

מטלת סיום קורס - פיתוח צד שרת

מטרת המטלה

מטרת שלב זה בפרויקט הגמר היא להקים את תשתית השרת, ליישם עבודה נכונה בארכיטקטורת שלוש שכבות (3-Tier Architecture), ולבצע תקשורת מלאה בין בסיס הנתונים, השרת והלקוח (Client). עליכם ליישם את הלוגיקה העסקית של הפרויקט הקבוצתי שלכם.

הנחיות כלליות

- המטלה מתבצעת **בקבוצות הפרויקט**.
- יש ליישם את הקוד בשפת #C.
- הגישה לנתונים תתבצע אך ורק באמצעות ADO.NET (אין להשתמש ב-Entity Framework).
- יש להקפיד על עקרונות כתיבת קוד נקי וחלוקה ברורה לשכבות.

חלק 1: מסד הנתונים

עליכם לתכנן ולהקים את מסד הנתונים עבור הפרויקט שלכם ב-SQL Server.

1. צרו את הטבלאות הרלוונטיות לישויות בפרויקט.
2. הקפידו על קשרי גומלין (Relationships) ומפתחות (PK, FK).
3. הקפידו על כתיבת פרוצדורות.

חלק 2: בניית שכבות השרת (BL & DAL)

יש לבנות את הלוגיקה של המערכת בחלוקה לשכבות:

- **שכבת DAL (Data Access Layer):** מחלקות האחראיות על ההתחברות ל-DB, הרצת שאילתות (Select, Insert, Update, Delete) באמצעות SqlConnection, SqlCommand ו-SqlDataReader.
- **שכבת BL (Business Logic):** מחלקות המכילות את הלוגיקה העסקית, בדיקות תקינות (Validations) ועיבוד נתונים לפני שליחתם ל-DB או חזרתם לקונטרולר. השכבה תקרא לפונקציות מה-DAL.

חלק 3: יצירת Web API

- יש ליצור **לפחות 4 קונטרולרים (Controllers) שונים** המייצגים 4 ישויות מרכזיות במערכת שלכם (לדוגמה: משתמשים, מוצרים, הזמנות, הודעות).
- כל קונטרולר צריך לחשוף נקודות קצה (End Points) המשתמשות בפעלי HTTP מתאימים (GET, POST, PUT, DELETE).
- הקונטרולר יפנה אך ורק לשכבת ה-BL לקבלת מידע.

חלק 4: ממשק משתמש

- צרו דפי HTML מתאימים ל-API שיצרתם בחלק הקודם.
- כתבו קוד JavaScript המשתמש בפקודת fetch כדי לפנות לכל אחד מ-4 הקונטרולרים שיצרתם. **זכרו לעבוד כפי שנלמד בכיתה:**
 - קובץ [index.js](#) ראשי אליו מייבאים את הפונקציות השונות. בקובץ זה יהיו אירועים.
 - קובץ js מתאים לכל ישות עם פונקציות רלוונטיות עבורה.
- הציגו את הנתונים המתקבלים מהשרת באופן נעים לעין ומעוצב באמצעות CSS.

חלק 5: העלאת ותצוגת קבצים

עליכם לאפשר העלאת קובץ (תמונה או מסמך) עבור אחת מהשכבות שבחרתם (לדוגמה: הוספת תמונה למשתמש או למוצר).

1. **שמירה:** הקובץ הפיזי יישמר בתיקייה ייעודית בשרת (למשל images בתוך wwwroot), ואילו בבסיס הנתונים (DB) יישמר רק נתיב הקובץ (Path/Filename) כשדה מסוג מחרוזת (NVARCHAR).
2. **API:** יש לממש פונקציה בקונטרולר המקבלת קובץ (IFormFile), שומרת אותו בשרת, ומעדכנת את ה-DB בנתיב החדש.
3. **Client:** בטופס ה-HTML, יש להוסיף שדה input type="file" ולהציג את התמונה / קובץ שהועלה לאחר שליפת הנתונים.

הוראות הגשה

1. יש להגיש את המטלה **בשבוע הבא**.
2. יש להעלות קובץ ZIP לתיבת ההגשה במודל המכיל את כל הקוד וכן קישור לגיטהאב.

מחווון הבדיקה

קטגוריה	משקל	קריטריונים לניקוד מלא (100%)	הפחתת ניקוד (דוגמאות)
הגנה על הפרויקט (בחינה בע"פ)	20%	שליטה מלאה בקוד, הבנת הזרימה, ומענה על שאלות תיאורטיות (כולל הסבר על יתרונות SP).	(20- נק') חוסר ידע על הקוד. (10- נק') חוסר הבנה למה השתמשו בפרוצדורות.
מסד נתונים ופרוצדורות (Stored Procedures)	15%	חובה: כל הפניות ל-DB מתבצעות דרך Stored Procedures בלבד. אין כתיבת שאילתות SQL ה-#C. בתוך קוד שימוש בפרמטרים לפרוצדורה.	(15- נק') כתיבת שאילתות SQL "רגילות" בקוד במקום SP. (5- נק') לוגיקה עסקית מורכבת בתוך ה-SP במקום ב-BL.
מימוש ADO.NET ב-#C	15%	הגדרת CommandType.StoredProcedure. שימוש ב-using וסגירת חיבורים. קריאת נתונים תקינה (Reader).	(10- נק') השארת חיבורים פתוחים. (5- נק') שגיאות תחביר בקריאה לפרוצדורה.
ארכיטקטורה (N-Tier)	15%	הפרדה ל-DAL, BL, API. הקונטרולר נקי מלוגיקה.	(10- נק') ערבוב שכבות.
היקף פונקציונליות	15%	4 קונטרולרים, מימוש מלא של CRUD.	(5- נק') חוסר בקונטרולרים.
טיפול בקבצים (File Upload)	10%	העלאת קובץ, שמירה בתיקייה, שמירת נתיב ב-DB.	(5- נק') שמירת קובץ ב-DB.
אינטגרציית צד לקוח (Client)	10%	קוד JS תקין (Fetch), תצוגה ויזואלית.	(5- נק') ממשק לא עובד.

בנק שאלות הכנה להגנה על הפרויקט

חלק א': ארכיטקטורה ומבנה (Architecture)

1. הסבירו את ארכיטקטורת N-Tier שבניתם. מה התפקיד של כל שכבה (DAL, BL, API)?
2. למה אנחנו לא כותבים את הקוד שניגש למסד הנתונים ישירות בתוך ה-Controller? מה הבעיה עם זה?
3. תארו את זרימת המידע (Data Flow) מרגע שלקוח לוחץ על כפתור באתר ועד שהמידע נשמר בטבלה. אילו מחלקות ופונקציות מופעלות בדרך?
4. אם נרצה מחר להחליף את מסד הנתונים מ-Oracle ל-SQL Server, באילו שכבות נצטרך לגעת ובאילו לא? מדוע?

חלק ב': מסד הנתונים ו-SQL Stored Procedures

1. מדוע השתמשנו ב-SQL Stored Procedures במקום לכתוב שאילתות SQL רגילות (inline SQL) בתוך הקוד? ציינו לפחות שני יתרונות.
2. איך הקוד ב-C# "יודע" שהוא מפעיל פרוצדורה ולא סתם פקודת טקסט? (רמז: CommandType).
3. היכן מוגדרת מחרוזת החיבור (Connection String)? האם זה בטוח לשמור אותה בקוד עצמו? איפה נהוג לשמור אותה בפרויקט אמיתי?
4. מהם היתרונות של שימוש בפרמטרים (SqlParameter) מול שרשור מחרוזות?

חלק ג': מימוש טכני (C# & ADO.NET)

1. מה המשמעות של הפקודה using סביב יצירת ה-SqlConnection? מה יקרה אם תהיה שגיאה (Exception) בתוך הבלוק ולא נשתמש ב-using?
2. מה ההבדל בין ExecuteReader, ExecuteNonQuery, ו-ExecuteScalar? מתי נשתמש בכל אחד מהם?
3. כיצד אתם מתמודדים עם שגיאות? אם ה-DB למטה, מה המשתמש יקבל? (האם התוכנה תקרוס או תחזיר הודעה מסודרת?)
4. הראו לי בקוד: כיצד מנעתם SQL Injection?

חלק ד': טיפול בקבצים (File Upload)

1. כשמשתמש מעלה תמונה, מה בדיוק נשמר בבסיס הנתונים ומה נשמר בתיקייה בשרת?
למה לא לשמור את הקובץ עצמו בתוך ה-DB?
2. כיצד צד הלקוח (Client) שולח את הקובץ? (הסבירו את השימוש ב-FormData).
3. איך דאגתם שלא יוצר מצב שבו שני משתמשים מעלים קובץ עם אותו שם (למשל image.jpg) והאחד דורס את השני?

חלק ה': שאלות "מה אם" (Scenarios)

(שאלות אלו בודקות הבנה עמוקה ויכולת דיבאגינג)

1. "אם אני מוחק עכשיו שורה בטבלה ב-SQL ידנית, האם האתר שלכם יקרוס כשננסה להציג אותה, או שידע להתמודד עם זה?"
2. "אם אני משנה את שם הפרוצדורה ב-SQL, מה עלי לשנות בקוד ה-C#?"
3. "הצביעו על שורת קוד אחת שאתם גאים בה במיוחד, או שהיה לכם קשה במיוחד לכתוב אותה, והסבירו למה."

טיפים להגנה מוצלחת

- **דעו את הקוד:** אל תגידו "את זה יוסי כתב אז אני לא יודע". כל חבר צוות אחראי על כל הפרויקט.
- **השתמשו במונחים מקצועיים:** במקום להגיד "הדבר הזה שמתחבר", אמרו "Connection Object". במקום "הקוד שמביא נתונים", אמרו "DAL Layer".
- **היו כנים:** אם אתם לא יודעים תשובה, עדיף לומר "אני לא בטוח כרגע, אבל אני משער ש..." מאשר לנסות להמציא תשובה לא הגיונית.

בהצלחה!!