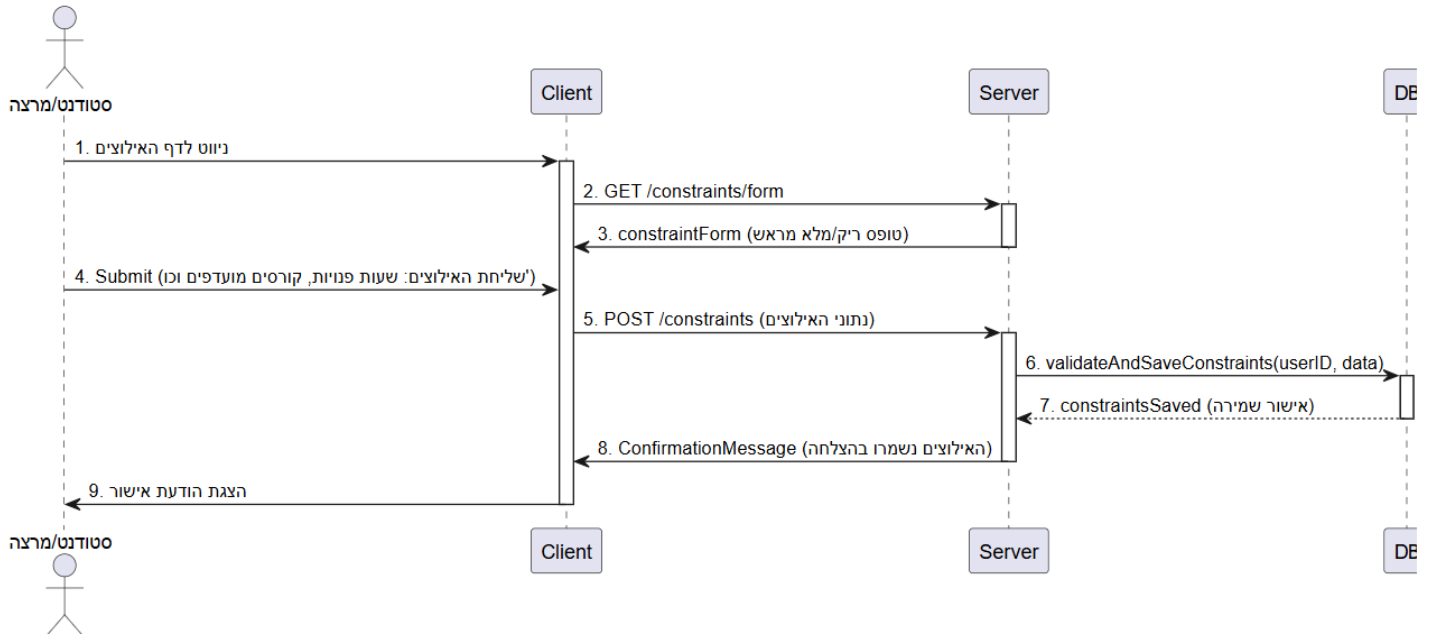
תרשימים ראשון - התחברות למערכת (Login):



בתרשים זה אפשר לראות הגדרת הכניסה למערכת (Login) המערב את כל המשתמשים הקיימים במערכת. בתרשים זה אפשר לראות שני תרחישים מרכזיים שהם הצלחה ומעבר עמוד (לאחר אימות), וכישלון שמקפיץ הודעה בהתאם.

### תרשים שני - הזנת אילוצים (Submit Constraints):

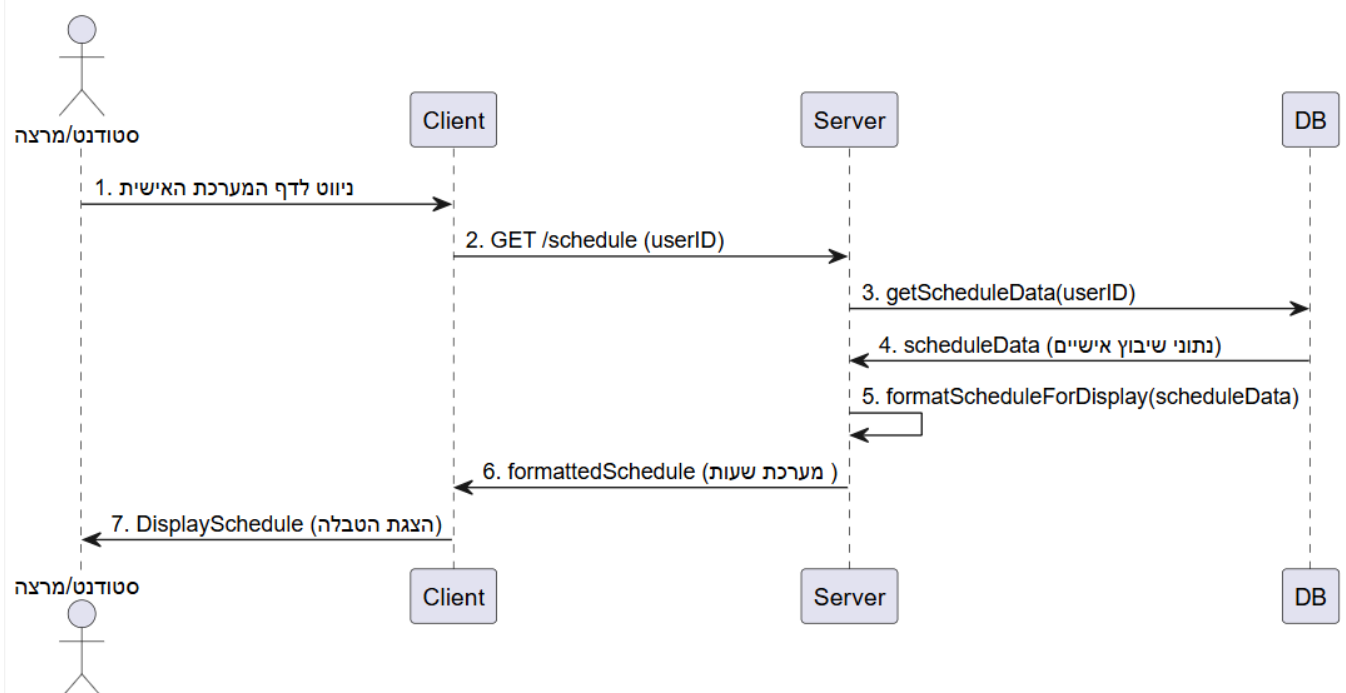
Sequence diagram - הזנת אילוצים (Submit Constraints)



בתרשים אפשר לראות שהאינטראקציה להזנת אילוצים מוגדרת עבור משתמשים שהם סטודנט/מרצה. התהליך כולל בקשה לטופס האילוצים, מילוי ולאחר שליחת הנתונים מהלקוח אל השרת, מבצע אימות ושמירה. התוצאה הסופית היא קבלת הודעת אישור על שמירה מוצלחת.

### תרשים שלישי - צפייה במערכת אישית (View Schedule)

Sequence diagram - צפייה במערכת אישית (View Schedule)

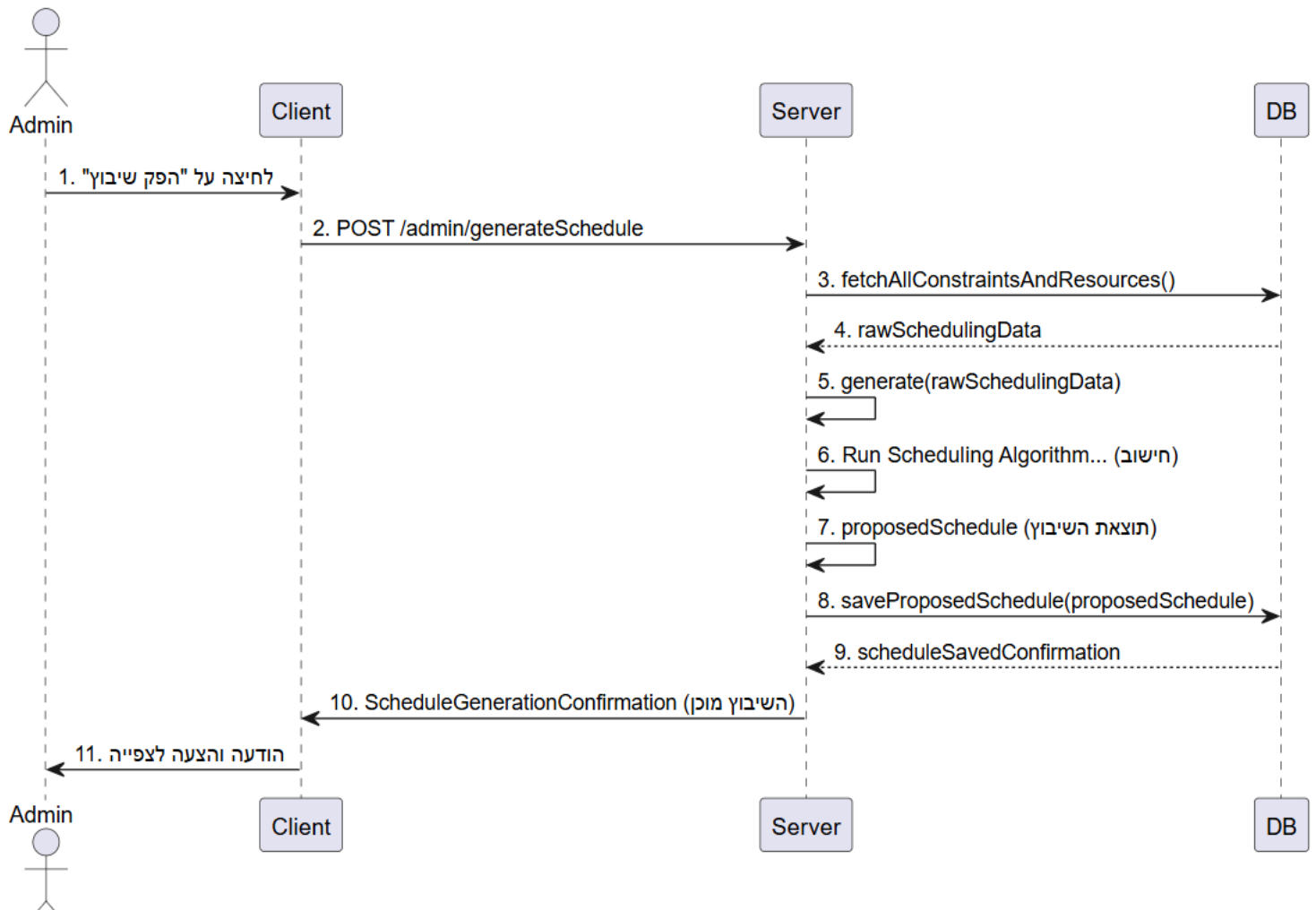




בתרשים אפשר לראות שתהליך הצגת מערכת השעות האישית מוגדר עבור סטודנטים ומרצים, ומתאר את השלבים להחזרה והצגה של נתונים מותאמים אישית.

תרשים רביעי - הפקת שיבוץ (Generate Schedule):

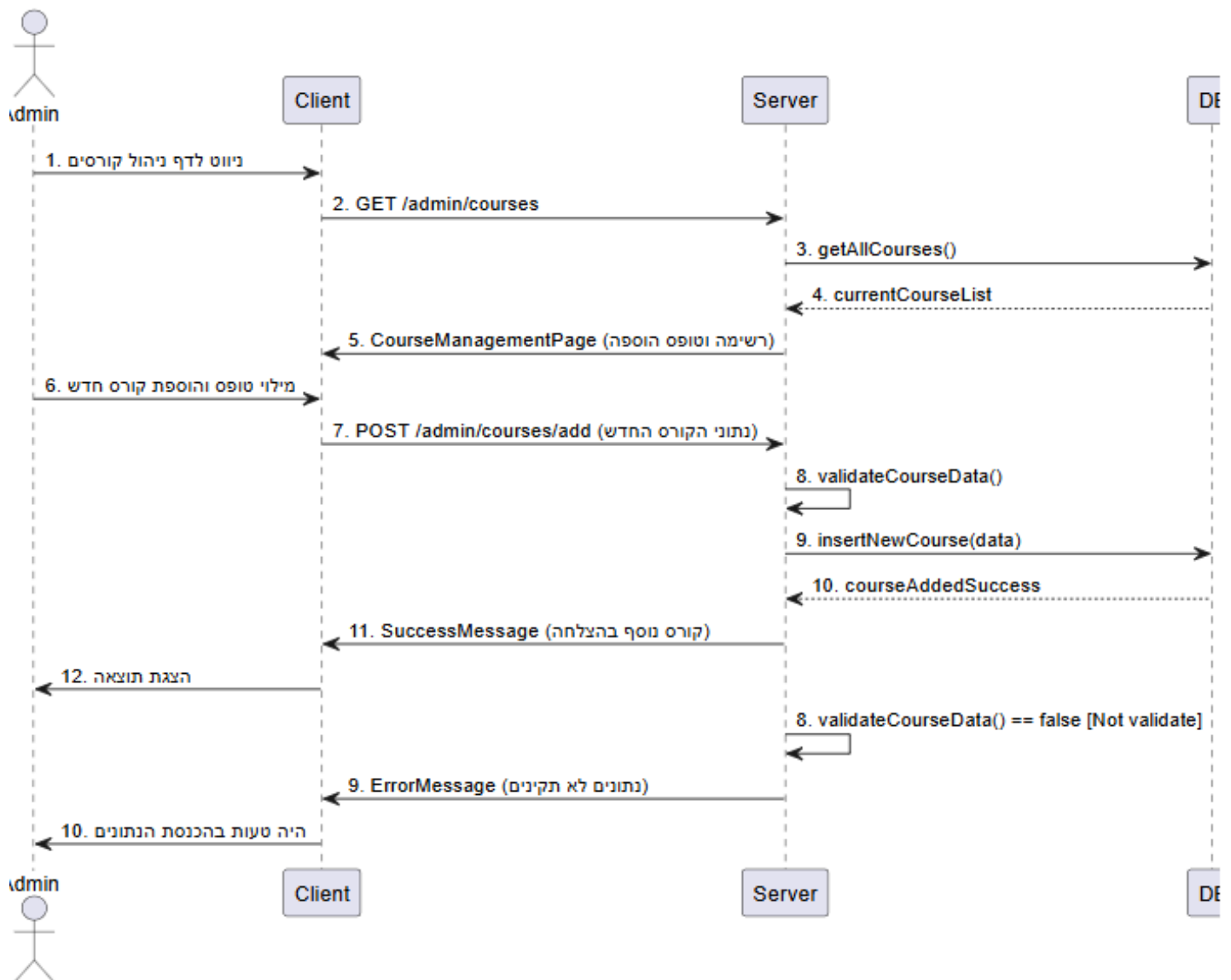
Sequence diagram - הפקת שיבוץ (Generate Schedule)



בתרשים אפשר לראות שתהליך הפקת השיבוץ מבוצע בלחיצת כפתור על ידי מנהל המערכת (Admin), ומציג באופן ישיר את כל החישובים שמתבצעים בתוך השרת. לאחר שהשרת אוסף את כל הנתונים הנדרשים (שלבים 3-4) הוא מבצע בעצמו את אלגוריתם השיבוץ המורכב (שלבים 5-7), ולאחר קבלת תוצאת השיבוץ המומלצת, הוא שומר אותה במסד הנתונים. לבסוף, המערכת מודיעה למנהל על השלמת התהליך וזמינות השיבוץ לצפייה.

תרשים חמישי - ניהול קורסים (Manage Resources):

Sequence diagram - ניהול קורסים



בתרשים אפשר לראות שתהליך ניהול הקורסים מוגדר לפעולת מנהל המערכת (Admin), הוא מציג את השלבים להוספת קורס חדש למערכת. התהליך הכולל בקשת רשימת קורסים הקיימים מה - DB, הצגת טופס הוספה למנהל ובעקבותיו שליחת נתוני הקורס החדש לשרת. לאחר קבלת הנתונים, השרת מבצע אימות למסד הנתונים ומחזיר למנהל הודעת הצלחה או הודעת שגיאה בהתאם לתוצאת העימות.

## ארכיטקטורה:

המערכת מבוססת על מודל **שרת-לקוח**, שהוא מודל עבודה נפוץ בפיתוח מערכות מחשוב מודרניות. במודל זה קיימת הפרדה ברורה בין שני גורמים עיקריים: הלקוח (Client) והשרת (Server). הלקוח הוא הרכיב דרכו המשתמש מתקשר עם המערכת, בדרך כלל באמצעות דפדפן או אפליקציה, ואחראי על הצגת המידע וקבלת קלט מהמשתמש. השרת, לעומת זאת, אחראי על ביצוע עיקר העבודה הלוגית של המערכת, עיבוד הנתונים, ניהול חוקי העסק והתקשורת עם מסד הנתונים.

התקשורת בין הלקוח לשרת מתבצעת באמצעות בקשות (Requests) ותגובות (Responses) לרוב בפרוטוקול HTTP. הלקוח שולח בקשה לשרת, השרת מעבד אותה בהתאם ללוגיקה ומחזיר תשובה מתאימה. מודל זה מאפשר חלוקת עומסים, אבטחת מידע טובה יותר, תחזוקה נוחה והרחבת המערכת בעתיד ללא פגיעה בחוויית המשתמש.

כדי להבטיח מערכת יציבה, מודולרית וקלה לתחזוקה, המערכת מפותחת על פי **ארכיטקטורת שלוש השכבות**, אשר מפרידה בין רכיבי המערכת העיקריים בהתאם לאחריותם. הפרדה זו מאפשרת פיתוח מסודר, תחזוקה קלה, בדיקות יעילות ושדרוגים עתידיים ללא תלות בין הרכיבים.

**1. שכבת התצוגה - frontend:** שכבת ממשק המשתמש אחראית על האופן שבו המשתמש רואה את המערכת ומתקשר איתה. זוהי השכבה המוצגת ישירות למשתמש, והיא כוללת מסכים, טפסים, כפתורים, טבלאות, גרפים והודעות מערכת. תפקידה המרכזי הוא לספק חוויית שימוש נוחה, ברורה ואינטואיטיבית, תוך הצגת מידע בצורה ויזואלית מובנת.

השכבה מנהלת את כל אינטראקציות הקלט והפלט: קליטת נתונים מהמשתמש, הצגת תוצאות ותגובות מהמערכת, וביצוע בדיקות בסיסיות כגון אימות קלט (לדוגמה: שדות חובה או פורמט תקין). עם זאת, שכבה זו אינה מבצעת לוגיקה מורכבת ואינה ניגשת ישירות למסד הנתונים. היא מתקשרת עם שכבת הלוגיקה באמצעות קריאות לשירותי השרת, וכך שומרת על הפרדה ברורה בין הממשק לפונקציונליות הפנימית של המערכת.

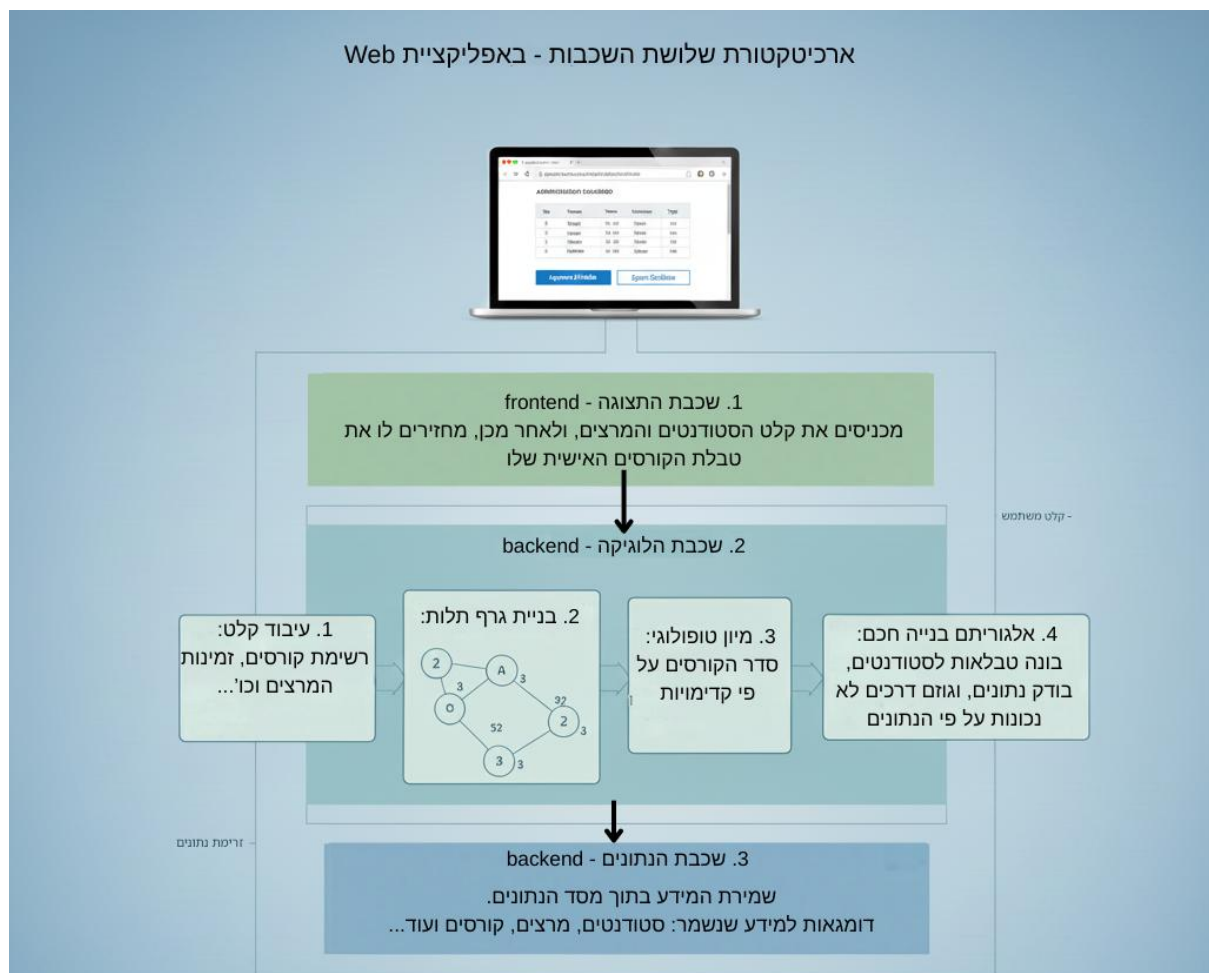
**2. שכבת הלוגיקה - backend:** שכבת הלוגיקה מהווה את ליבת המערכת וממומשת בצד השרת. שכבה זו אחראית על עיבוד הנתונים המתקבלים מהלקוח, יישום חוקי העסק, קבלת החלטות והפעלת התהליכים המרכזיים של המערכת. היא מגדירה כיצד המערכת אמורה להתנהג במצבים שונים, בהתאם לדרישות העסקיות.

שכבה זו משמשת כמתווך בין שכבת ממשק המשתמש לבין שכבת הנתונים. היא בודקת את תקפות הנתונים, מבצעת חישובים, מנהלת זרימות עבודה ומחליטה אילו פעולות יש לבצע על מסד הנתונים. בנוסף, שכבת הלוגיקה העסקית מטפלת בנושאים כגון הרשאות משתמשים, אבטחת מידע, טיפול בשגיאות וניהול תהליכים מורכבים. הפרדה זו מאפשרת שינוי של הלוגיקה העסקית מבלי להשפיע ישירות על הממשק או על מבנה הנתונים.

**3. שכבת הנתונים - backend:** שכבת הנתונים אחראית על ניהול ואחסון המידע של המערכת. תפקידה הוא לבצע פעולות של שליפה, הוספה, עדכון ומחיקה של נתונים (CRUD) מול מסד הנתונים, תוך שמירה על שלמות, עקביות ואבטחת המידע. שכבה זו אינה מכילה לוגיקה עסקית, אלא מתמקדת בגישה יעילה ומאובטחת לנתונים.

השכבה מספקת ממשק אחיד לגישה לנתונים עבור שכבת הלוגיקה העסקית, ומסתירה את פרטי המימוש של מסד הנתונים, כגון סוג המסד, מבנה האוספים או שאלות ספציפיות. גישה זו מאפשרת להחליף או לשדרג את מסד הנתונים בעתיד עם מינום שינויים בשאר חלקי המערכת, ותומכת בעבודה רציפה ובזמן אמת של המערכת כולה.

להלן תרשים המתאר את ארכיטקטורת שלושת השכבות:



בתרשים אפשר לראות שארכיטקטורת שלוש השכבות (Three-Tier Architecture) ממומשת באפליקציית Web, והיא מפרידה בבירור בין האחריות והתפקוד של כל שכבה במערכת.

השכבה הראשונה היא שכבת התצוגה, שכבה זו מיוצגת תחת הספרה 1 בתרשים והיא ממשק המשתמש. היא מיוצגת על ידי ממשק ה- Web בדפדפן, והיא מטפלת בקלט המשתמש (כגון הכנסת אילוצים) ומציגה את הנתונים, כמו טבלת הקורסים האישית.

השכבה השנייה היא שכבת הלוגיקה היא מיוצגת תחת הספרה 2 בתרשים. היא ממוקמת ב- backend, והיא לב המערכת. שכבה זו אחראית על עיבוד נתוני הקלט וכוללת לוגיקה מורכבת כמו בניית גרף תלויות, מיון לפי קדימויות ויישום אלגוריתם חכם לשיבוץ כדי לייצר את מערכת השעות הסופית.

השכבה השלישית והאחרונה שמופיעה תחת הספרה 3 היא שכבת הנתונים ממוקמת בתחתית, והיא אחראית אך ורק על אחסון ושליפת הנתונים (כגון נתוני סטודנטים, מרצים, קורסים) עבור שכבת הלוגיקה העסקית.

התרשים מדגים כיצד המידע זורם בין השכבות, כאשר כל שכבה מטפלת בתחום האחריות שלה בלבד, מה שמבטיח מודולריות וקלות תחזוקה.

## 6. הליכים עיקריים בתחום למידת מכונה - לא רלוונטי

הצעה זו אינה עוסקת בלמידת מכונה, לכן סעיף זה לא רלוונטי.

## 7. הליכים עיקריים במע' הפעלה/רשתות/תקשורת/אבטחה - לא רלוונטי

ההצעה הזו לא עוסקת במערכות הפעלה/רשתות/תקשורת/אבטחה. לכן סעיף זה לא רלוונטי. הערה חשובה: התייחסות למודל Client-Server מופיע בהרחבה בסעיף 5 של ההצעה.

## 8. תיאור פרוטוקולי תקשורת

המערכת מקיימת תקשורת רציפה בין הרכיבים השונים. כדי להבטיח אמינות, יעילות ואבטחה, המערכת משתמשת בכמה פרוטוקולי תקשורת סטנדרטים:

**HTTP** - זהו פרוטוקול שכבת היישום המשמש להעברת "היפר-טקסט" ונתונים בין הלקוח לשרת. HTTP (HyperText Transfer Protocol) משמש כבסיס לתקשורת שרת-לקוח. המערכת תשתמש בפרוטוקול זה בשביל להעביר מידע, משכבת התצוגה אל שכבת הלוגיקה ובנוסף משכבת הלוגיקה אל שכבת הנתונים.

**TCP/IP** הוא צמד הפרוטוקולים הבסיסיים והמרכזיים של רשת האינטרנט. פרוטוקול TCP (Transmission Control Protocol) מבטיח שכל חבילות הנתונים שנשלחות בין המערכות יגיעו בשלמותן ובסדר הנכון ליעד, מה שהופך את התקשורת לאמינה.

לעומתו IP (Internet Protocol), אחראי על ניתוב חבילות הנתונים בין רשתות שונות, ומבטיח שהנתונים יגיעו מהמקור לכתובת היעד הנכונה.

**REST API** המערכת בנויה על שהיא גישה סטנדרטית המאפשרת תקשורת יעילה בין רכיבי הרשת. כל הנתונים המועברים במערכת, בין השרת ללקוח ודרך נקודות הקצה, מוצגים בפורמט JSON שיוסבר בסעיף הבא.

בנוסף, המערכת תשלב API חיצוני לצורך הוספת ערך מוסף וחויית משתמש משופרת. לדוגמה, באמצעות הטמעת API של תחזית מזג אוויר, המערכת תציג המלצות לבוש פרקטיות במסך הבית של המשתמש בהתאם לתחזית בימי הלימוד.

**JSON** הוא פורמט סטנדרטי להצגת נתונים, המבוסס על טקסט פשוט וקריא. הפורמט מאפשר לייצג מידע בצורה מסודרת באמצעות זוגות של מפתח וערך, ולכן הוא נוח הן לקריאה על ידי אדם והן לעיבוד על ידי תוכנות JSON. הוא נפוץ מאוד בשימוש במערכות תוכנה מודרניות, ובעיקר משמש להעברת נתונים בין שרת ללקוח במסגרת תקשורת בין יישומים.

## 9. פיתוחים עתידיים

- שילוב מערכת הודעות פנימית בזמן אמת (צ'אט שרת-לקוח).
- הצגת סטטיסטיקות מתקדמות על ציוני קורסים וביצועי סטודנטים - הסטטיסטיקות יוצגו למנהל המערכת בלבד.
- פיתוח אפליקציית מובייל משלימה עם גישה למערכת האישית של הסטודנט.

## 10. תיאור טכנולוגיות הנדסה

- שפת תכנות:
  - שפת התכנות המרכזית בה נעשה שימוש היא **Java 21**. זוהי שפת תכנות עילית, מונחית עצמים (Object Oriented Programming - OOP), הנחשבת לאחת השפות הנפוצות והיציבות ביותר בעולם. פיתוח התוכנה Java מאפשרת בניית מערכות מורכבות, מאובטחות וקלות לתחזוקה, ומתאימה במיוחד לפיתוח אפליקציות Full-Stack הכוללות גם צד שרת (Backend) וגם צד לקוח (Frontend). השפה תומכת בניהול זיכרון אוטומטי ועבודה בסביבות מרובות משתמשים.

- מסגרות עבודה לתשתיות תוכנה (Frameworks) - אשר מהוות תשתית מוכנה לפיתוח תוכנה. מסגרת עבודה היא אוסף של קוד, כלים וספריות שמספקים שלד ברור לאפליקציה, ובכך חוסכים זמן פיתוח ומונעים כתיבת קוד חוזר. ה - Framework מכתוב דרך עבודה מסודרת, כגון הפרדה לשכבות, חלוקה למודולים, וניהול נכון של בקשות, נתונים ולוגיקה.
  - בצד השרת נעשה שימוש ב- **Spring Boot 3.5.8**, מסגרת עבודה לפיתוח מערכות Backend ב - Spring Boot. שפת העילית, Java מאפשרת הקמה מהירה של שרת HTTP מבוסס Apache Tomcat, יצירת שירותים (Services) להפעלת הלוגיקה, עבודה עם מאגרי נתונים (Repositories) לביצוע פעולות CRUD. וכן בניית REST Controllers המטפלים בבקשות ותגובות HTTP. היתרון המרכזי של Spring Boot הוא הפשטות והאוטומציה שהיא מספקת, לצד גמישות והתאמה לפרויקטים בקנה מידה גדול.
  - בצד הלקוח נעשה שימוש ב - **Vaadin 24.9.5**, מסגרת עבודה המאפשרת פיתוח ממשקי משתמש עשירים ישירות ב - Java, ללא צורך בכתיבה ידנית של HTML, CSS או JavaScript.
  - Vaadin מספקת מגוון רחב של רכיבי UI מוכנים, כאשר הלוגיקה של הרכיבים רצה בצד השרת והשינויים מסונכרנים אוטומטית לדפדפן באמצעות WebSockets ומנגנון PUSH כך ניתן לפתח ממשק משתמש מתקדם, אחיד וקל לתחזוקה.
- סביבות פיתוח משולבות (IDEs):
  - לצורך כתיבת הקוד וניהול הפרויקט נעשה שימוש בסביבת פיתוח משולבת (IDE) מסוג **VS Code 1.107.1**. זוהי סביבת פיתוח מודרנית הכוללת עורך קוד מתקדם, קומפיילר, מנפה שגיאות, השלמות קוד אוטומטיות (IntelliSense) והצגת תיעוד API. בנוסף VS Code, מאפשרת ניהול פרויקטים באמצעות Maven ותומכת בהרחבות שונות, כגון MongoDB לגישה מהירה למסד הנתונים בענן ו - PlantUML ליצירת תרשימי Use Case, Sequence ו - Class Diagrams.
- ספריות עזר:
  - ספריות העזר שבפרויקט עדיין לא ידועות.
- מנגנונים נוספים:
  - מסד הנתונים מנוהל באמצעות **MongoDB Atlas**, שירות ענן המאפשר שליטה מלאה על הנתונים דרך הדפדפן. MongoDB Atlas מספק מנגנוני אבטחה מתקדמים, כגון ניהול משתמשים והרשאות באמצעות סיסמאות, והגבלת גישה ברמת הרשת באמצעות רשימות כתובות IP מורשות בלבד, ובכך מבטיח שמירה על שלמות ואבטחת המידע.

## 11. מסד נתונים

לצורך שמירת הנתונים, המערכת משתמשת במסד נתונים בענן מסוג **NoSQL**. בחירה זו נעשתה לאחר בחינת מספר חלופות, כאשר השיקול המרכזי היה גמישות, היכולת להתמודד עם מבנים דינמיים של שאלוני אישיות, לאחסן סוגי מידע משתנים, ולעדכן נתונים בזמן אמת בממשק ה- Web.

מסד נתונים בענן הוא מסד נתונים המאוחסן ומנוהל על גבי תשתיות מרוחקות של ספק שירות, ולא על שרת מקומי. עבודה בענן מאפשרת זמינות גבוהה, גישה מאובטחת מכל מקום, גיבוי אוטומטי, והתאמה קלה לעומסי שימוש משתנים, ללא צורך בתחזוקה פיזית של שרתים. פתרון זה מתאים במיוחד למערכות מודרניות הפועלות דרך האינטרנט ודורשות עבודה רציפה ובזמן אמת.

NoSQL הוא סוג של מסדי נתונים שאינם מבוססים על מבנה טבלאי קשיח כמו במסדי נתונים רציונליים (SQL). במקום זאת, הוא מאפשר אחסון נתונים באוספים גמישים כגון מסמכים, דבר המקל על עבודה עם מידע שאינו אחיד או משתנה לאורך זמן. גישה זו מתאימה במיוחד למערכות בהן מבנה הנתונים עשוי להתפתח, כמו שאלונים, טפסים דינמיים ונתוני משתמשים מגוונים.

בפרויקט זה נעשה שימוש ב- **MongoDB**, מסד נתונים NoSQL מבוסס אוספים. הוא מאפשר עבודה נוחה עם נתונים מורכבים ודינמיים, ביצועים טובים בעבודה בזמן אמת, ואינטגרציה פשוטה עם מערכות Web ושרתי Backend. שילוב זה מספק פתרון יעיל, גמיש ומדרגי לניהול נתוני המערכת.

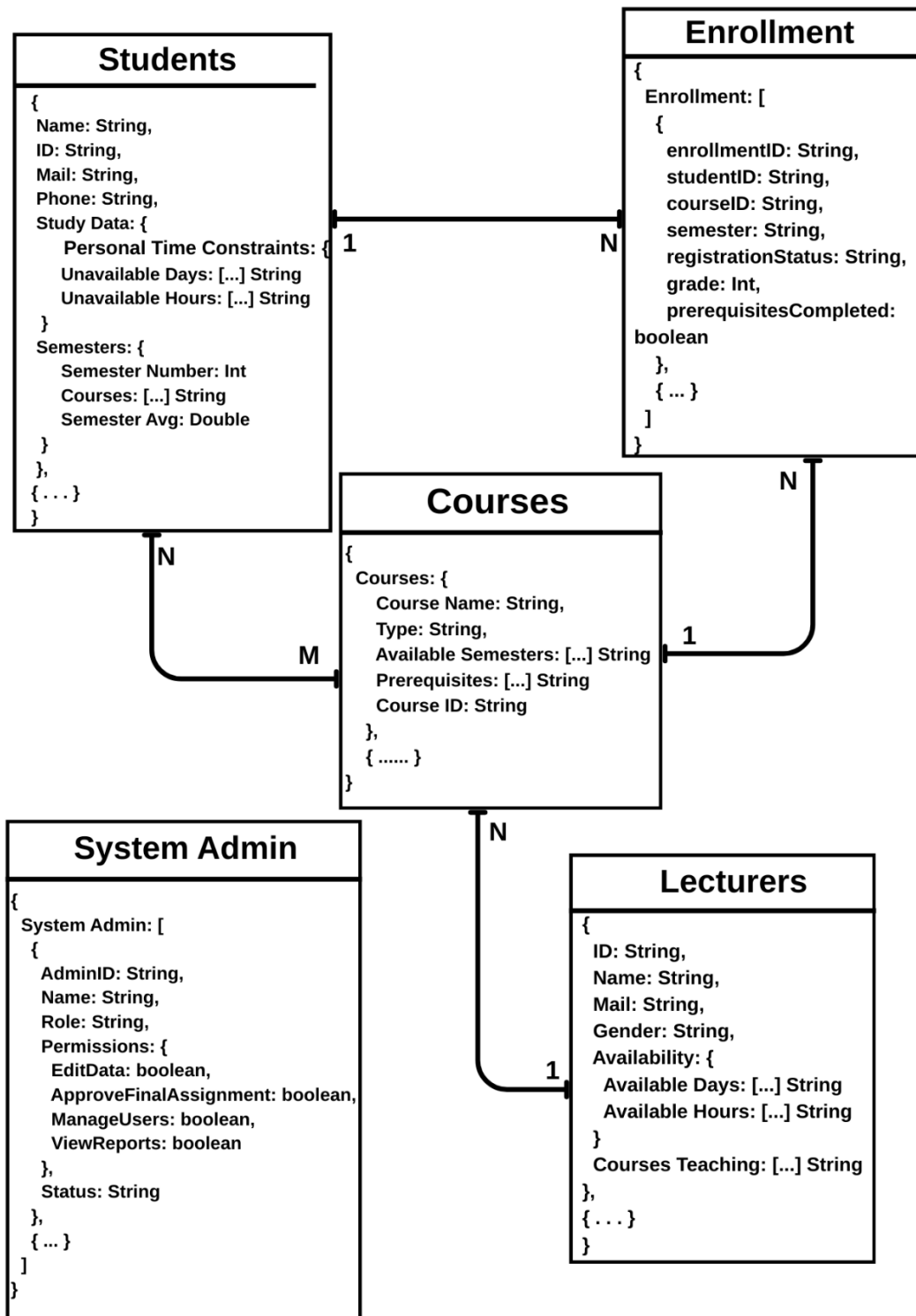
המידע שמאוחסן במסד הנתונים מכיל את האוספים הבאים:

- **סטודנטים Students** - מכיל את פרטי הסטודנטים, כמו שם, ת"ז, טלפון ומייל. אילוצי שעות אישיים, סמסטרים, קורסים וציונים. בנוסף, המערכת תשמור ממוצע סמסטריאלי, ונתונים נוספים.
- **קורסים Courses** - כוללת פרטים על כל קורס: שם הקורס, סוגו (חובה/בחירה), סמסטרים ודדלינאפשריים, ותנאי קדם.
- **מרצים Lecturers** - מכילה מידע על המרצים, זמינותם, הקורסים שהם מלמדים והעדפות הוראה. כמובן שגם כאן ישמר מידע על המרצים, כגון שם, מין, ת"ז וכו'...
- **הרשמות Enrollment** - קישור בין סטודנטים לקורסים בהם נרשמו, כולל סטטוס הרשמה, ציונים ומידע על השלמת דרישות קדם.
- **מנהלי מערכת System admin** - מכילה מידע על משתמשי מערכת בעלי הרשאות מיוחדות, כולל מנהל מערכת ואחרים שיכולים לערוך נתונים או לאשר את השיבוץ הסופי.

השימוש במסד הנתונים זה מאפשר למערכת לסנכרן את המידע בזמן אמת, לבצע אלגוריתמים מורכבים על אילוצים והעדפות, ולשמור על גמישות גם כאשר הנתונים משתנים בתדירות גבוהה.



להלן תרשים ERD המתאר את מסד הנתונים של המערכת:



בתרשים ERD זה אפשר לראות את מבנה מסד הנתונים של המערכת ואת יחסי הגומלין בין הישויות המרכזיות המרכיבות אותה. התרשים מציג באופן חזותי כיצד מקושרים הסטודנטים, המרצים והמנהלים אל הקורסים ואל נתוני ההרשמה, וממחיש את ארגון המידע הנדרש לצורך ביצוע תהליך השיבוץ והניהול התקין.

## 12. פרטים פורמליים

### לוחות זמנים:

שלבי עבודה	לסיים עד תאריך
בחירת פרויקט, חקירה ולמידה לעומק של נושאי הפרויקט	1.12.2025
כתיבה והגשת הצעת הפרויקט לאישור משרד החינוך	11.12.2025
מימוש הקוד של האלגוריתם המרכזי, ביצוע בדיקות ושיפורים	13.1.2026
בניית צד שרת	3.2.2026
בניית מסד הנתונים ושילובו	17.2.2026
בניית צד לקוח	3.3.2026
כתיבת ספר הפרויקט	24.3.2026
הגשת הפרויקט כולו (ספר + קוד) להגנה וקבלת ציון מגן	7.4.2026

חתימת מנחה הפרויקט:

כפיר  
יוסף  
יעקובוב

חתימת הסטודנט:

חתימת רכז המגמה: