תרגיל שני.

בתרגיל זה נרצה להפוך את קוד הקליינט הסטטי שלנו לדינמי, על ידי מימוש שרת אינטרנטי.
אנא קראו את כל ההנחיות וההוראות מראש.

חלק 0: איך לחשוב בMVC ובפרט בעזרת התשתית של ASP.NET MVC.

כדי לסייע לכם לתרגם את הדרישות שמתוארות בתרגיל שלנו לקוד שעליכם לכתוב, נתאר דוגמא **שונה** מהתרגיל, ונראה כיצד היינו מממשים אותה, ואז נמתח קווי דימיון בין הדוגמא הזאת לתרגיל שלנו:

נניח היינו בונים אתר חדשות.
אזי, כמובן שהייתה לנו מחלקת Article, למשל:

```
public class Article {

    public int Id { get; set; }

    public string Title { get; set; }

    public string Body { get; set; }

    public Category Category { get; set; }
}
```

נניח גם שיש לנו מחלקת Category, למשל:

```
public class Category {

    public int Id { get; set; }

    public string Name { get; set; }

    public List<Article> Articles { get; set; }
}
```

המחלקות הנ״ל מקיימות ביניהם קשר יחיד-לרבים, כי קטגוריה אחת יכולה להכיל הרבה כתבות, אך כתבה אחת יכולה להיות משוייכת רק לקטגוריה אחת.
(שימו לב, באתר אחר היו יכולים להגדיר את הקשר בתור רבים-לרבים, ואז כתבה אחת הייתה יכולה להיות משוייכת להרבה קטגוריות).
ולכן, כדי לבטא את הקשר יחיד-לרבים הנ״ל, באובייקט מסוג Category יש רשימה של אובייקטים מסוג Article, ואז, בהינתן אובייקט מסוג Category ניתן לראות את כל הכתבות ששייכות לקטגוריה הספציפית.
(שימו לב שגם מאובייקט מסוג Article ניתן לראות את הכתבה שאליה הוא משוייך דרך הproperty של Category).

נניח כי בנינו בhtml דפים סטטיים עבור:

- עמוד ראשי, המציג את כל הכתבות באתר
- עמוד כתבה, המציג כתבה ספציפית.
- עמוד יצירה של כתבה חדשה. (טופס)
- עמוד עדכון כתבה קיימת. (טופס)
- עמוד מחיקת כתבה קיימת.

כעת אנחנו רוצים ליצור שרת בMVC בעזרת ASP.NET.

מחלקות ה Article ו Category הנ״ל הן מחלקות השייכות למודל.

השלב הבא הוא ליצור קונטרולר עבור Article, למשל:

```
public class ArticlesController : Controller {

    public ActionResult Index() {
        // Do something
    }

    public ActionResult Create() {
        // Do something
    }

    [HttpPost]
    public IActionResult Create([Bind("Id,Title,Body")] Article article) {
        // Do something
    }

    // More code …
  }
```

בקונטרולר הזה יהיו מספר פעולות (actions):

- הצגת כל הכתבות (Index)
  - הפעולה הזאת הייתה מופעלת כאשר בדפדפן היו מזינים את הקישור: http://foo.com/Articles/Index
  - הפעולה הזאת הייתה נעזרת במודל כדי להביא מהdb את כל הכתבות הקיימות ולהמיר אותן לאובייקטים מסוג Article עבור כל כתבה שכזאת. ואז, הייתה נעזרת בview - שזה בעצם קוד ה html של העמוד הראשי, כדי להציג באופן ויזואלי את רשימה האובייקטים מסוג Article.

למשל, בקונטרולר לעיל ישנה בדיוק פעולת index שכזאת. אם נראה מימוש לדוגמא שלה:

```
public ActionResult Index() {
        List<Article> articles = _service.GetAllArticles();
        return View(articles);
```

```
}
```

בדוגמא לעיל, הפעולה Index מביאה מהמודל את כל הכתבות בעזרת "שכבת" ביניים שנקרא service layer. באופן הזה, הקונטרולר שלנו לא יודע מאיפה מגיעות הכתבות או איך - וזה כל המטרה. בכל רגע נתון נוכל להחליף את המימוש של הservice ולהביא את המידע ממקור אחר או בדרך אחרת - בלי שהקונטרולר יצטרך להשתנות.

את רשימת הכתבות (האובייקטים) הפעולה מתרגמת לhtml שהדפדפן יכול להבין בעזרת view מתאים. למשל, הקוד הנ"ל ישתמש ב view הבא אשר יהיה בקובץ שנקרא index.cshtml:

```
@model IEnumerable<Article>

<table class="table">
  <tbody>
    @foreach (var item in Model) {
      <tr>
        <td>
          @item.Title
        </td>
        <td>
          @item.Body
        </td>
      </tr>
    }
  </tbody>
</table>
```

קבצי cshtml הינם קבצים מיוחדים בכך שהם מאפשרים לשלב קוד html עם לוגיקה דינמית ב #C. עם זאת, נדגיש, שמה שמועבר לדפדפן הינו html בלבד, ללא קוד ה #C, שרץ בשרת בלבד.

השורה הראשונה בקובץ "אומרת" שהview הזה עתיד לקבל רשימה של אובייקטים מסוג Article (ואכן, כמו שראינו בקונטרולר, זה מה שקורה).

בהמשך הקוד, אנחנו מייצרים טבלה בצורה דינמית, כך שכל שורה בטבלה היא בעצם אובייקט מסוג Article מהרשימה. הקוד הזה מורץ בשרת, והפלט שלו נשלח לדפדפן.

- הצגת כתבה ספציפית (details)
  - הפעולה הזאת הייתה מופעלת כאשר בדפדפן היו מזינים את הקישור: http://foo.com/Articles/Details/3
  - הפעולה הזאת הייתה נעזרת במודל כדי להביא מהdb את הכתבה שהid שלה הוא 3 וליצור עבורה אובייקט Article מתאים, ואז, הייתה נעזרת בview - שזה בעצם קוד ה html של עמוד כתבה, כדי להציג באופן ויזואלי את האובייקט מסוג Article.
- יצירת כתבה (create)
  - הפעולה הזאת בעצם מתחלקת ל2 פעולות: אחת עבור GET ואחת עבור POST (כמו שמופיע בדוגמת הקונטרולר שרשומה למעלה)
  - בעת ביצוע GET לכתובת: http://foo.com/Articles/Create, הפעולה הייתה מחזירה את הview המתאים, כלומר, קוד ה html של טופס יצירת כתבה חדשה.

- בעת ביצוע POST לכתובת: http://foo.com/Articles/Create, הפעולה הייתה מקבלת את הפרטים של הכתבה החדשה, הייתה יוצרת אובייקט Article מתאים, ושומרת אותו לdb בעזרת המודל.
- עדכון כתבה (edit)
  - הפעולה הזאת בעצם מתחלקת ל2 פעולות: אחת עבור GET ואחת עבור POST
  - בעת ביצוע GET לכתובת: http://foo.com/Articles/Edit/3, הפעולה הייתה שולפת בעזרת המודל את הכתבה שהid שלה הוא 3 ומחזירה את הview המתאים, כלומר, קוד ה html של טופס עדכון כתבה קיימת, אשר מציג בטופס את הפרטים של הכתבה כפי שהם שמורים בdb.
  - בעת ביצוע POST לכתובת: http://foo.com/Articles/Edit/3, הפעולה הייתה מקבלת את הפרטים המעודכנים של הכתבה שהמזהה שלה הוא 3, והייתה מעדכנת את הכתבה בdb ע״י המודל.
- מחיקת כתבה (delete)
  - הפעולה הזאת בעצם מתחלקת ל2 פעולות: אחת עבור GET ואחת עבור POST
  - בעת ביצוע GET לכתובת: http://foo.com/Articles/Delete/3, הפעולה הייתה שולפת בעזרת המודל את הכתבה שהid שלה הוא 3 ומחזירה את הview המתאים, כלומר, קוד ה html שמציג את פרטי הכתבה הקיימת, כפי שהם שמורים בdb.
  - בעת ביצוע POST לכתובת: http://foo.com/Articles/Delete/3, הפעולה הייתה מוחקת את הכתבה שהמזהה שלה הוא 3 מהdb ע״י המודל.

אומנם, התמקדנו באובייקטים מסוג Article, אבל, כדי לטפל באובייקטים מסוג Category הקוד לא היה כל כך שונה.
ל Category היה קונטרולר נפרד משלו, אשר היה מטפל בפעולות שקשורות לקטגוריות באתר.
למשל:

- רוצים ליצור קטגוריה חדשה ⇐ create של הקונטרולר של Category
- רוצים להראות רשימה של כל הקטגוריות ⇐ index של הקונטרולר של Category
- רוצים לראות רשימה של כל הכתבות ששייכות לקטגוריה מסויימת ⇐ details של הקונטרולר של Category

וכך הלאה.

עכשיו, נחזור לתרגיל שלנו, ונוכל למצוא הרבה קווי דימיון, למשל:

- הרשמה (יצירה) של משתמש חדש ⇐ יצירה של כתבה חדשה. (create)
- הצגה של כל אנשי הקשר במסך הצ׳אטים ⇐ הצגת כל הכתבות בעמוד הראשי. (index)
- הצגה של התכתבות עם איש קשר ספציפי ⇐ הצגת פרטים של כתבה ספציפית. (details)

וכך הלאה.

<u>חלק ראשון: תכנון העצמים.</u>

בחלק זה, עליכם לחשוב על האפליקציה שבניתם בתרגיל הראשון, ולאפיין אלו עצמים תזדקקו להם, מה השדות שיהיו להם **<u>ואלו קשרים הם יקיימו ביניהם</u>**, למשל קשר רבים-ליחיד, רבים-לרבים וכו׳.

שימו לב, על האפליקציה לאפשר הרשמה של כמה משתמשים שירצו, וכל משתמש שמתחבר רואה את אנשי הקשר שלו בלבד ויכול לתקשר איתם.
בתרגיל זה אתם נדרשים לתמוך רק בשליחת הודעות טקסטואליות בין אנשי המשתמשים.
(ולא בתמונות, אודיו וכו׳)

<u>חלק שני: יצירת פרויקט ASP.NET.</u>

צרו פרויקט ASP.NET core mvc, וצרו במודל את המחלקות מהחלק הראשון.

- <u>מי **שלא** השתמש ב React בתרגיל הראשון:</u>
  עליכם להעביר את קוד הקליינט שכתבתם בתרגיל הראשון לפרויקט ה ASP.NET core MVC.
  בכדי לעשות זאת, עליכם לחשוב אלו קונטרולרים תצטרכו ליצור, ובפרט, אלו actions יהיו בכל קונטרולר.
  השתמשו בדוגמאת העזר שתוארה בחלק 0 וחישבו כיצד המסכים שבניתם בתרגיל 1 ימופו לקונטרולרים ולactions.

- <u>מי **שכן** השתמש ב React בתרגיל הראשון:</u>
  אתם אינכם צריכים להעביר את קוד הקליינט שלכם בתור views, כי קוד הReact שלכם ממשיך בתור הview.
  במקום זאת, עליכם לבנות עמודי דירוג לאפליקציה.
  בעמוד הראשי של הדירוג (בserver - לא בצד לקוח), יופיע הדירוג הממוצע הנוכחי כרגע ורשימת כל הדירוגים (כפי שיוגדרו מיד).
  בעמוד הוספת דירוג, כל מי שרוצה יוכל להזין דירוג לאפליקציה (1-5), טקסט מילולי (פידבק), ואת השם שלו. לאחר הזנת דירוג חדש, הדירוג יופיע ברשימת הדירוגים בעמוד הדירוג הראשי, ושם יוצג התאריך והשעה שבו הוזן הדירוג. (המשתמש לא מזין זאת).
  ניתן יהיה לבצע חיפוש ברשימת הדירוגים.
  ניתן יהיה ללחוץ על כל אחד מהדירוגים ברשימה ולראות אותו בעמוד בפני עצמו.
  ניתן יהיה ללחוץ על כל אחד מהדירוגים ברשימה ולערוך אותו.
  ניתן יהיה ללחוץ על כל אחד מהדירוגים ברשימה ולמחוק אותו.

  עליכם לממש את המודל/קונטרולר/view עבור העמודים הנ״ל.
  חובה לעשות זאת בASP.NET <u>ולא ניתן</u> להוסיף זאת לפרויקט ה React.
  יש להתאים את העיצוב הבסיסי של העמודים לעיצוב של שאר האתר.

חלק שלישי: יצירת Web service.

נרצה שהצ׳אט שתכתוב קבוצה אחת, יוכל לתקשר עם צ׳אט שכתבה קבוצה אחרת.

אם נחשוב על צ׳אטים שאנחנו משתמשים בהם ביום-יום, שמה יש רק שרת אחד - ואיתו מתקשרים כל המשתמשים כולם.
אבל, כדי שהמשתמשים יוכלו להשתמש בצ׳אט בעזרת אתר אינטרנט או באפליקציה לטלפון - השרת צריך לחשוף סט של מתודות, או במילים אחרות API - כדי שהקליינטים השונים (אתר/אפליקציה) יוכלו לתקשר מול השרת.

אצלנו המצב דומה, אבל מעט שונה. אצלנו, כל קבוצה בונה את השרת שלה בעצמה.
לכן, משתמש שמשתמש באפליקציה שפיתחה קבוצה אחת, יכול (לכאורה) לדבר רק עם המשתמשים שמשתמשים באותו שרת.
אבל, מכיוון שאנחנו נגדיר API אחיד, שכל השרתים של כל הקבוצות יממשו אותו, משתמש באפליקציה של קבוצה X יוכל לדבר עם כל משתמש באפליקציה של קבוצה Y.
לשם כך, נממש web service אשר יחשוף RESTful API.

על ה API לתמוך בפונק׳ הבאה על ידי תשובות בפורמט JSON.
(נאמר שהאתר יושב בדומיין http://foo.com/)

עבור הכתובת: http://foo.com/api/contacts
- פעולת ה Get תחזיר את כל אנשי הקשר של המשתמש הנוכחי
- פעולת ה Post תיצור איש קשר חדש עבור המשתמש הנוכחי

עבור הכתובת: http://foo.com/api/contacts/:id
- פעולת ה Get תחזיר את הפרטים של איש הקשר שהמזהה שלו הוא id:
- פעולת ה Put תעדכן את הפרטים של איש הקשר שהמזהה שלו הוא id:
- פעולת ה Delete תמחק את איש הקשר שהמזהה שלו הוא id:

עבור הכתובת: http://foo.com/api/contacts/:id/messages
- פעולת ה Get תחזיר את כל ההודעות שנשלחו או התקבלו עם איש הקשר של המשתמש הנוכחי
- פעולת ה Post תיצור הודעה חדשה בין איש הקשר והמשתמש הנוכחי

עבור הכתובת: http://foo.com/api/contacts/:id/messages/:id2
- פעולת ה Get תחזיר את ההודעה שהמזהה שלה הוא id2: של איש הקשר שהמזהה שלו הוא id:
- פעולת ה Put תעדכן את ההודעה שהמזהה שלה הוא id2: של איש הקשר שהמזהה שלו הוא id:
- פעולת ה Delete תמחק את ההודעה שהמזהה שלה הוא id2: של איש הקשר שהמזהה שלו הוא id:

עבור הכתובת: http://foo.com/api/invitations
- פעולת ה Post מכילה הזמנה לשיחה חדשה

עבור הכתובת: http://foo.com/api/transfer
- פעולת ה Post מכילה הודעה חדשה עבור אחד מהמשתמשים

כולם חייבים לממש את הAPI הנ״ל.

אפשר לממש את זה בעזרת קונטרולרים באותו פרויקט מהחלק הראשון, ואפשר בעזרת פרויקט ASP.NET Web-API.

חובה על כולם לממש את הAPI במדויק.

להלן דוגמאות קלט/פלט לעבודה מול ה API (הפעולות הורצו בסדר שבו הן מופיעות):

Request	Response
Request URL: https://localhost:7265/api/contacts/ **Request Method:** GET **Status Code:** 200	[{"id":"bob","name":"Bobby","server":"localhost:7265","last":"I know what you did last summer", "lastdate":"2022-04-24T08:00:03.5994326"}, {"id":"alice","name":"Alicia","server":"localhost:7266","last":"any last words?", "lastdate":"2022-04-24T08:01:03.5994326"}]
Request URL: https://localhost:7265/api/contacts/alice **Request Method:** GET **Status Code:** 200	{"id":"alice","name":"Alicia","server":"localhost:7266","last":"any last words?", "lastdate":"2022-04-24T08:01:03.5994326"}
Request URL: https://localhost:7265/api/contacts/1 **Request Method:** GET **Status Code:** 404	
Request URL: https://localhost:7265/api/contacts/ **Request Method:** POST **Status Code:** 201 **Request Payload** view source {id: "ch", name: "Charlie", server: "localhost:7266"} id: "ch" name: "Charlie" server: "localhost:7266"	
Request URL: https://localhost:7265/api/contacts/ **Request Method:** GET **Status Code:** 200	[{"id":"bob","name":"Bobby","server":"localhost:7265","last":"I know what you did last summer", "lastdate":"2022-04-24T08:00:03.5994326"}, {"id":"alice","name":"Alicia","server":"localhost:7266","last":"any last words?", "lastdate":"2022-04-24T08:01:03.5994326"} , {"id":"ch","name":"Charlie","server":"localhost:7266","last":null, "lastdate":null}]
Request URL: https://localhost:7265/api/contacts/ch **Request Method:** PUT **Status Code:** 204 **Request Payload** view source {name: "Charles", server: "localhost:7266"} name: "Charles" server: "localhost:7266"	
Request URL: https://localhost:7265/api/contacts/ **Request Method:** GET **Status Code:** 200	[{"id":"bob","name":"Bobby","server":"localhost:7265","last":"I know what you did last summer", "lastdate":"2022-04-24T08:00:03.5994326"}, {"id":"alice","name":"Alicia","server":"localhost:7266","last":"any last words?", "lastdate":"2022-04-24T08:01:03.5994326"} , {"id":"ch","name":"Charles","server":"localhost:7266","last":null, "lastdate":null}]
Request URL: https://localhost:7265/api/contacts/bob **Request Method:** DELETE **Status Code:** 204	
Request URL: https://localhost:7265/api/contacts/ **Request Method:** GET **Status Code:** 200	[{"id":"alice","name":"Alicia","server":"localhost:7266","last":"any last words?", "lastdate":"2022-04-24T08:01:03.5994326"} , {"id":"ch","name":"Charles","server":"localhost:7266","last":null, "lastdate":null}]

Request	Response
Request URL: `https://localhost:7265/api/transfer/` **Request Method:** `POST` **Status Code:** 201 **Request payload: {** **“from": "alice",** **“to": "ch“,** **“content": “hello"** **}**	
Request URL: `https://localhost:7266/api/invitations/` **Request Method:** `POST` **Status Code:** 201 **Request payload: {** **“from": "alice",** **“to": "ch“,** **“server": "localhost:7265"** **}**	

Request	Response
Request URL: `https://localhost:7265/api/contacts/alice/messages/` **Request Method:** `GET` **Status Code:** 200	[{"id":181,"content":“How are you?","created":"2022-04-24T19:46:09.7077994",“sent":false},{"id":183,"content":“Ok","created":"2022-04-24T19:46:46.08033",“sent":true}]
Request URL: `https://localhost:7265/api/contacts/alice/messages/181` **Request Method:** `GET` **Status Code:** 200	{"id":181,"content":“How are you?","created":"2022-04-24T19:46:09.7077994",“sent":false}
Request URL: `https://localhost:7265/api/contacts/alice/messages/181` **Request Method:** `DELETE` **Status Code:** 204	
Request URL: `https://localhost:7265/api/contacts/alice/messages/` **Request Method:** `POST` **Status Code:** 201 ▼ **Request Payload** view source ▼ `{content: "Hello"}` `content: "Hello"`	
Request URL: `https://localhost:7265/api/contacts/alice/messages/183` **Request Method:** `PUT` ▼ **Request Payload** view source ▼ `{content: "Hello"}` `content: "Hello"`	

- מי **שלא** השתמש ב React בתרגיל הראשון:
רוב המימוש שלכם היה בחלק הקודם. הAPI הנ״ל רק חושף **בנוסף** את המודלים שכבר יצרתם.

מי שרוצה, יכול גם להשתמש ב API הזה כדי לבצע תקשורת א-סינכרונית עם השרת שלו במקום תקשורת סינכרונית, אך זו אינה דרישה ולא חובה.
למשל:
דוגמא לתקשורת סינכרונית היא שכאשר משתמש התחבר בהצלחה הוא מועבר לעמוד שבו הוא רואה את כל אנשי הקשר שלו. בעת לחיצה על איש קשר העמוד כולו נטען מחדש, אך הפעם מוצגת גם ההתכתבות עם איש הקשר שסומן.
אם המשתמש שולח הודעה חדשה, העמוד כולו נטען מחדש (מתרפרש) ובעמוד שנטען מחדש מוצגת ההודעה החדשה כחלק מההתכתבות.
שיפור פשוט לזה היה על ידי שימוש בiframe עבור ההתכתבות, אשר רק הוא היה נטען מחדש בכל פעם שהיו לוחצים על איש קשר אחר.

דוגמא לתקשורת א-סינכרונית היא שכאשר משתמש התחבר בהצלחה הוא מועבר לעמוד שבו הוא רואה את כל אנשי הקשר שלו, **אך** בעת לחיצה על איש קשר העמוד **אינו** נטען מחדש, אלא מתבצעת פניה א-סינכרונית לשרת המביאה רק את ההתכתבות הספציפית עם איש הקשר שסומן.
אם המשתמש שולח הודעה חדשה, העמוד **אינו** נטען מחדש (מתרפרש), אלא נשלחת הנחיה א-סינכרונית לשרת אודות ההודעה החדשה.

- מי **שכן** השתמש ב React בתרגיל הראשון:
הAPI הנ״ל משמש את האפליקציה שבניתם גם לפעולות הבסיסיות של לקבל רשימת אנשי קשר, לקבל התכתבות, התחברות/הרשמה וכו׳.
ולכן, אתם יכולים להוסיף בהתאם לצרכים שלכם פונקציונאליות נוספת, אך חובה לתמוך בAPI כפי שמפורט לעיל בדיוק.

כולם חייבים להקפיד על ה API בדיוק כפי שתואר לעיל, מפני שבעזרתו הקליינטים של קבוצות שונות יוכלו לדבר.

נמחיש כיצד:

- משתמש רוצה להוסיף איש קשר חדש:
  - לוחץ על הכפתור ונפתח החלון המודלי
  - בחלון זה הוא מזין:
    - את הכינוי של איש הקשר (מה שהוא רוצה)
    - את שם המשתמש של איש הקשר (איש הקשר חייב להגיד לו את שם המשתמש המדויק שלו)
    - את הכתובת של השרת ששם רשום המשתמש
  - בעת לחיצה על הוספה, מתבצעות שתי פעולות:
    - נוסף איש קשר בשרת שאליו רשום המשתמש הנוכחי (זה שעושה את ההוספה), בהתאם לנתונים
    - **נוסף איש קשר בשרת ששם רשום המשתמש השני**, על ידי פנייה לשרת השני לפי הAPI והפעלת הinvitations.
      כלומר, תשלח בקשת POST לכתובת http://foo.com/api/invitations (כאשר foo.com זה השם של השרת השני).

- משתמש רוצה לשלוח הודעה לאיש קשר קיים:
  - מזין את ההודעה ולוחץ על הוספה
  - בעת לחיצה על הוספה, מתבצעות שתי פעולות:
    - נוספת הודעה להתכתבות בין המשתמש לאיש קשר בשרת שאליו רשום המשתמש הנוכחי
    - **נוספת הודעה להתכתבות בין המשתמש לאיש הקשר בשרת ששם רשום המשתמש השני**, על ידי פנייה לשרת השני לפי הAPI.
      כלומר, תשלח בקשת POST לכתובת http://foo.com/api/transfer.

<u>חלק רביעי (ואחרון): תקשורת בזמן אמת.</u>

נרצה שכאשר משתמש X שולח הודעה למשתמש Y כאשר שניהם מחוברים כרגע לאתר (כל אחד בדפדפן שלו), הוא יקבל את ההודעה מיידית.

ה API שהגדרנו בחלק הקודם מאפשר למשתמש X לשלוח את ההודעה לשרת של משתמש Y.

נרצה שהשרת של משתמש Y יוכל "לדחוף" את ההודעה באופן מיידי למשתמש Y, בלי שמשתמש Y יצטרך לבקש אותה.

יש לממש את הפונק' הנ"ל בעזרת SignalR.

הנחיות:

- חובה לממש service layer בקונטרולרים. עם זאת, הserviceים לא חייבים לעבוד מול הdb בתרגיל הזה. מי שרוצה - יכול לממש זאת כבר כעת (זה יידרש לתרגיל הבא, כאשר נבקש לעשות זאת בעזרת ORM שנקרא Entity Framework). מי שלא רוצה, יכול במקום פשוט לממש עבודה מול משתנים סטטיים. למשל, הservice שהשתמשנו בו בקונטרולר בתחילת התרגיל יכול למשל להראות ככה:

```
public interface IArticlesService {
   IEnumerable<Article> GetAllArticles();
   void SaveNewArticle(Article article);
   // …
}

public class ArticlesService : IArticlesService {
      private static List<Article> articles = new List<Article>();

      public IEnumerable<Article> GetAllArticles() {
         return articles;
      }

      public void SaveNewArticle(Article article) {
         articles.Add(article);
      }

      // …
}

public class ArticlesController : Controller {

      private readonly IArticlesService _service;

      public ArticlesController(IArticlesService service){
         _service = service;
      }

      public ActionResult Index() {
         List<Article> articles = _service.GetAllArticles()
                  return View(articles);
         }
      }
      // …
}
```

שימו לב שכדי שה Dependency Injection אל ה Constructor יעבוד, צריך להוסיף בProgram.cs את הservice.

- ה Web API שהוגדר בשלב 2 צריך להשתמש ב JSON Web Token או בקיצור JWT בשביל שנדע מי המשתמש הנוכחי המדובר - עבור Messages ו Contacts. עבור transfer ו invitations אין שימוש ב JWT.
- נא לא לאחסן (או לשלוח) מידע חשוב (כמו סיסמאות אמיתיות או טקסט חשוב) בשרת, כי התרגיל לא לוקח בחשבון דגשים רבים שיש לקחת בהקשר של ה security של האפליקציה.

<u>הנחיות הגשה:</u>

ההגשה היא עד יום שלישי ה17.5 (כולל היום הזה - כלומר עד חצות).
זיכרו שיש לכם ימי חסד.

מוזמנים לפתוח GIT REPO חדש (אפשר להמשיך עם המאגר הישן), ולהעלות לשם את כל הקוד שלכם, צד שרת + צד לקוח. אנא הוסיפו לקובץ הREADME את כל פרטי ההרצה.

ההגשה היא דרך המודל, אחד מחברי הקבוצה מגיש קובץ טקסט (txt.) עם שמות המגישים, תעודות הזהות וקישור למאגר (גם אם מדובר באותו המאגר מהתרגיל הראשון).

בהצלחה!