הנחיות ראש רשות התקשוב הממשלתי

שם ההנחיה: תפיסת ההפעלה - REST API בשדרת המידע	
פרק ראשי: העברות מידע והנגשת מאגרים	מספר הנחיה: 7.2.6
פרק משני: טכנולוגי	מס' גרסה: 1.0 בתוקף מ-01.04.2022

1. מטרת ההנחיה

1.1 רשות התקשוב הממשלתי מקדמת מדיניות כוללת לשיפור העברת המידע הממשלתי, מתוך תפיסה הרואה בנתונים כלי משמעותי לייעול עבודת הממשלה ושיפור השירות לאזרחים. על מנת למצות את הערך הגלום במידע ולאפשר שיתוף מהיר, פשוט ויעיל של נתונים, מגבשת הרשות מתודולוגיות וכלים ליישום העברת מידע, הן בפן העסקי והן בפן הטכנולוגי. נדבך מרכזי של שיפור העברת המידע הוא היכולת לייצר ממשקי שיתוף נתונים מתקדמים ומונחי-משתמשים (API AS A PRODUCT). במסגרת זו, ההנחיה כוללת את הרכיבים הבאים:

1.1 תפיסת ההפעלה של שדרת המידע כשירות

1.2 הנחיה מפורטת ומדריך טכני ליישום תקן Open API בשירותי REST, בהתאם למדיניות הממשלתית.

1.2 מטרת הנחיה זו היא להציג את תפיסת שדרת המידע הממשלתי כשירות ולספק הנחיה טכנית מפורטת אודות מימוש תקן REST עבור שירותי API המפורסמים בשדרת המידע הממשלתי.

1.3 הנחיה זו מהווה נוהל עבודה רוחבי, ועל כן תוכנה מחייב את קהל היעד.

2. קהל היעד

1.4 מנהלי היחידות מונחות רשות התקשוב במשרדי הממשלה ויחידות הסמך

1.5 מנהלי יישומים.

1.6 מנהלי טכנולוגיות (CTO).

1.7 מנתחי מערכות ומפתחים האמונים על שירותי API במשרדים

3. הגדרות ומושגים

1.8 ראה מילון מונחי התקשוב הממשלתי.

1.9 מפרסם המידע

הארגון שממנו נדרש המידע, הוא זה שיפתח תהליך החצנת המידע בהתאם לצורך עסקי לשיתוף או עדכון מידע. החצנת המידע תיעשה בהתאם למדיניות זו ולשיטות העבודה מול צרכן המידע.

1.10 צורך המידע

ארגון אשר מבקש את המידע, ישתמש במידע לטובת חישובים פנימיים שלו ולצרכיו. יעזור באפיון הדרישה מול מפרסם המידע, כך שידע כיצד לעבוד עם המידע המתקבל.

1.11 API – APPLICATION PROGRAMMING INTERFACE - ממשק תכנות יישומים

אוסף של פקודות, פונקציות ופרוצדורות מוכנות מראש, המאפשרות למתכנתים לעשות שימוש פשוט בהן, מבלי הצורך לכתוב אותן בעצמם. כך יוכלו להשתמש במידע של היישום אותו הם צורכים לטובת היישום שלהם.

1.12 REST – REPRESENTATIONAL STATE TRANSFER

ארכיטקטורה למימוש שירותי רשת המבוססים על "שרת-לקוח". לקוחות יוזמים בקשות לשרתים, השרתים מעבדים את הבקשות ומחזירים תגובות מתאימות. בשיטת עבודה זו, הלקוח יכול להיות בשני מצבים : "מצב מנוחה" – מצב בו הוא מתקשר עם המשתמש ולכן לא תופס משאבים מהשרת, ומצב "שינוי מצב" - בו השרת מעבד את הבקשה של הלקוח עד החזרת תגובה. התקשורת בין השרת ללקוח מתבצעת בעזרת מבנה הJSON.

1.13 SOAP – SIMPLE OBJECT ACCESS PROTOCOL

פרוטוקול תקשורת להעברות מידע מבוסס על מבנה הXML. פרוטוקול זה מאפשר הגדרת פונקציות וביצוע קריאה להן בין שרת אחד לאחר.

1.14 INTEROPERABILITY - תְּפָעוּלִיוּת בֵּינִית

היכולת של שתיים או יותר מערכות או רכיבים להחליף מידע ולעשות שימוש במידע שהוחלף

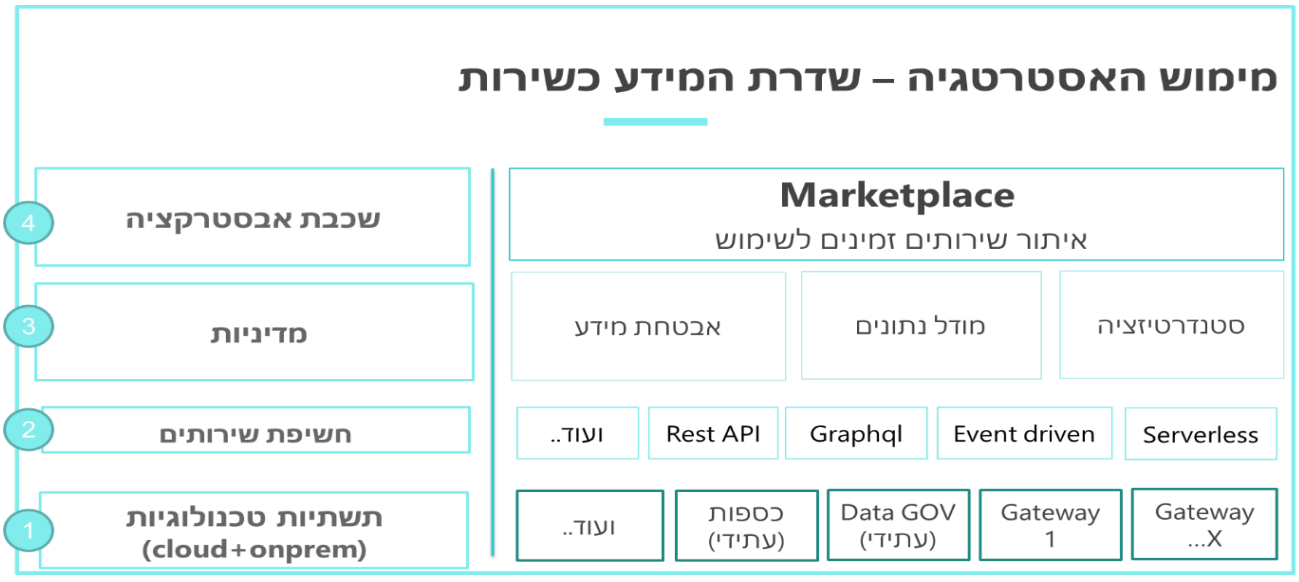
1.15 משרדי ממשלה מתאפיינים בריבוי מערכות מידע, עד כדי מאות מערכות שונות במשרד בודד. מערכות אלו פותחו לשרת צרכים תפעוליים, ולרוב אינן יודעות "לדבר" אחת עם השניה ישירות ולהעביר ביניהן נתונים באופן מובנה, קל וחומר בין משרדים שונים. לצורך יישום מוצלח של שילוב מערכות מרובות (אינטגרציה) באופן אחיד, יש לייצר שכבת ממשק המסדירה את הגישה למידע.

תפיסה זו נועדה לקדם שיתוף וזרימת מידע בין משרדי הממשלה, רשויות ציבוריות וגופים חיצוניים, לטובת שימוש דינמי בנתונים בזמן אמת ככלי לייעול עבודת הממשלה ושיפור השירות לאזרחים, תוך שמירה על בטחון מידע והגנת הפרטיות. בנוסף, שדרת המידע בתפיסה זו מהווה מסגרת עבודה להעברות מידע בממשלה מבוססת על פיתוח ממשקי מידע סטנדרטיים, אחידות ממשלתית של הישויות, המילונים, והמונחים, על בסיס מיפוי מידע וצרכים רוחבי, וניהול מרוכז שלהם על גבי תשתית אחודה.

חשוב לשים לב להבדל בין העברות מידע בין מערכות בתוך המשרד, לבין העברות מידע בין משרדים שונים. בעת החצנת, או העברת מידע בין מערכות בתוך המשרד, נדרש לחשוף את השירות ברכיבי התקשורת השונים במשרד בלבד. שדרת המידע נועדה לשמש כפתרון לשירותים אותם נדרש לחשוף לצרכנים מחוץ למשרד, החל ממשרדים מקבילים ועד אזרחים.

שדרת המידע הינה תפיסה שדרכה ניישם באמצעות טיפול על פי שכבות בהעברות המידע – החל מפתרונות טכנולוגיים ליישום התשתית ועד לממשק עבודה שבאמצעותה משתמש יכול לחפש, לאתר ולהשתמש בשירות שפורסם.

1.16 שדרת המידע משמשת שחקן מרכזי בכדי לממש את המעבר לטכנולוגיות הענן. השדרה מאפשרת פתרונות מבוססי ענן לחלוטין ופתרונות למשרדים אשר פועלים בשיטה היברידית, המשלבת החצנת שירותים בענן והחצנת שירותים הומתקנים בסביבת ה ON-PREM של המשרד.



1.17 שכבות הפתרון

1.3 תשתיות הטכנולוגית (1)

שכבת זו הינה התשתית הטכנולגית באמצעותה ממשים את יכולת העברת המידע בין גופים שונים. תשתית זו מיושמת באמצעות רכיבי API GATEWAY עבור שירותי WEB SERVICES ובעתיד יתאפשר חיבור טכנולוגיות נוספות כבסיס להעברת מידע בשדרת המידע. שדרת המידע תומכת בסוגי API GATEWAY שונים, בהתאם למדיניות שנקבעה.

1.4 גמישות טכנוליגות (2)

במסגרת תפיסה זו לא תהיה הגבלה על צורת מימוש. שירותים מסוגים שונים יוכלו להתחבר לשדרת המידע ולחשוף נקודות קצה (ENDPOINTS) בהתאם לצורך העסקי שיש לתת מענה. הפתרון יתמוך בפרסום שירותים מסוג REST API , שירות EVENT DRIVEN מיישם ארכיטוקורה של PUB/SUB (פירסום ורישום לשירות בתהליך א-סינכרוני) ו- SERVERLESS שיפותח בענן.

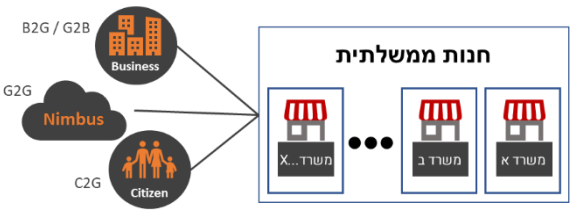
1.5 מדיניות (3)

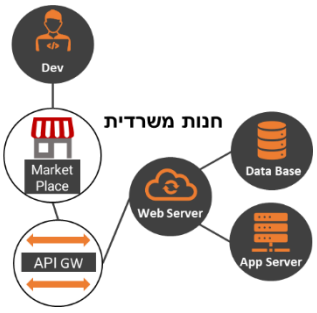
במטרה ליישם פתרון ממשלתי שיאפשר זרימה של מידע בין גופים יש לפעול על פי מדיניות אבטחת מידע, סטנדרטים לכתיבה ופרסום שירותים, ומודל נתונים ממשלתי שיאפשר שיח בשפה אחידה.

1.6 חנות השירותים Marketplace (4)

שכבה שתדע להתחבר לשכבות הבסיס ולחשוף למשתמש הקצה את כלל השירותים הזמינים לצריכה. למשתמש בשירות ייחשף מידע נוסף על השירות, כגון צורת הפניה, מי בעל השירות, מידע על הרשאות גישה, דוגמה פניה לשירות ועוד.

כמו בחלקים נוספים במסמך זה, נבדיל בין העבודה הפנים משרדית, לבין העבודה מול צרכנים חוץ משרדיים.





חנות השירותים בשדרת המידע הינה חנות ממשלתית, בה יפורסמו שירותים המיועדים להעברות מידע ללקחות חיצוניים למשרד, אם מדובר במשרדים מקבילים, או אזרחים וחברות חיצוניות. לעומת זאת, ניתן

יהיה להגדיר חנות משרדית, אשר בה המשרד מתנהל עצמאית ומציג את השירותים המשמשים את העברות המידע בין המערכות הפנימיות של המשרד.

מימוש שכבה זו תהיה באמצעות מוצר שיחצין את המידע מכלל המוצרים בשכבות הפתרון ולחשוף כלפי המשתמש תמונה אחת של כל השירותים הזמינים.

1.18 למה שדרת המידע כשירות?

1.7 מנקודת מבט משתמש הקצה - בתפיסה זו משתמש הקצה שמעוניין לצרוך שירות יראה את שכבת ה- Marketplace ובו יקבל מידע מלא על שירותים הקיימים וזמינים לשימוש וכן מידע פרטי על שירות מסוים שירצה לצרוך.

1.8 מנקודת מבט של מפרסם השירות יתקיימו תהליכי אוטומציה לפרסום שירות שיכיל בדיקות לנושא עמידה במדיניות לפני פרסום השירות. לדוגמא, בפיתוח שירות Rest Api תתבצע בדיקה שהשירות עומד בתקן Rest API לפני אישור הפרסום ב- Marketplace. דוגמא נוספת בנושא אבטחת מידע, שירות יוכל להתפרסם ב- API Gateway בתנאי שהוא מוגדר בהתאם למדיניות אבטחת מידע בנושא.

1.19 API GATEWAY בשדרת המידע

1.9 ה- GATEWAY הינו הרכיב הבסיסי ביותר בשדרת המידע, וזה שמאפשר תקשורת בין משרדי הממשלה השונים כרשת בתצורה של FULL MESH. משרדי הממשלה משתמשים בסוגים שונים של רכיבי GATEWAY, ועל כן שדרת המידע מאפשרת תמיכה בGATEWAYS שונים, כולל פתרונות בענן.

1.10 רכיב ה GateWay הוא רכיב לניהול מרוכז של ממשקי העברת נתונים מרובים בארגון - כמעין מרכזיה המנתבת כל פניה לשרות הנכון. בנוסף, מספק ה GateWay שכבת אבטחה וניהול הרשאות רוחבי, ומאפשר ניהול ובקרה מרכזיים על כל הממשקים.

1.11 שדרת המידע תאפשר עבודה עם סוגי GATEWAYS שונים, כגון: DATAPOWER,LAYER7 CA, GOOGLE GATEWAY, AWS GATEWAY וכו', בהתאם ליישום מדיניות אבטחת המידע.

1.12 תשתית זו אינה יוצרת מאגרי מידע חדשים או שומרת מידע המועבר באמצעותה, ואינה משנה את הרשאות הגישה למידע. התשתית שומרת לוג הכולל MetaData של הפניות לשירותים (מידע על שם השירות, כתובת השירות, תאריך פניה, וכו' – ללא שמירת המידע עצמו, הלוג נשמר לצורך תחקור, ניטור וסטטיסטיקה)

1.13 במקום להתממשק ישירות למערכות התפעוליות, צרכני המידע מתממשקים למקום אחד מוגדר ומנוהל, דבר שמקל על תהליך ההתממשקות. בנוסף, פיתוח ותחזוקת הממשקים ע"י המשרד הופכת לפשוטה וזולה יותר.

1.20 מתי נפרסם שירות בשדרת המידע?

1.14 בזמן פיתוח ופרסום שירותים צריך להתחשב במרכיבי תוכנה וחומרה שונים, בבסיסם נדרש לשאלה מיהם לקוחות השירות. בעת פרסום השירותים נהוג לחלק את איזור הפרסום ל-2:

• EAST-WEST – חיבור בין מערכות פנים משרדיות

איזור זה משמש בעבור חיבור פנימי בתוך המשרד, העברת מידע בין המערכות והשירותים הפנימיים ל המשרד.

• NORTH-SOUTH – חיבור בין שירותים של המשרד ללקוחות חיצוניים

איזור זה משמש לחיבוריות בין העולם "החיצון", במטרה לקבל ולהוציא מידע ממקורות חיצוניים למשרד.

1.15 שדרת המידע משמשת כפתרון לעיקרון מס' 2 – North-South, כך מתאפשר למשרד להחצין שירותים מול לקוחות חיצוניים למשרד, כמו משרדי ממשלה אחרים, חברות המחוברות לשדרה ועד אזרחים. כל זאת בעזרת כלי מרכזי אחד המאפשר את העבודה בקלות יתרה. אין להשתמש בשדרת המידע לחיבור בין מערכות פנים משרדיות – דבר אשר יגרור עלויות פיתוח ותשתיות נוספות.

1.21 שדרת המידע מאפשרת מספר ממשקי תקשורת שונים

1.16 Government To Governemt

רשת שדרת המידע המרכזית המבוססת על רשת התקשורת הממשלתית (Gov Net), לרשת זו מחוברים רכיבי ה- GateWays של משרדי הממשלה בתצורת Full Mesh ומאפשרת למשרדי הממשלה לפרסם ולצרוך שירותים ממשרדים המחוברים לרשת.

1.17 Regulated Government Services

סביבה מרכזית לחיבור גופים המקושרים לרשת שדרת המידע על בסיס חיבור IPVPN לרשת ממשל זמין. רשת זו מנוהלת על ידי ממשל זמין כאשר משרדי הממשלה אשר צורכים או מפרסמים שירותים ברשת זו פונים לרכיבי ה- APIGW's של סביבה זו לצורך קישור לגופים אלו (הגופים בשלב זה הינם בעיקר בנקים וחברות ביטוח).

1.18 Open Government Services

סביבה מרכזית לחיבור גופים לרשת שדרת המידע על בסיס קישור לרשת האינטרנט הציבורית מול ממשל זמין. רשת זו מנוהלת על ידי ממשל זמין כאשר משרדי הממשלה אשר צורכים או מפרסמים שירותים ברשת זו פונים לרכיבי ה- APIGW's של סביבה זו לצורך קישור לגופים החיצוניים המחוברים לרשת (גופים לאומיים – רשויות, קופות חולים, חברת חשמל, דואר ישאל. מערכות ענן, חברות ואזרחים).

1.22 תרחישי שימוש

1.19 הסבר להגדרת החיבורים בין לקוחות מסוגים שונים

- G – מסמל Government משרדי ממשלה ו/או יחידות ממשלתיות שונות אשר מחוברים לרשת הממשלתית Gov.Net (משרד המשפטים, ממשל זמין, משטרה, משרד האוצר – מרכב"ה, וכו').
- B – בהתאם למכרז מסמל Business, עסק / חברה.
- C – בהתאם למכרז מסמל Citizen, אזרח.

כל אחד מהישויות (C ,B ,G) יכולה להיות בצד הצורך, בצד המפרסם או גם צורך וגם מפרסם.

1.20 G2G

- משרדי ממשלה המפרסמים שירותים וצורכים שירותים האחד מהשני ברשת הממשלתית
- המשרדים יחשפו מידע שברשותם ובאישור ועדה משרדית לחשיפת המידע.
- כל משרד יכול לצרוך שירותים ו/או לפרסם שירותים.

1.21 G2B / B2G

- שירותים בין גורמי מידע מחוץ למשרדי הממשלה על גבי רשת ייעודית (Regulated Gov).
- חשיפת המידע בהתאם לחוקי הגנה על הפרטיות , אישור ועדה משרדית ודרישות הרגולטורים.
- הרכיב הצורך או מפרסם המידע מול המשרד הממשלתי הינו מערכת (מערכת בענן, ברשת הארגון וכו')

1.22 C2G

- תרחיש של אזרח אשר פונה לשירות בצורה ישירה לצורך קבלת מידע
- החצנת מידע אישי של האזרח מתוך שדרת המידע הממשלתית על גבי רשת ייעודית (Regulated Gov) או האינטרנט (Open Gov)
- בתרחישים אלו יהיה תהליך הזדהות ואימות של האזרח במערכות ממשל זמין.

- מימוש של Client הפונה לצריכת המידע – על ידי אפליקציה או דפדפן שפונה ישירות לשירות (ולא דרך מערכת מרכזית)
- במקרים רבים האזרח לא יפנה לשירות על ידי Client, אלה יפנה למערכת שתפנה בשמו לשירות – מבחינת השדרה התרחיש יהיה G2G/B2G, התרחישים הבאים מתארים תהליך זה

1.23 C2G2G - אזרח נכנס למערכת של משרד ממשלתי (אזור אישי לדוגמא) לצורך קבלת מידע > מערכת פונה למשרד ממשלתי לקבלת מידע.

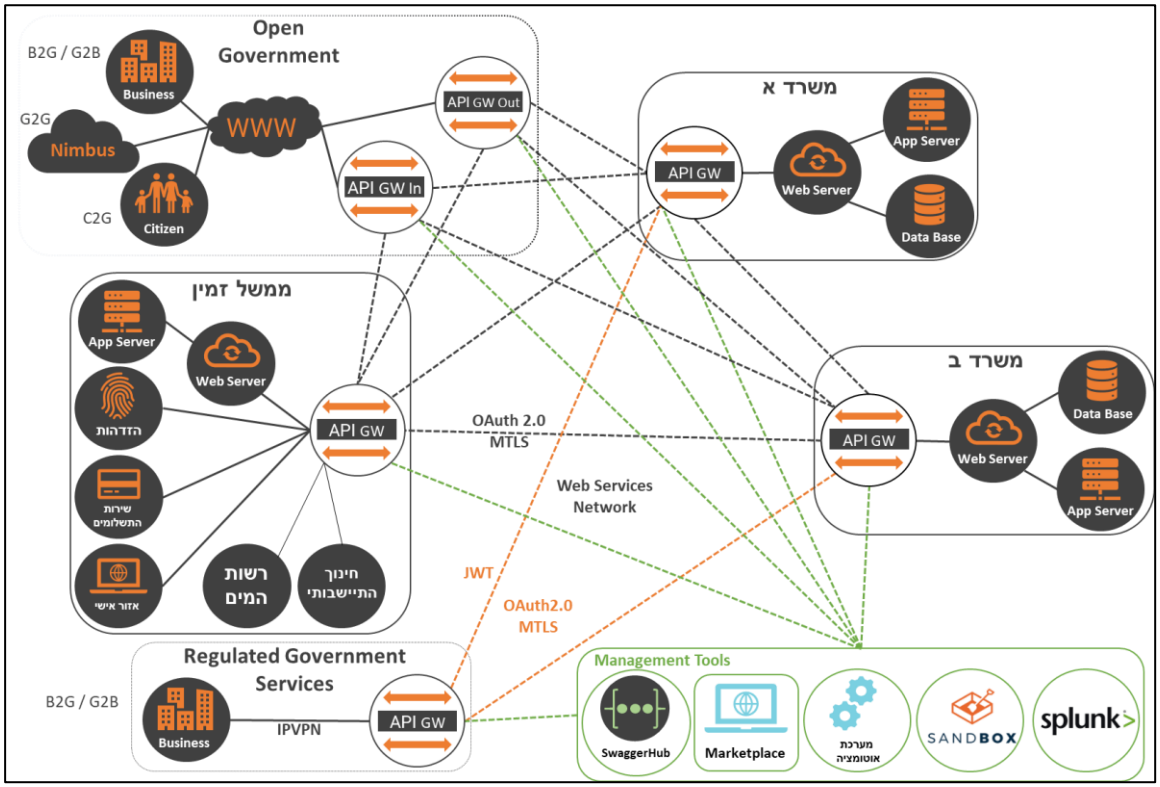
1.24 C2B2G – אזרח נכנס למערכת לצורך קבלת מידע (זימון תורים לדוגמא) > המערכת פונה למשרד משלתי לקבלת מידע

1.25 טבלה מסכמת הבדלים בין החיבורים השונים:

G2G	• Gov to Gov • חיבור משרדי הממשלה ברשת מאובטחת • ניהול עצמאי של השירותים המפורסמים • מאפשרת פרסום וצריכת שירותים בין המשרדים השונים • מבוססת על הרשת שממשלתית – Gov.NET
B2G	• Buisness to Gov - Regulated Government Services • מנוהל מרכזית על ידי ממשל זמין • רשת לפרסום וצריכת שירותים בין הממשלה לחברות תחת רגולציה ומחייבות חיבור ישיר למשרדי הממשלה • פרסום וצריכת שירותים בעלי רמת מידור גבוהה שאינה מאפשרת חשיפה לרשת האינטרנט • חיבור מאובטח מבוסס על IPVPN בחיבור למשרדי הממשלה • ממשל זמין יהווה את נקודת החיבור בין הרשת הממשלתית לבין הצרכנים החיצוניים • הרשת מאפשרת צריכה של שירותים על ידי הממשלה ומהממשלה (דו כיווני) • הרשת תאפשר פרסום שירותים מזוהים בהם יידרש הצרכן להיות מזוהה ומורשה מראש לצריכת השירותים
C2G	• Customers to Gov - Open Government Services • בסיס חיבור האינטרנט ציבורית לרשת ממשל זמין • קישור לגופים החיצוניים (רשויות, קופות חולים, חברת חשמל, דואר ישאל. מערכות ענן, חברות ואזרחים) • רשת לפרסום וצריכת שירותים בין הממשלה לצרכנים חיצוניים • ממשל זמין יהווה את נקודת החיבור בין הרשת הממשלתית לבין הצרכנים החיצוניים • משרדי הממשלה לא יהיו חשופים ישירות לרשת האינטרנט • הרשת מאפשרת צריכה של שירותים על ידי הממשלה ומהממשלה (דו כיווני) • הרשת תאפשר פרסום שירותים אנונימיים (נתונים כלליים כאשר אין חשיבות לצרכן) • הרשת תאפשר פרסום שירותים מזוהים בהם יידרש הצרכן להיות מזוהה ומורשה מראש לצריכת השירותים • תינתן עדיפות לשימוש ברשת זו – חיבור פשוט יותר, פחות תפעול מול הצרכנים, לא דורש תקציבים נוספים.

1.23 בתרשים הבא ניתן לראות החיבוריות של המרכיבים השונים בשדרת המידע.

- משרד א' , משרד ב'.... משרד X
- Open Government
- Regulated Services
- ממשל זמין
- ותוכנות ניהול



1.24 גמישות טכנולוגית

כפי שהוצג לעיל, שדרת המידע לא תגביל את צורות המימוש האפשריות לשירות. מטרת גמישות זו היא לאפשר המשכיות תמיכה בטכנולוגיות עתידיות. בדוגמא שהוצגה בחזון, ניתן לזהות מספר טכנולוגיות:

1.26 REST API

ארכיטקטורת עבודה המאפשרת עיצוב ופיתוח של שירותים המתבססים, לרוב, על פרוטוקול התקשורת **HTTP**, עליו מתבססת התקשורת ברשת האינטרנט. בשיטה זו לקוח מייצר בקשה לשרת. השרת מעבד את הבקשה ומייצר תשובה אשר נשלחת ללקוח.

1.27 GraphQL

שפת שאילתות זו פותחה ע"י פייסבוק והועבר לקרן פומבית ב2018. GraphQL מאפשר גישה לפיתוח ממשקי API בה הלקוח מגדיר את מבני הנתונים הנדרש עבורו, מבנה זה מוחזר ללקוח מהשרת בעת בקשה, דבר אשר מאפשר גמישות במבני הנתונים השונים ללקוחות השונים, אך מגביל את היכולת להשתמש בפתרונות מטמון שונים (כיוון שלא בהכרח יחזרו אותם הנתונים בכל פניה).

1.28 Event-Driven

ארכיטקטורת תוכנה בה מסתכלים על העברת מידע כסוג של אירוע

ספק האירוע (Provider) – היישות המייצרת את האירוע.

צרכן אירוע (**Consumer**) – היישות המקבלת את האירוע והמידע המצורף לה, על ישות זו לנתח את המידע ולהגיב בהתאם.

לכל אירוע ייתכנו מספר צרכנים אשר יבצעו את התהליכים הנדרשים בהתאם (ואף במקביל).

שיטת עבודה זו מאפשרת העברת ארועים בין מערכות שונות, בתוך הארגון ואף מחוץ לו.

1.29 Serverless

שיטת עבודה לפיתוח אפליקציות על בסיס Node.JS, Google Cloud Functions, AWS Lambda ועוד. מאחורי הרעיון עומד הצורך לפתח אפליקציות אשר יוחצנו כ Function as a service. אפליקציה בסיסית יכולה להכיל מספר קטן של פונקציות כאלו להשלים משימה. אך ניתן לשלב מספר רק של פונקציות כאלו במטרה לפתח צד-שרת מלא לאפליקציה.

בשלב ראשון נבחר לממש שירותי REST API – שיטת עבודה רווחת במשרדים כיום, ובכך נגדיר את דרך העבודה הנדרשת איתם.

5. תקן ממשלתי לפיתוח שירותי REST API

1.25 יש להיצמד להגדרות כפי שהן מוגדרות ב - OpenAPI 3 Specification.
חלק מההגדרות בתקן Rest API מוגבלים על פי הנחיות אבטחת מידע. לבירור בנושא זה יש לפנות לצוות שדרת המידע.

1.26 שמות משאבים ופורמט שמות

הגדרה	דוגמא לשימוש נכון	דוגמא לשימוש לא נכון
שמות המשאבים יהיו באנגלית ולא "בעברית-אנגלית" (כתיבה אנגלית של מילים בעברית).	companies/	havarot/
יש להשתמש בשמות עצם עבור משאבים, שם המשאב נדרש לשקף את תוכן המשאב. בנוסף שם של שירות לא אמור להצביע על סוג פעולת CRUD (Create, Read, Update, Delete) שמתרחשת בשירות.	users/{id}/	getUser/
שמות המשאבים יופיעו תמיד ברבים.	users/	user/
יש להשתמש במקף (-) בלבד להפרדת בין מילים. אין להשתמש ב underscores (pending_orders) או ב camel case (PendingOrders).	users/{id}/pending-orders/	users/{id}/Pending_Orders/

יש להשתמש באותיות קטנות בלבד (Lowercase) ב API URL (a ולא A) כיוון ש URIs הם case-sensitive, דבר זה ימנע בלבול בין אותיות קטנות וגדולות בפניות השונות לשירות.		
אין להוסיף slash (/) לסוף ה URIs. השאיפה היא ל URI נקי כמה שיותר.	users/{id}/pending-orders/	/users/{id}/pending-orders/
אין להשתמש בשמות קיצור למשאבים.	users/{id}/phone-number	users/{id}/tel-no/
אין לציין את סיומת הקובץ באמצעות שימוש בנקודה בעת גישה למשאב המחזיר קובץ.	/users/{id}/pending-orders	/users/{id}/pending-orders.json

1.27 פרסום שירות מסוג REST API בשדרת המידע

1.30 פיתוח מאובטח

משרדי הממשלה הינם יעד לתקיפות סייבר ע"י גורמים שונים בארץ ובעולם, על המפתחים לעמוד בכללי אבטחת מידע במהלך הפיתוח כדי להגן על המערכות שעליהם הם עובדים מן הקצה אל הקצה(E2E) .

תהליך פיתוח מאובטח נדרש לכסות כל תחנה שבה אנו עוברים בדרך, משליחת נתונים לשירות עד לקבלת המידע חזרה מהשרת מכיוון שבכל שלב בתהליך יתכנו נקודות תורפה במערכת שיכולים להוות פרצות אבטחה ולסכן את המידע והשרתים.

יש לבצע בדיקות תקינות על המידע שנשלח גם ב client שדרכו נצרך השירות, גם ב Server שמקבל את ערכים ל body, וגם ב API GW.

על המשרדים לאכוף אצלם בדיקות קלט הן ברמת האפליקציה עצמה על פי הגורם המנחה עבור המשרד והן ברמת ב API GW של המשרד.

עדכוני אבטחה בקוד חייב להתבצע למרות המשאבים הרבים הנדרשים לביצועם לצורך שילובם בכל חלקי המערכת.

קיימת הנחיית יה"ב 13.5 – "פיתוח מאובטח". ההנחיה נמצאת באתר יה"ב. אתר זה נגיש לממונה על הגנת הסייבר במשרד. יש ליישם הנחייה זו בשלבי הפיתוח.

1.31 התמודדות עם תווים מיוחדים בשדרת המידע

כדי למנוע מצב של פגיעה במידע או בשרתים דרך הקלדה של פקודות זדוניות או תווים מיוחדים מסוכנים בתיבת טקסט חופשי בשירות כל שהוא דרך המשתמש, ומצד שני לא לפגוע בחוויית המשתמש בשירות, הוגדר Policy שמקשיח את מה שנשלח לשרתים דרך patterns (תבניות) וחסימה של מילים או צירופים חשודים.

ה patterns מאפיינים תבניות מסוכנות שיכולות לסכן מידע או שרתים, שיחסמו לשליחה כבר אצל המשתמש כגון תבניות ל scripts או טקסטים מסוג DROP TABLE שיכול לגרום לאיבוד מידע/חשיפתו ופגיעה בשרתים.

שדרת המידע תומכת בשליחה של תווים מיוחדים בפנייה אל שירותים בעזרת השימוש ב patterns.

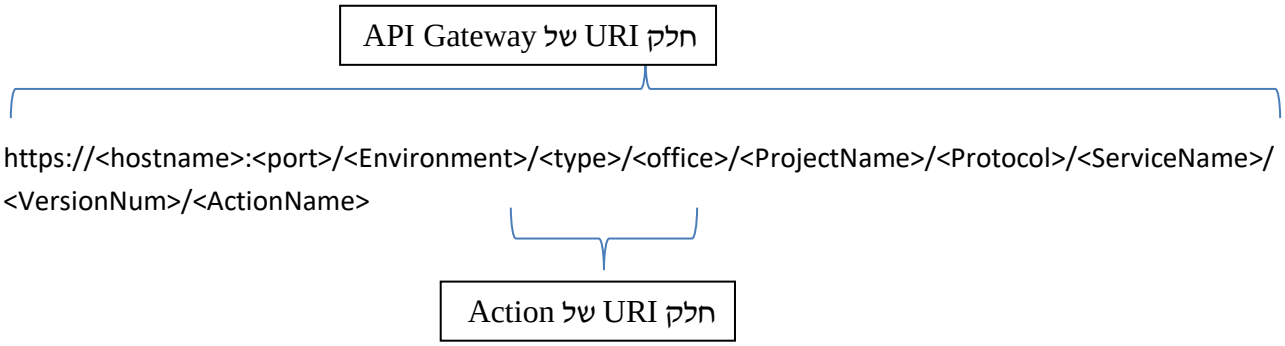
ניתן לראות את רשימת ה patterns אצל מנהל הפיתוח של המשרד.

במקרה שאין תווים מיוחדים בשירות, שדרת המידע לא תאכוף תווים מיוחדים עבור השירות לשיפור חווית משתמש, ביצועיים ומהירות.

1.32 העברת פרמטרים

Uri הוא ביטוי המזהה נתיב למשאב (Resource) פיזי או לוגי באופן חד חד ערכי, המאפשר גישה וביצוע פעולות על משאב ספציפי. ה- Uri מאפשר לפנות למשאב ללא כתיבה של פרמטרים ב Uri עצמו, וכך להקטין חשיפה של מידע רגיש בעת שליחת בקשה למידע בשירות.

1.33 מבנה של URI בשדרת המידע



חלק בURI	הסבר
hostname	שם השרת המארח
port	באיזה פורט יושב השירות
Environment	הסביבה שיושב בה השרת (test, dev, prepord, prod)
Type	סוג השירות consume/publish (אם סוג השירותpublish לא צריך לכתוב את ה type)
Office	המשרד שמפרסם את השירות
ProjectName	שם הפרוייקט
Protocol	SOAP או REST
ServiceName	שם השירות
VersionNum	גרסא של השירות (v1,v2)
ActionName	שם הפעולה בשירות

דוגמה : https://XXX.X.X.X:XXXX/test/Office/rest/companiesdata/v1/getcompanyinfo

1.28 שליחה מאובטחת ללא פרמטרים ב URI תוך שימוש ב POST

שליחה של מידע רגיש על האזרח חייבת להיות מאובטחת, ולכן לא נשתמש בפרמטרים ב URI בעת פניה לשירות

מידע רגיש כגון מספרי ת.ז, חשבונות בנק, מידע אישי, קומבינציות של תאריך לידה עם תעודת זהות, מספר כרטיס אשראי, מספר רישיון נהיגה, פרטי מידע על המערכת ישלחו תמיד במתודת POST ב Payloadשל המטודה כך שהמידע לא יוצג ב URI למניעה של דליפת המידע במערכות השונות כגון שמירת לוגים בתהליך צריכת שירות.

1.29 שליחת מידע לשירות

כתובת אינטרנט תקנית יכולה להיות מורכבת רק מאותיות, מספרים וכמה סימנים מיוחדים (כגון נקודתיים, לוכסן וכד') הממוקמים במקומות מתאימים בלבד.

בעת פניה לשירות ניתן להעביר מידע נוסף אל השירות באמצעות כתובת האינטרנט שלו (**URL**), לדוגמה כאשר משתמשים ב **QUERY STRING**.

HTTP://TARGETDOMAIN.COM/SAMPLE.ASPX?FIELD1=VALUE1&FIELD2=VALUE2

במקרה כזה יתכן ונצטרך להשתמש בסימנים מיוחדים שאינם תקניים ואשר עלולים לשבש את כתובת האינטרנט ולגרום לכך שהדפדפן לא "יבין" מהו המשאב שעליו להציג שכן לאותם תווים תפקידים מיוחדים בתוך כתובת תקנית. (כמו לדוגמה התו & שיש לו שימוש מסוים כמפריד בין ערכים בתוך **QUERY STRING** אבל יתכן שמשתמש ירצה לשלוח את ה תו הזה כחלק מ מחרוזת התווים).

אחת הדרכים להתגבר על הבעיה היא בעזרת קידוד הכתובת, **URL ENCODING**, החלפת כל תו אסור בתו המייצג אותו ואינו אסור. בצד השירות נדרש להחליף את הכתובת המקודדת חזרה טקסט המקורי כדי לעבוד איתו. לדוגמה: התו רווח מוחלף ב +.

פתרון זה אינו מומלץ למימוש כיוון שישנה סכנה למידע ולשרתים בשימוש בתווים מיוחדים גם כאשר הם מקודדים ב **URL**, בעיקר בעקבות תהליך ה**DECODING** של הכתובת אשר יתכן ויגרור שימוש זדונים בתווים המיוחדים.

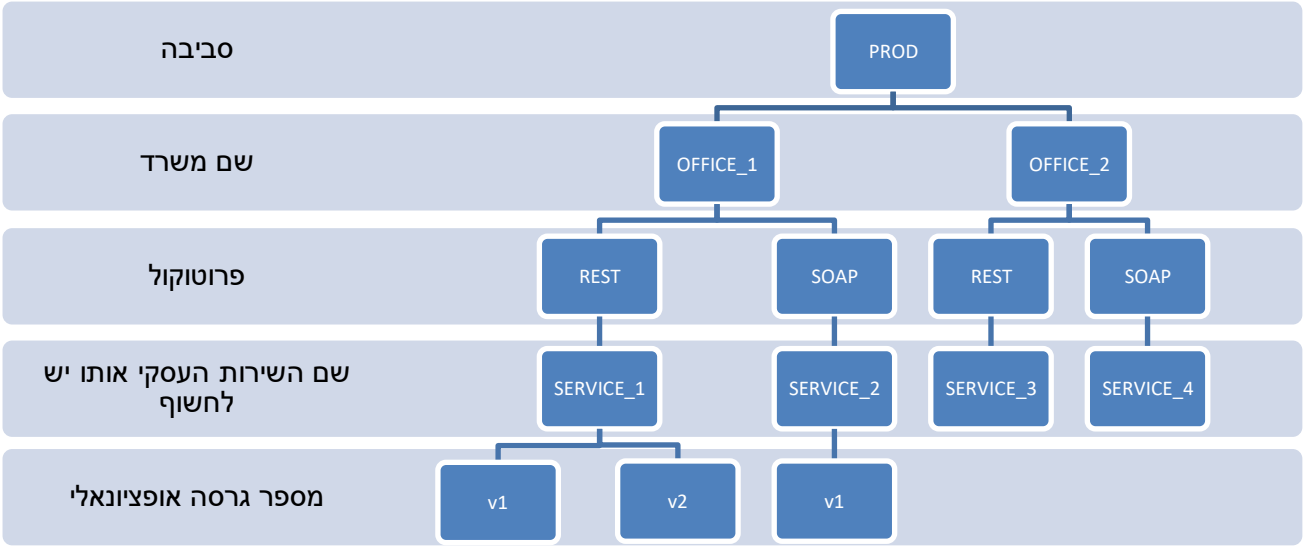
הדרך המומלצת להעברת מידע לשירותים השונים היא דרך **REQUEST BODY**, כלומר שליחת אובייקט **JSON** עם הבקשה ובו המידע הנדרש.

1.30 מבנה URI של OPEN GOV IN/OUT

- C2G OUT: ממשל זמין יחשוף למשרדים URI במבנה שהגדרנו לעיל, והמשרד יחשוף למשתמש קצה שירות consume באותה מבנה רק עם הערך consume-out בשדה ה type.
- C2G IN: ה-URI שייחשף לעולם יהיה זהה ל URI שהמשרד חשף אצלו (לפי המבנה שהוגדר לעיל כשירות publisher).

הערה: בעת צריכת שירותים ממשרדים דרך תשתית OpenGov, השגיאות שיחשפו החוצה יהיו מותאמות אישית (customized) עם קוד שגיאה 500 והודעה Internal Server Error.

להלן מבנה עץ הספריות הלוגי בשער הממשקים בו ימוקמו השירותים השונים בהתאמה:



1.31 חובת שימוש HTTP HEADER והקשחת הסכמה באמצעותו

HTTP headers מאפשר ל client ולserver להעביר מידע נוסף ביניהם ב response/request.

מידע זה יכול להיות הגדרות אבטחה שמועברות ב HTTP header המקשיחות את המידע שחוזר כ response ולהגן מפני התקפות ברשת.

יש להוסיף את ה headers הבאים :

X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY/SAME ORIGIN

למידע נוסף בהרחבה באתר של Open Web Application Security Project, קהילה פתוחה אשר מטרתה לקדם פיתוח מאובטח בעזרת מאמרים, כלים חינניים וכו׳: https://owasp.org/www-project-secure-headers

בנוסף יש להקשיח את תוכן המידע העובר בהודעות HTTPS באמצעות הוספת הכותר Content-Type שיכיל את התוכן הבא עבור JSON ו XML בהתאמה:

Application/JSON
Application/XML

יש ליישם את הקשחת סוג התוכן הן בבקשה מהמשתמש והן בתשובה החוזרת מהשרת.

אין להוסיף headers נוספים מעבר למה שצוין לעיל.

על פי מדיניות אבטחת מידע יוסרו headers שאינם מאושרים. המדינית עלפי ה Headersמוסרים מותקן ב gateway ולכן הביצוע יהיה בשלב הריצה (runtime). לבירור נושא זה ניתן לפנות לצוות שדרת המידע.

1.32 העברת קבצים

אם יש צורך בהעברת קבצים המשרד יפנה לצוות שדרת המידע בממשל זמין לקבלת מענה.

1.33 עיצוב טכני של שירות API ע״י SWAGGER

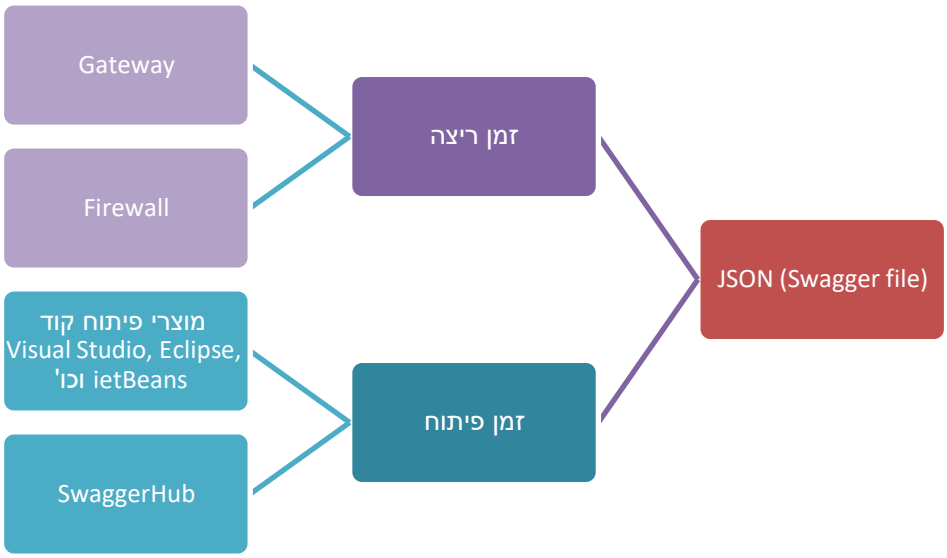
שירות מהווה נקודת תקשורת בין מפתח השירות לבין צרכן השירות. על מנת לייצר שיח פתוח וברור בין הצדדים, חשוב מאד לבנות חוזה טכני שמגדיר בצורה ברורה את הציפיות ואת אופן השימוש בשירות. ניתן להתייחס לחוזה הטכני כאל ״מדריך למשתמש״ שמפרט את כל תכולות השירות.

מסמך תיעוד זה - קובץ ה- Swagger הינו קובץ במבנה JSON הכולל את פרטי השירות והמבנה המפורט של השירות (Web Service) כולל ההגדרות כוללות path לשירות, פעולת HTTP של השירות (GET,POST,PUT,DELETE), הסבר על מטרת השירות, קלט/פלט, הישויות שהשירות מחזיר והמבנה שלהם, Schema עבור הפרמטרים – אורך, סוג השדה, האם חובה, תווים מותרים- pattern/regex וכו׳.

ע״י קובץ Swagger ניתן ליצור סטנדרטיזציה לשירותי REST, באמצעות יצירת שפה אחידה לתיאור השירותים.

ספקי מידע יכולים לתאר באופן אחיד את השירותים שהם חושפים, לתאר את שיטות ההזדהות לשירותים, להוסיף דוגמאות והסברים, העוזרים לצרכני המידע לגשת לשירותים ללא צורך בתקשורת ישירה עם הספק.

רכיבי אבטחת המידע לשירותים (Web Services) בהם רכיבי ה- APIGW מתבססים על קובץ ה- Swagger לביצוע האכיפה ובדיקת המידע העובר בשירות (כ- JSON Schema).



כפי שניתן להתרשם מהאיור, קובץ ה Swagger מאפשר לייצר סטנדרט שישמש גם בזמן פיתוח – מימוש הפונקציונאליות בכלי הפיתוח השונים, עבודה על גרסאות ואפיונים בעזרת מוצרים ייעודיים. אותו קובץ ישמש גם את רכיבי הרשת השונים בזמן ריצה, כל זאת בכדי לוודא כי אכן הפונקציונאליות והמידע שאמור לעבור עובר כפי שהוגדר מלכתחילה.

1.34 רישום ל LOG אפליקטיבי

כדי לייצר תיעוד של הפעולות בשרות לצורך תחזוקה, תיקון באגים ופיקוח יש לדאוג לכך שכל פעולה של שירות תכתב לקובץ לוג ייעודי באחריות מפתח השירותים. הפורמט יכלול לפחות את השדות הבאים:

Timestamp | src ip | service name/id | action | user/apikey/id | result

להלן מידע לדוגמא שניתן לשמור במאגר נתונים שמיועד ללוג:

- פרטי המערכת הפונה
- פרטי הפונה
- נתוני הפניה
- שעת הפניה
- מידע נוסף

שליחת נתונים בפורמט JSON

שליחת נתונים מול תשתית שדרת המידע תותר אך ורק בפורמט JSON.

1.35 HEALTH CHECK

על מנת ששדרת המידע תוכל לדגום את השירותים, על כותב השירות לממש מטודת בדיקה בכלל השירותים.

המתודה תקרא: health-check, ותחזיר {"status": "success"}.

Action Name: health-check

HTTP response status code: 200
Response payload: {"status": "success"}
HTTP Method: Get

1.36 הקשחת סכמה (REST Schema)

לכל שירות תוגדר סכמת מבנה נתונים שתכלול את ההגדרות לנתונים הנשלחים בפניה לשירות

- סוג הקלט
- אורך
- האם חובה
- תווים מותרים (regEx/Pattern)

1.37 RegularExpression /JsonSchema

Regular Expression היא תבנית טקסט המגדירה את התווים המותרים לשימוש בשדה מסוים ואת ארכו.

עבור כל שדה שנשלח לשירות מסוים יש להגדיר regEx/Pattern.

כך נבצע וולידציה על מידע שנשלח לשרת בעת בקשת שירותים.

את התבנית המותרת עבור השדות (regEx/Pattern) ומידע והקשחות נוספות (האם השדה חובה, סוג השדה, הסבר על השדה) נגדיר בקובץ ה - Swagger (בקובץ קונפיגורציה מסוג Json) שיהוה סכמה של הנתונים הנשלחים .

קובץ ה- Swagger ישמש לאכיפה ובצוע בדיקת נתונים ברכיבי ההגנה וה- API GW.

1.38 Error Handling (טיפול בשגיאות) והחזרת קודי שגיאה

כדי למנוע בלבול עבור משתמשי ה -**API** כאשר מתרחשת שגיאה, יש לטפל בשגיאות ולהחזיר קודי תגובות המתאימים לשגיאה שהתרחשה ולטפל בה.

על השירות להחזיר הודעות שגיאה גנריות אשר אינן מסגירות מידע אודות תשתיות או מבנה המערכת או על מהות השגיאה. על שירות להחזיר לפחות שגיאה אחת במקרה של קריסה.

1.34 טבלת קודי השגיאות הנפוצים של פרוטוקול HTTP

Bad Request 400	קלט צד לקוח נכשל באימות.
Unauthorized 401	המשתמש אינו מורשה לגשת למשאב, בדרך כלל חוזר שהמשתמש אינו מאומת
Forbidden 403	המשתמש מאומת אך אסור לו לגשת למשאב זה.

Not Found 404	משאב לא נמצא.
Internal Server Error 500	שגיאת שרת כללית.
Bad Gateway 502	תגובה לא חוקית משרת ה-Gateway.
Service Unavailable 503	מציין שמשהו לא צפוי קרה בצד השרת (כמו עומס יתר על השרת, חלקים מסוימים נכשלו ועוד).

1.35 שדרת המידע מטפלת בשגיאות באופן שונה בהתאם לרשת שבה השירות מפורסם:

- ברשת הממשלתית G2G השגיאה שמוחזרת מהשירות תעבור לצרכן השירות להמשך טיפול
- ברשת כלפי לקוחות חיצוניים ברשת B2G ו-G2C השגיאה נתפסת על ידי רכיב ה api gateway ומוחזר שגיאה במיסוך – שגיאת 500, שגיאת שרת כללית.

1.39 הגבלת גודל פניה (בדגש על גודל פניה בהעברת קבצים במקרה של BASE64)

הגדרת ברירת המחדל הינו 128 קילו בייט.

כך גם לגבי משתנה base64 שנשלח ב body של הפנייה.

ניתן להעביר קינון אובייקטים ב body עד רמה 6.

הגדרות אלו הינם הגדרות הבסיס. בהתאם לצורך ניתן לפנות לצוות שדרת המידע חהחרגות.

1.40 הרשאות ואימות

מנגנון הרשאות ואימות מול שדירת המידע הינו מבוסס על מנגנון MTLS לפי תקן OAUTH 2.0. הלקוח מקבל מפתחות גישה, באמצעותם הAPI-Gateway מאמת את הלקוח ואת ההרשאות לצריכת מידע.

כיוון שמנגנון ההרשאות והאימות ממומש באמצעות מוצר ה-API-Gateway אין צורך לממש אותו ברמת הקוד בעת פיתוח ממשקים חדשים אך יש לנהל את ההרשאות ע"י הגדרת Scope ברכיב API-Gateway.

1.41 פרוטוקול הצפנה הנתמך (TLS 1.2, 1.3)

השימוש בפרוטוקול SSL/TLS לאבטחה הוא חובה לצורך שמירת הפרטיות.

6. נספחים

1. נספח א - מדריך לתקן **REST** (**STANDARDIZATION**) ע"י **SWAGGER**
2. נספח ב' - מודלי נתונים- **SCHEMAS**

7. מסמכים קשורים

1.42 דיווח על אירועי סייבר ל SOC -הממשלתי, הנחייה 5.4

1.43 HYPERTEXT TRANSFER PROTOCOL (HTTP/1.1): SEMANTICS AND CONTENT

1.44 HYPERTEXT TRANSFER PROTOCOL (HTTP) STATUS CODE REGISTRY

1.45 עקרונות פיתוח מערכת להיערכות לענן

1.46 OPENAPI 3.0: HOW TO DESIGN AND DOCUMENT APIS WITH THE LATEST OPENAPI SPECIFICATION

8. גרסאות ההנחיה

מס'	סטאטוס	מהות שינוי	סעיפים שהושפעו	בתוקף מ-	נכתב ע"י	אושר ע"י
1.0	מאושר	גרסה ראשונה			קרן בר-לב רחל רן ההנחיה גובשה בשיתוף צוות שדרת המידע בממשל זמין	מוריה זיסוביץ
1.0	מאושר	ריענון		1.1.2026	קרן בר-לב	קרן בר-לב

9. ש להיצמד להגדרות כפי שהן מוגדרות ב - [OPENAPI 3 SPECIFICATION].

בסעיפים הבאים יינתן הסבר לסעיפי התקן.

שדות

1.47 OPENAPI – חובה

הגרסה של OpenApi. ערכים קבועים- תומך בגרסאות 3.0.0-3.0.3

```
openapi: 3.0.0
```

1.48 INFO OBJECT- חובה

פרטי וגרסת ה API.

מכיל את האובייקטים הבאים:

שם אובייקט	חובה / רשות	הסבר
Title	חובה	שם כללי ל API. מומלץ להיות שם מייצג לפעילות של ה API.
Version	חובה	גרסת ה API- משמשת לתיעוד שינויים ב API, הגרסה יכולה להיות גם במבנה של תאריך לדוגמא 23/09/20.
Description	חובה	משמש כדי להציג מידע כללי ועסקי עבור ה API, שינויים בגרסאות, תיקוני באגים וכו'.
Contact	חובה	פרטי איש הקשר בחברה ליצירת קשר אודות ה API – הכולל שם איש הקשר, מייל וURL לפרטים נוספים.
TermsOfService	חובה	כתובת לתנאי שימוש ב API – חייב להיות בפורמט של URL.
License	רשות	מכיל את שם הרישיון ו URL לפרטי הרישיון לשימוש ב API.

```yaml
openapi: 3.0.0
info:

  # You application title. Required.
  title: Sample Pet Store App

  # API version. You can use semantic versioning like 1.0.0,
  # or an arbitrary string like 0.99-beta. Required.
  version: 1.0.0

  # API description. Arbitrary text in CommonMark or HTML.
  description: This is a sample server for a pet store.

  # Link to the page that describes the terms of service.
  # Must be in the URL format.
  termsOfService: http://example.com/terms/

  # Contact information: name, email, URL.
  contact:
    name: API Support
    email: support@example.com
    url: http://example.com/support

  # Name of the license and a URL to the license description.
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html

  # Link to the external documentation (if any).
  # Code or documentation generation tools can use description as the text of the link.
  externalDocs:
    description: Find out more
    url: http://example.com
```

1.49 SERVERS - חובה

מערך של השרתים המארחים את ה API, ניתן להגדיר בקובץ אחד כמה סביבות.

```yaml
servers:
  - url: https://api.example.com/v1
    description: Production server (uses live data)
  - url: https://sandbox-api.example.com:8443/v1
    description: Sandbox server (uses test data)
```

כל אובייקט במערך מכיל:

שם אובייקט	חובה / רשות	הסבר
URL	חובה	ה URL מחליף את ה Host, Schemes ו- Base Path מגרסה 2.0. `scheme://host[:port][/path]` מבנה ה URL:
Scheme	חובה	סוג הפרוטוקול בו חשוף ה API, לדוגמא:

		``` https://api.example.com https://api.example.com:8443/v1/reports http://localhost:3025/v1 http://10.0.81.36/v1 ws://api.example.com/v1 wss://api.example.com/v1 ```
Host	חובה	מכיל את שם השרת אותו חושפים ללקוח לבצע פניות.
Port	רשות	במידה ומשתמשים ב port שונה מה port הקבוע לפרוטוקול לדוגמא עבור HTTPS פורט 443.
BasePath	רשות	הכתובת הרלטיבית לכל ה End Points (paths) המוגדרים בקובץ. ``` ttps://api.example.com/v1/users?role=admin&status=active _____/___/ _____/ server URL endpoint query parameters path ``` אופציונלי- להגדיר בתוך ה path את הנתיב המלא כולל הגרסה - פחות מומלץ
Description	רשות מומלץ להוסיף לצורך תיעוד	ניתן להוסיף גם description עבור כל סביבה, לדוגמא סביבת טסט או פרוד

פרמטרים בServers URL:

ניתן להשתמש בגרסה 3.0 בפרמטרים ב URL ע"י הגדרת משתנה במבנה {Variable Name}. בשונה ממשתנים המוגדרים ב Path. משתנה URL תמיד מוגדר כ String. שם

```yaml
servers:
  - url: https://{customerId}.saas-app.com:{port}/v2
    variables:
      customerId:
        default: demo
        description: Customer ID assigned by the service provider
      port:
        enum:
          - '443'
          - '8443'
        default: '443'
```

המשתנה חייב להיות זהה ב URL למשתנה המוגדר (case sensitive), לדוגמא:

שם אובייקט	חובה / רשות	הסבר
Default	חובה	הערך הקבוע במידה והמשתמש לא שלח ערך אחר.
Description	רשות	תיאור של המשתנה
Enum	רשות	מערך ערכים קבועים שיכולים להתקבל למשתנה זה, חייב לכלול את הערך של Default

לדוגמא דרך נוספת לתיאור סביבות ואזורים:

```yaml
servers:
  - url: https://{environment}.example.com/v2
    variables:
      environment:
        default: api    # Production server
        enum:
          - api         # Production server
          - api.dev     # Development server
          - api.staging # Staging server
```

```yaml
servers:
  - url: https://{region}.api.cognitive.microsoft.com
    variables:
      region:
        default: westus
        enum:
          - westus
          - eastus2
          - westcentralus
          - westeurope
          - southeastasia
```

1.50 PATHS- חובה

```yaml
paths:
  /users/{id}:
    summary: Represents a user
    description: >
      This resource represents an individual user in the system.
      Each user is identified by a numeric `id`.

    get:
      ...
    patch:
      ...
    delete:
      ...
```

אובייקט זה מתאר את כל הנתיבים האפשריים ב API (Resources).

כל נתיב יתחיל תמיד ב / ויתאר את ה Resource אשר נעשה עליו פעולה לדוגמא /users לפעולות כלליות עבור כל המשתמשים ו- /users/{id} לפעולות עבור משתמש ספציפי.

ב Open Api השילוב של Path ו HTTP Method מתאר Operation, לדוגמא: פניה ב GET לנתיב /users/{id} היא פעולה (Operation) עבור משיכת פרטים אודות המשתמש, פניה ב PATCH לאותו נתיב תשמש לעדכון הפרטים.

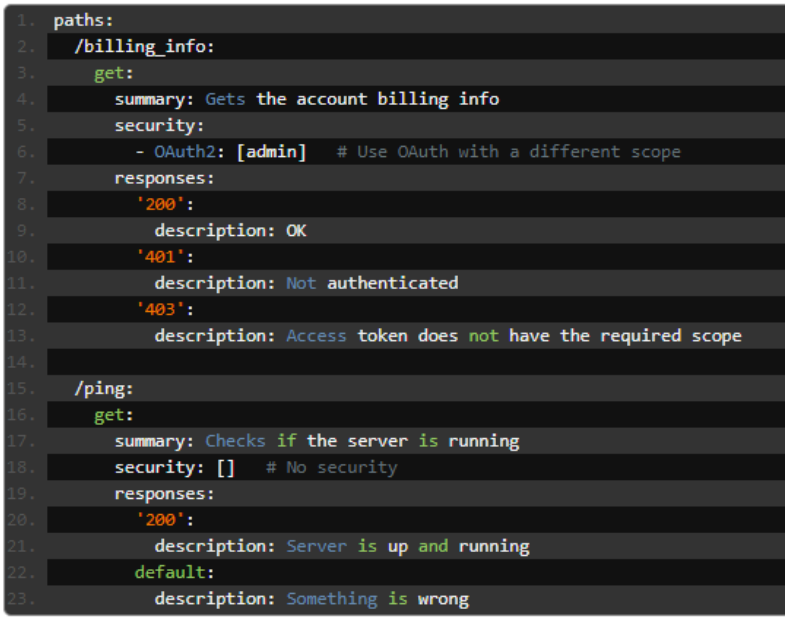
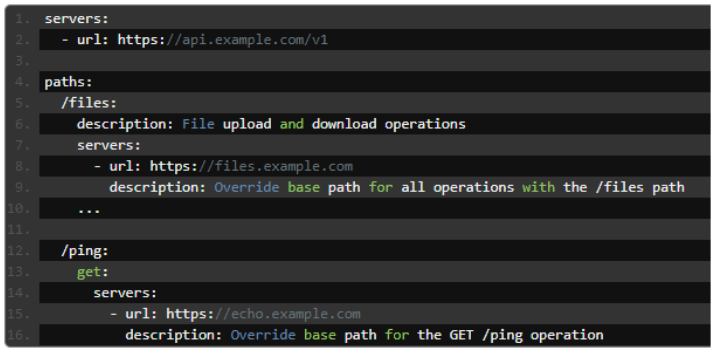
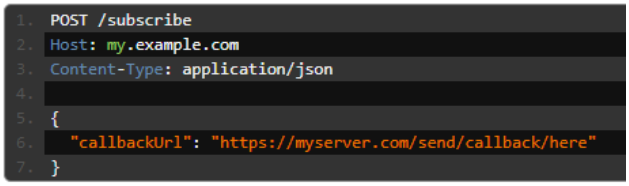
ניתן להגדיר ברמת הנתיב Summary ו Description לתיאור כללי של כל ה Operations (כפי שבדוגמא למעלה).

כל נתיב מכיל את האובייקטים הבאים:

שם אובייקט	חובה / רשות	הסבר
Summary	רשות	תיאור קצר (שורה אחת) לתיאור ה Resource.
Description	רשות	תיאור ארוך (מספר שורות) לתיאור ה Resource.

Deprecated	רשות	מציין אם נתיב זה יוסר בגרסה הבאה של ה API. משתנה Boolean, default=false. לדוגמא:
		`/pet/findByTags:`
		`  get:`
		`  deprecated: true`
		בדרך כלל יוצג ב Swagger UI בצורה הבאה:
		

Security	רשות	מערך המתאר את שיטות האוטנטיקציה הנדרשות ל Operation.

הגדרה זאת תעקוף (Override) להגדרת ה Security הכללית ב  .Swagger		
מערך להגדרת השרתים ל Operation ספציפי. יש להגדיר את מערך Servers כפי שהוגדר ל Swagger הכללי. הגדרה זאת  תעקוף (Override) את הגדרת ה Servers הכללית.	רשות	**Servers**
ניתן בגרסה 3.0 לתאר את המבנה של שירות אליו נרשם הלקוח (subscription). לדוגמא באמצעות שליחת URL  (webhook), בו הלקוח יקבל עדכונים מהשרת: הגדרת השירות:	רשות	**Callback**

Operations	חובה	לכל נתיב חובה להגדיר לפחות Operation אחד. גרסה 3.0 תומכת בפעולות הבאות: כעת השרת ישלח ללקוח עדכונים: כדי לתאר את מבנה הבקשה שתשלח ללקוח, נוסיף להגדרת השירות את אובייקט ה Callback: אובייקט זה יכול להכיל כמה Events (בדוגמא myEvent למעלה), כאשר כל אחד יכלול אחד או יותר Callback URLs. ההגדרה של הנתיב בתור {$request.body#/callbackUrl} מציינת כי הנתיב הוא נגזרת של הפרמטר callbackUrl מתוך גוף הבקשה. לקריאה נוספת בנושא: https://swagger.io/docs/specification/callbacks

```yaml
paths:
  /subscribe:
    post:
      summary: Subscribe to a webhook
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                callbackUrl:    # Callback URL
                  type: string
                  format: uri
                  example: https://myserver.com/send/callback/here
              required:
                - callbackUrl
```

```http
POST /send/callback/here
Host: myserver.com
Content-Type: application/json

{
  "message": "Something happened"
}
```

```yaml
paths:
  /subscribe:
    post:
      summary: Subscribe to a webhook
      requestBody:
        …
      callbacks:   # Callback definition
        myEvent:   # Event name
          '{$request.body#/callbackUrl}':   # The callback URL,
                                            # Refers to the passed URL
            post:
              requestBody:   # Contents of the callback message
                required: true
                content:
                  application/json:
                    schema:
                      type: object
                      properties:
                        message:
                          type: string
                          example: Some event happened
                      required:
                        - message
              responses:   # Expected responses to the callback message
                '200':
                  description: Your server returns this code if it accepts the callback
```

		.get, post, put, patch, delete, head, options, and trac לדוגמא תיאור של get: paths: /users/{id}: get: tags: - Users summary: Gets a user by ID. description: >- A detailed description of the operation. Use markdown for rich text representation, such as **bold**, *italic*, and [links](https://swagger.io). operationId: getUserById parameters: - name: id in: path description: User ID required: true schema: type: integer format: int64 responses: '200': description: Successful operation content: application/json: schema: $ref: '#/components/schemas/User' externalDocs: description: Learn more about user operations provided by this API. url: http://api.example.com/docs/user-operations/

כל Operation מכיל את האובייקטים הבאים:

שם אובייקט	חובה / רשות	הסבר
Tags	רשות	מערך של שמות לוגיים המאפשרים איגוד וחיפוש APIs. לדוגמא: הגדרת שם Resource או שיוך עסקי.
Summary	רשות	תיאור קצר (שורה אחת) לתיאור ה Operation.
Description	רשות	תיאור ארוך (מספר שורות) לתיאור ה Operation.
OperationId	רשות	מזהה חד ערכי לכל Operation. במידה וקיים ב Swagger חייב להיות שונה לכל Operation. משמש לזיהוי נוסף באפליקציות מסוימות המזהות את ה Operation לפי ערך זה.
Parameters	רשות	מערך המתאר את הפרמטרים האפשריים ל Operation. בשונה מגרסה 2.0, במערך זה ניתן להגדיר את הפרמטרים המתקבלים ב Path, Query, HTTP Header ו cookie. לא ניתן להגדיר את ה Body במערך זה

כל אובייקט במערך Parameters יכיל:

1.36 פרמטרים כלליים

שם אובייקט	חובה / רשות	הסבר

Name	חובה	שם הפרמטר. השם הוא case sensitive במידה והמשתנה הוא ב Path, הוא חייב להיות מוגדר באותו שם בנתיב ובאובייקט ה Parameter
In	חובה	מיקום המשתנה. שדות קבועים: "path", "header", "query" ו "cookie". – תיאור הפרמטרים מובא בהמשך לטבלה.

Required	חובה	מגדיר אם הפרמטר אופציונלי מבחינת הלקוח. משתנה Boolean, default=false.
		במידה והמשתנה ב Path חובה להיות true, משום שמשתנה
		`parameters:`
		`- in: path`
		`name: userId`
		`schema:`
		`type: integer`
		`required: true # <----------`
		`description: Numeric ID of the user to get.`
		זה חייב להישלח ע"י הלקוח.

Description	רשות	תיאור ארוך (מספר שורות) לתיאור ה Parameter

Deprecated	רשות	מציין אם משתנה זה יוסר בגרסה הבאה של ה API. משתנה
		`- in: query`
		`name: format`
		`required: true`
		`schema:`
		`type: string`
		`enum: [json, xml, yaml]`
		`deprecated: true`
		`description: Deprecated, use the appropriate `Accept` header instead.`
		Boolean, default=false.

allowEmptyValue	רשות	רלוונטי עבור Query Parameters בלבד.
		מאפשר לשלוח משתנה ללא ערך. משתנה Boolean, default=false.
		לדוגמא, אם נרצה לשלוח משתנה בשם metadata ללא ערך:
		`GET /foo?metadata`
		נגדיר אותו כך:
		`parameters:`
		`- in: query`
		`name: metadata`
		`schema:`
		`type: boolean`
		`allowEmptyValue: true # <-----`

allowReserved	רשות	רלוונטי עבור Query Parameters בלבד.

		התווים ":/?#[]@!$&'()*+,;=" מוגדרים כתווים מיוחדים לשימוש ב URI. במידה ויש צורך לקבל בפרמטר Query אחד מהתווים האלה, מומלץ לקודד אותו באמצעות url-encode.
		לדוגמא, התו "/" יקודד ל "F%2":
		`GET /file?path=quotes%2Fh2g2.txt`
		במידה ונרצה שהתו "/" יעבור ללא קידוד, נגדיר אותו כך:
		`parameters:`
		`- in: query`
		`name: path`
		`required: true`
		`schema:`
		`type: string`
		`allowReserved: true # <-----`
		הפניה לשירות תתבצע כך:
		`GET /file?path=quotes/h2g2.txt`

Schema	חובה	**ניתן להחליף ב Content** תיאור סוג הפרמטר. ניתן להגדיר הגבלות נוספות לדוגמא: minimum, maximum, enum וכו'. הסבר מלא יפורט בחלק מודלי נתונים בהמשך המסמך.
Content	רשות	Open Api תומך בהעברת מערכים ואובייקטים מורכבים (Serialization) כפרמטרים ב API. ניתן לתאר גם כמה סוגי מדיה שונים ע"י שימוש באובייקט CONTENT במקום SCHEMA (אובייקט CONTENT מכיל בתוכו SCHEMA). לקריאה נוספת בנושא: https://swagger.io/docs/specification/serialization

Example	רשות	וגמה של הפרמטר. הערך חייב להתאים ל schema המוגדרת ל Parameter.
		במידה והוגדרה סכמה עם דוגמא, הדוגמא שהוגדרה ב Operation תוצג.
		`parameters:`
		`- in: query`
		`name: limit`
		`schema:`
		`type: integer`
		`minimum: 1`
		`example: 20`

Examples	רשות	מיפוי של דוגמאות, הכולל שם כללי לכל אובייקט של דוגמא, Summary לתיאור של כל דוגמא, ו Value החייב להתאים ל Schema המוגדרת ל Parameter.

ניתן להגדיר example או examples עבור כל Operation, אבל לא את שניהם.

```yaml
    parameters:
      - in: query
        name: ids
        description: One or more IDs
        required: true
        schema:
          type: array
          items:
            type: integer
        style: form
        explode: false
        examples:
          oneId:
            summary: Example of a single ID
            value: [5]   # ?ids=5
          multipleIds:
            summary: Example of multiple IDs
            value: [1, 5, 7]   # ?ids=1,5,7
```

הסבר על אפשרויות הפרמטר in:

1. Query Parameter

המשתנים הנפוצים ביותר, המועברים בסוף ה URI עם הפרדה של "?", בפורמט של key=value, עם הפרדה של "&" בין משתנים, לדוגמא:

```
GET /pets/findByStatus?status=available
GET /notes?offset=100&limit=50
```

```yaml
    parameters:
      - in: query
        name: offset
        schema:
          type: integer
        description: The number of items to skip before starting to collect the result set
      - in: query
        name: limit
        schema:
          type: integer
        description: The numbers of items to return
```

2. Header Parameter

משתמש להגדרת משתנים המועברים בתור Custom HTTP Header ע"י הלקוח. מומלץ להגדיר שמות Headers אפליקטיביים בשם המתחיל ב: "x-".

לא ניתן להגדיר בצורה זאת את הערכים עבור:

Content-type – נגזר מתוך אובייקט Content ב requestBody.

Accept – נזר מתוך אובייקט Content ב Responses.

Authorization – נגזר מתוך Security ו SecuritySchema.

לדוגמא, פניה לשירות באמצעות Header בשם X-Request-ID המכיל UUID:

```
GET /ping HTTP/1.1
Host: example.com
X-Request-ID: 77e1c83b-7bb0-437b-bc50-a7a58e5660ac
```

```
paths:
  /ping:
    get:
      summary: Checks if the server is alive
      parameters:
        - in: header
          name: X-Request-ID
          schema:
            type: string
            format: uuid
          required: true
```

3. Cookie Parameter

מוגדר בתור HTTP Header בשם Cookie בו יכולים לעבור מספר פרמטרים בפורמט key=value, המופרדים באמצעות נקודה-פסיק ורווח "; ". לדוגמא:

```
GET /api/users
Host: example.com
Cookie: debug=0; csrftoken=BUSe35dohU3O1MZvDCUOJ
```

```
      parameters:
        - in: cookie
          name: debug
          schema:
            type: integer
            enum: [0, 1]
            default: 0
        - in: cookie
          name: csrftoken
          schema:
            type: string
```

1.37 requestBody

פרמטר רשות אשר מתאר את מבני גוף הבקשה (BODY) האפשריים ב API.

בגרסה 3.0 ניתן להגדיר פה סוגי מדיה שונים לכל לאותה בקשה, למשל מסוג: JSON, XML, form data, plain text ועוד.

```yaml
paths:
  /pets:
    post:
      summary: Add a new pet

      requestBody:
        description: Optional description in *Markdown*
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Pet'
          application/xml:
            schema:
              $ref: '#/components/schemas/Pet'
          application/x-www-form-urlencoded:
            schema:
              $ref: '#/components/schemas/PetForm'
          text/plain:
            schema:
              type: string

      responses:
        '201':
          description: Created
```

האובייקט מכיל:

רשימת תתי האובייקטים האפשריים לאובייקט זה:

שם אובייקט	חובה / רשות	הסבר

Content	חובה	מתאר את מבנה התשובה החוזרת ללקוח.
		אפשר לתאר סוגי מדיה שונים שה API מאפשר לקבל כבקשה מה Client. כאשר נגדיר application/json לדוגמא, ה client יצטרך לשלוח content-type עם הערך application/json.
		ניתן להגדיר את סוג המדיה בשימוש ב wildcards כדי לתאר סוגים שונים באובייקט אחד. לדוגמא, העלאת קובץ תמונה
		`paths:`
		`  /avatar:`
		`  put:`
		`  summary: Upload an avatar`
		`  requestBody:`
		`  content:`
		`  image/*: # Can be image/png, image/svg, image/gif, etc.`
		`  schema:`
		`  type: string`
		`  format: binary`
		מסוגים שונים יכול להיות מוגדר ע"י image/*, לדוגמא:

		הגדרה של media בתור */* שווה ערך להגדרת application/octet-stream. בעבור כל סוג מדיה נדרש להגדיר מידע נוסף, אשר מוצג מתחת לטבלה זו.
Description	רשות	תיאור ארוך (מספר שורות) לתיאור מבנה הבקשה.
Required	רשות	מגדיר אם מבנה הבקשה אופציונלי מבחינת הלקוח. משתנה Boolean, default=false. אם ה Body הוא required יש לשנות ערך זה ל true.

4. בעבור כל מדיה נדרש להגדיר את תתי האובייקטים הבאים:

שם אובייקט	חובה / רשות	הסבר
Schema	חובה	ניתן להגדיר סכמה מקומית או ע"י שימוש ב $ref.

Example	רשות	דוגמה של הבקשה. האובייקט והערכים חייבים להתאים ל schema המוגדרת למדיה.
		במידה והוגדרה סכמה עם דוגמא, הדוגמא שהוגדרה במדיה
		`requestBody:`
		`  content:`
		`  application/json:`
		`  schema:`
		`  $ref: '#/components/schemas/Pet'`
		`  example:`
		`  name: Fluffy`
		`  petType: dog`
		תוצג.

Examples	רשות	מיפוי של דוגמאות, הכולל שם כללי לכל אובייקט של דוגמא, Summary לתיאור של כל דוגמא, ו Value החייב להתאים ל Schema המוגדרת למדיה.

ניתן להגדיר example או examples עבור כל מדיה, אבל לא את שניהם.

```yaml
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Pet'
            examples:

              dog:  # <--- example name
                summary: An example of a dog
                value:
                  # vv Actual payload goes here vv
                  name: Fluffy
                  petType: dog

              cat:  # <--- example name
                summary: An example of a cat
                externalValue: http://api.example.com/examples/cat.json   # cat.json contains {"name": "Tiger", "petType": "cat"}

              hamster:  # <--- example name
                $ref: '#/components/examples/hamster'

components:
  examples:
    hamster:  # <--- example name
      summary: An example of a hamster
      value:
        # vv Actual payload goes here vv
        name: Ginger
        petType: hamster
```

אובייקט חובה בעבור כל Operation, נדרש להגדיר קוד HTTP אחד, בדרך כלל קוד הצלחה.

```yaml
paths:
  /users:
    get:
      summary: Get all users
      responses:
```

```yaml
      responses:
        '200':
          description: OK
        '400':
          description: Bad request. User ID must be an integer and larger than 0.
        '401':
          description: Authorization information is missing or invalid.
        '404':
          description: A user with the specified ID was not found.
        '5XX':
          description: Unexpected error.
```

לכל Operation יש בדרך כלל קוד הצלחה אחד וכמה קודי שגיאה, חובה לתאר באמצעות Description לכל

```yaml
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'

        # These two error responses have the same schema
        '400':
          description: Bad request
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
        '404':
          description: Not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
```

קוד את השגיאה המוכרת ו/ או קוד כללי. לדוגמא:

ניתן להגדיר בנוסף לקוד ההצלחה, תשובה קבועה (default), במידה וכל השגיאות מוגדרות באותו מבנה.
לדוגמא אובייקט Responses המכיל קוד הצלחה ושני קודי שגיאה:

```yaml
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'

        # Definition of all error statuses
        default:
          description: Unexpected error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
```

ניתן להגדיר במקום קודי השגיאה הספציפיים default המגדיר באופן כללי את השגיאות מבנה התשובה מוגדר באמצעות אובייקט Content בזהה לבקשה.

לקודי שגיאה מסוימים לדוגמא- 204 אינם מכילים BODY. כדי לציין שהם ריקים יש להגדירם ללא אובייקט Content, לדוגמא:

```yaml
      responses:
        '204':
          description: The resource was deleted successfully.
```

ניתן להגדיר לתשובה אחת מספר סכמות שונות ע"י שימוש ב oneOf או anyOf, לדוגמא:

```yaml
      responses:
        '200':
          description: A JSON object containing pet information
          content:
            application/json:
              schema:
                oneOf:
                  - $ref: '#/components/schemas/Cat'
                  - $ref: '#/components/schemas/Dog'
                  - $ref: '#/components/schemas/Hamster'
```

לתיאור Headers אפליקטיביים עבור התשובה- אופציונלי, יש להגדיר לכל קוד אובייקט Headers, המכיל מיפוי של שם Header, עם Schema ו Description. לדוגמא:

```yaml
paths:
  /ping:
    get:
      summary: Checks if the server is alive.
      responses:
        '200':
          description: OK
          headers:
            X-RateLimit-Limit:
              schema:
                type: integer
              description: Request limit per hour.
            X-RateLimit-Remaining:
              schema:
                type: integer
              description: The number of requests left for the time window.
```

לא ניתן בגרסה הנוכחית להגדיר ברמה גלובלית עבור כל ה Responses, לקודים שונים או Operations שונים, Headers קבועים. יש להגדיר עבור כל קוד בנפרד.

במידה ויש חזרה על תשובות ל Operations שונים, מומלץ להגדיר אותם באובייקט components ולבצע שימוש חוזר באמצעות "$ref" לאובייקט המתאים.

1.51 COMPONENTS - אופציונלי

אובייקט זה משמש כאחסון להגדרות ב API, בכדי לבצע שימוש חוזר לתיאור של אובייקטים נפוצים.
Components מחליף מגרסה 2.0 את האובייקטים: definitions, parameters, responses ו
SecurityDefinitions, אשר אוגדו תחת Components. Definitions מוגדר כ schemas ו securityDefinitions

```yaml
components:
  # Reusable schemas (data models)
  schemas:
    ...

  # Reusable path, query, header and cookie parameters
  parameters:
    ...

  # Security scheme definitions (see Authentication)
  securitySchemes:
    ...

  # Reusable request bodies
  requestBodies:
    ...

  # Reusable responses, such as 401 Unauthorized or 400 Bad Request
  responses:
    ...

  # Reusable response headers
  headers:
    ...

  # Reusable examples
  examples:
    ...

  # Reusable links
  links:
    ...

  # Reusable callbacks
  callbacks:
    ...
```

מוגדר כ securitySchmes. (שים לב! secutiySchmEs ולא securitySchmAs).

מבנה Components

- Schemas
- Parameters
- SecuritySchemes
- requestBodies
- responses
- headers
- examples
- links
- callbacks

השימוש ברכיב (Component) מתבצע ע"י שימוש באובייקט "$ref", במבנה הבא:

```yaml
$ref: '#/components/<type>/<name>'
```

לדוגמא, הפניה לסכמה של משתמש תתבצע ע"י:

כל האובייקטים מלבד ה SecuritySchemes קיימים כחלק מאובייקט ה Operation. הגדרתם

```
$ref: '#/components/schemas/User'
```

כComponenets משמשת לרוב לשימוש חוזר באותם אובייקטים.

SecuritySchemes מגדיר את סוגי האוטנטיקציה האפשריים. אובייקט זה לא מגדיר את שיטת האוטנטיקציה הנדרשת!

```yaml
components:
  securitySchemes:

    BasicAuth:
      type: http
      scheme: basic

    BearerAuth:
      type: http
      scheme: bearer

    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key

    OpenID:
      type: openIdConnect
      openIdConnectUrl: https://example.com/.well-known/openid-configuration

    OAuth2:
      type: oauth2
      flows:
        authorizationCode:
          authorizationUrl: https://example.com/oauth/authorize
          tokenUrl: https://example.com/oauth/token
          scopes:
            read: Grants read access
            write: Grants write access
            admin: Grants access to admin operations
```

ניתן להגדיר כמה סוגי אוטנטיקציה באובייקט זה. כל הגדרה תכלול שם שרירותי לשיטה, המוגדר בתור אובייקט הכולל type – חובה.

סוגי האוטונטיקציה האפשריים להגדרה:

1. HTTP - Authorization Header המכיל Basic או Bearer.
   - יש להגדיר type: http.
   - Scheme - basic עבור הזדהות מול ldap לרוב. Bearer לשימוש ב jwt.
2. API Key - משמש לרוב לזיהוי אפליקציה.
   - יש להגדיר type: apiKey
   - In - יכול להיות query, header או cookie.
   - Name - שם הפרמטר / header
3. Oauth2 – משמש לתיאור oauth2 ה flows הנתמכים, ניתן לתאר יותר מאחד:
   - authorizationCode

- Implicit
- Password
- clientCredentials

לכל flow יש להגדיר:

1. authorizationUrl- נתמך ב authorizationCode ו Implicit.
2. tokenUrl- נתמך ב authorizationCode, Password ו clientCredentials.
3. refreshUrl- אופציונלי בכל ה flows, אם ישנה תמיכה בקבלת token חדש.
4. Scopes- אופציונלי בכל ה flows. במידה ואין scope מומלץ להגדיר את השדה אם אובייקט ריק { }.

OpenIdConnect - לתיאור תהליך גילוי open Id Server.

יכיל openIdConnectUrl - כתובת לשרת.

1.52 SECURITY- אופציונלי

```yaml
security:
  - ApiKeyAuth: []
  - OAuth2:
      - read
      - write

# The syntax is:
# - scheme name:
#     - scope 1
#     - scope 2
```

מערך המגדיר גלובלית את שיטת האוטנטיקציה הנדרשת לכל ה Operations ב swagger.

Security משתמש בהגדרות ה SecuritySchemes מתוך אובייקט ה components.

כל אובייקט במערך יהיה אחד משמות האוטנטיקציה שהוגדרו ב SecuritySchemes, כאשר שיטת האוטנטיקציה תומכת בשימוש ב scopes (oauth2 ו openIdConnect), יש לתאר במערך את ה scopes הנדרשים לשימוש ב API.

כאשר אין שימוש ב scopes (לא נתמך או שלא הוגדר), יש להגדיר בערך מערך ריק '[]'.
ניתן להגדיר לכל scope תיאור ע"י הוספת ערך לשם ה scope, לדוגמה:

```
        scopes:
          read: Grants read access
          write: Grants write access
          admin: Grants access to admin operations
```

ניתן להגדיר יותר משיטת אוטנטיקציה אחת, לדוגמא:

```
security:    # A OR B
  - A
  - B
```

```
security:    # A AND B
  - A
    B
```

```
security:    # (A AND B) OR (C AND D)
  - A
    B
  - C
    D
```

לדוגמא, תיאור אוטנטיקציה מלא עבור oauth2 בשימוש ב authorization code:

שלב ראשון הגדרת ה securitySchemes ב components

שלב שני הגדרת ה security כללית / ספציפית ל Operation

```
# Step 1 - define the security scheme
components:
  securitySchemes:
    oAuthSample:    # <---- arbitrary name
      type: oauth2
      description: This API uses OAuth 2 with the implicit grant flow. [More info](https://api.example.com/docs/auth)
      flows:
        implicit:   # <---- OAuth flow(authorizationCode, implicit, password or clientCredentials)
          authorizationUrl: https://api.example.com/oauth2/authorize
          scopes:
            read_pets: read your pets
            write_pets: modify pets in your account

# Step 2 - apply security globally...
security:
  - oAuthSample:
    - write_pets
    - read_pets

# ... or to individual operations
paths:
  /pets:
    patch:
      summary: Add a new pet
      security:
        - oAuthSample:
          - write_pets
          - read_pets
      ...
```

למידע נוסף אודות אוטנטיקציה:

https://swagger.io/docs/specification/authentication

1.53 Tags - אופציונלי

משמש לאיגוד APIs לקבוצות לפי לוגיקה עסקית לרוב, יכול לעזור בניהול וחיפוש.

ה Tags המוגדרים לכל Operation חייב להיות זהים ל Tags הגלובליים.

```yaml
tags:
  - name: pets
    description: Everything about your Pets
    externalDocs:
      url: http://docs.my-api.com/pet-operations.htm
  - name: store
    description: Access to Petstore orders
    externalDocs:
      url: http://docs.my-api.com/store-orders.htm
```

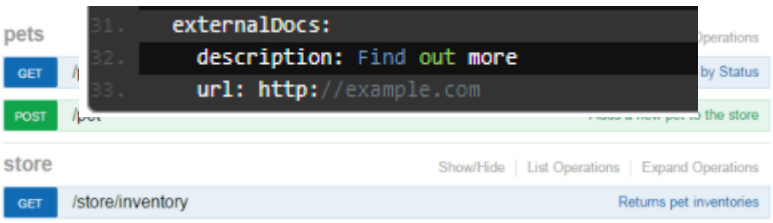
ניתן אופציונלית להוסיף לכל Tag גם description לתיאור, ו externalDocs לקישור למידע נוסף, לדוגמא:

ה Tags משמשים בנוסף לאיגוד ה Operations ב Swagger-UI לקבוצות, לדוגמא:

1.54 EXTERNALDOCS - אופציונלי

משמש להוספת קישור למידע חיצוני אודות ה API.

חובה להגדיר URL למסמך וניתן להוסיף description לקישור, לדוגמה:



10. גרסת **OPEN API 3.0** תומכת בגרסת **JSON SCHEMA DRAFT 05**.

כדי להגדיר מודל נתונים יש להגדיר באובייקט הסכמה type. ה- types הנתמכים:

- String – כולל קבצים
- Number
- Integer
- Boolean
- Array
- Object

Type אינו מערך, במידה ורוצים להגדיר יותר מסוג נתונים אחד יש להגדיר כך:

```yaml
# Incorrect
type:
  - string
  - integer
```

```yaml
# Correct
oneOf:
  - type: string
  - type: integer
```

1.55 NUMBERS

ישנם 2 סוגי מספרים, number ו Integer, כאשר number כולל גם מספרים שלמים. ניתן לציין format להגדרת סוג מספר ספציפי יותר.

type	format	Description
number	–	Any numbers.
number	float	Floating-point numbers.
number	double	Floating-point numbers with double precision.
integer	–	Integer numbers.
integer	int32	Signed 32-bit integers (commonly used integer type).
integer	int64	Signed 64-bit integers (long type).

מספרים אשר נמצאים בתוך מרכאות, לדוגמא: "17", נחשבים string ולא מספר.

1.56 מינימום ומקסימום

ניתן להגדיר לכל מספר גם הגבלת גודל ע״י הוספת minimum ו maximum, לדוגמא:

```yaml
type: integer
minimum: 1
maximum: 20
```

במצב ברירת המחדל, ערכי המינימום והמקסימום יהיו חלק מהטווח.

```
minimum ≤ value ≤ maximum
```

בכדי להגביל את ערכי, יש להוסיף exclusiveMinimum ו exclusiveMaximum. לדוגמא: הגדרת מספר

```yaml
type: number
minimum: 0
exclusiveMinimum: true
maximum: 50
```

עשרוני בטווח בין 0-50 כאשר 0 אינו נכלל:

1.57 כפולות

ניתן להגדיר כי מספר הוא מכפלה של מספר אחר, לדוגמא:

```yaml
type: integer
multipleOf: 10
```

משתנה זה יקבל את הערכים: 10, 20, 30, 0, 10-, 20- וכו׳.

ניתן להגדיר כפולה למספרים עשרוניים אבל אינו מומלץ בגלל אמינות חישוב ועיגול המספר.

1.58 STRINGS

מחרוזת שורות, מוגדר ע״י type: string. String מהווה כל תו המוגדר בתוך מרכאות, לדוגמא: ”value“.

ניתן להגביל מחרוזת בגודל מינימלי ומקסימלי ע״י:

```yaml
type: string
minLength: 3
maxLength: 20
```

ניתן להגדיר format עבור כל string המגדיר את המבנה של המחרוזת. סוגי הפורמטים הקיימים בגרסה 3.0:

1. Date- תאריך מלא במבנה YYYY-MM-DD, המוגדר לפי RFC 3339, section 5.6[1]. לדוגמא: 2020-10-15.
2. Date-time- תאריך ושעה מלאים במבנה YYYY-MM-DD-THH:MM:SSZ, המוגדר לפי RFC 3339, section 5.6. לדוגמא: 2020-10-15T12:00:00Z.
3. Password- משמש להגדרת משתנה לשיש צורך להסתירו בהצגת הערך ב UI.

[1] https://tools.ietf.org/html/rfc3339#section-5.6

4. Byte- מחרוזת המקודדת בפורמט base64.

5. Binary- משמש להגדרת קובץ.

ניתן להגדיר כל format אחר שאינו נתמך, לדוגמא: email, uuid, uri, hostname, ipv4 וכו'.

כלים אשר תומכים בפורמטים האלה יוכלו להשתמש לוולידציה, אחרת יתמכו במשתנה כ string ללא פורמט.

1.59 PATTERN

```yaml
ssn:
  type: string
  pattern: '^\d{3}-\d{2}-\d{4}$'
```

ניתן להגדיר לכל מחרוזת את התבנית המאופשרת ע"י שימוש ב regex (regular expression). הערך שיתקבל יהיה חייב להיות מתאים לתבנית.
הכתיב של ה regex לקוח מתוך JavaScript (ספציפית ECMA 262[2]).

Regex הוא case sensitive לכן הגדרה של [a-z] שונה מהגדרה של [A-Z].

1.60 BOOLEAN

סוג זה יכול לקבל רק 2 סוגי ערכים true או false (ללא מרכאות!). הערך null אינו קביל.

1.61 NULL

לא קיים בגרסה 3.0 סוג ערך null אלא מאפשר להגדיר סוגים אחרים לקבל ערך null ע"י הגדרת nullable: true.

```yaml
# Correct
type: integer
nullable: true

# Incorrect
type: null

# Incorrect as well
type:
  - integer
  - null
```

1.62 ARRAYS

מערך של משתנים. כאשר מגדירים type: array, חובה להגדיר אובייקט items. לדוגמא מערך מחרוזות:

```yaml
type: array
items:
  type: string
```

[2] https://www.ecma-international.org/ecma-262/5.1/#sec-15.10.1

ניתן להגדיר מערך בתוך מערך, לדוגמא:

```yaml
# [ [1, 2], [3, 4] ]
type: array
```

```yaml
# [ {"id": 5}, {"id": 8} ]
type: array
items:
  type: object
  properties:
    id:
      type: integer
```

ניתן להגדיר מערך של אובייקטים, לדוגמא:

ניתן להשתמש בהגדרה של אובייקט מקושר ע"י $ref, לדוגמא:

```yaml
# Array of Pets
type: array
items:
  $ref: '#/components/schemas/Pet'
```

ניתן להגדיר מערכים עם סוגי אובייקטים שונים, לדוגמא:

```yaml
type: array
items: {}

# [ "hello", -2, true, [5.7], {"id": 5} ]
```

```yaml
items:
  oneOf:
    - $ref: '#/components/schemas/Cat'
    - $ref: '#/components/schemas/Dog'
```

כאשר נגדיר את items כאובייקט כללי {}, נוכל לקבל כל ערך למערך- אינו מומלץ!

ניתן להגביל את כמות האובייקטים במערך ע"י שימוש ב minItems ו maxItems, לדוגמא:

```yaml
type: array
items:
  type: integer
minItems: 1
maxItems: 10
```

כאשר לא מוגדר למערך minItems מערך ריק [], נחשב לערך קביל.

ניתן לאכוף כי ערכי המערך יהיו חד ערכיים ע"י שימוש ב uniqueItems: true, לדוגמא:

```yaml
type: array
items:
  type: integer
uniqueItems: true

# [1, 2, 3] - valid
# [1, 1, 3] - not valid
# [ ] - valid
```

אובייקט הינו אוסף של צמדי key/value. כדי להגדיר את מבנה האובייקט יש להגדיר לכל type: object, אובייקט properties המכיל את המיפוי, לדוגמא:

```yaml
type: object
properties:
  id:
    type: integer
  name:
    type: string
```

טיפ: ב openApi מומלץ להגדיר את כל האובייקטים תחת components/schemas ולא כחלק מה Operation.

במידה ורוצים לאכוף כי שדות מסוימים באובייקט הם שדות חובה, יש להגדיר מערך required, שמות ה keys חייבים להיות תואמים לשמות שהוגדרו ב properties, לדוגמא:

```yaml
type: object
properties:
  id:
    type: integer
  username:
    type: string
  name:
    type: string
required:
  - id
  - username
```

שים לב! מערך ה required ממוקם ברמת ה properties ולא ברמת השדה, לדוגמא:

```yaml
type: object
properties:
  id:
    type: integer
    required: true  # Wrong!

required:           # Correct
  - id
```

לא ניתן להגדיר required: [], אם כל השדות הם אופציונליים אין לציין required.

READ/WRITE-ONLY 1.64

ניתן לציין כי שדה מסוים משמש לקריאה או כתיבה בלבד ע"י שימוש ב readOnly או writeOnly. זה יכול להיות שימושי למצבים בהם שירות ה GET יחזיר יותר פרטים מאשר שירות ה POST דורש ולהיפך. ע"י ציון של פרמטרים בצורה זאת נוכל להשתמש באותה סכמה בכדי להגדיר את שתי הבקשות. לדוגמא:

```yaml
type: object
properties:
  id:
    # Returned by GET, not used in POST/PUT/PATCH
    type: integer
    readOnly: true
  username:
    type: string
  password:
    # Used in POST/PUT/PATCH, not returned by GET
    type: string
    writeOnly: true
```

1.65 אובייקטים מקוננים (NESTED)

אובייקט יכול לכלול בתוכו אובייקטים אחרים, לדוגמא:

```yaml
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: integer
        name:
          type: string
        contact_info:
          # The value of this property is an object
          type: object
          properties:
            email:
              type: string
              format: email
            phone:
              type: string
```

במצב זה נעדיף לפצל את ההגדרה לשני אובייקטים עם קישור באמצעות $ref, לדוגמא:

```yaml
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: integer
        name:
          type: string
        contact_info:
          $ref: '#/components/schemas/ContactInfo'

    ContactInfo:
      type: object
      properties:
        email:
          type: string
          format: email
        phone:
          type: string
```

אובייקט ללא סכמה

ניתן להגדיר אובייקט ללא properties אשר יכול לקבל כל Key/value ע"י:

```yaml
type: object
```

הגדרה זאת שוות ערך

```yaml
type: object
additionalProperties: true
```

```yaml
type: object
additionalProperties: {}
```

1.66 כמות PROPERTIES

ניתן להגביל את כמות ה key/value הנשלחים באובייקט ע"י minProperties ו maxProperties, לדוגמא:

```yaml
type: object
minProperties: 2
maxProperties: 10
```

אופציה זאת תשמש אותנו לרוב כאשר הגדרנו אובייקט ללא סכמה ונרצה להגביל את כמות הפרטים הנשלחים.

1.67 קבצים

בניגוד לגרסה 2.0, קבצים מוגדרים ע"י type: string בציון פורמט binary או byte לקובץ מקודד ב base64.

```yaml
type: string
format: binary  # binary file contents
```

```yaml
type: string
format: byte    # base64-encoded file contents
```

1.68 ENUM

ניתן להגדיר אופציות קבועות מראש לערכים למשתנים.

לדוגמא, משתנה מיון sort המקבל 2 ערכים אפשריים asc או dsc:

```yaml
paths:
  /items:
    get:
      parameters:
        - in: query
          name: sort
          description: Sort order
          schema:
            type: string
            enum: [asc, desc]
```

מומלץ להוסיף הסבר על הערכים כחלק מה description של הפרמטר, לדוגמא:

```yaml
      parameters:
        - in: query
          name: sort
          schema:
            type: string
            enum: [asc, desc]
          description: >
            Sort order:
             * `asc` - Ascending, from A to Z
             * `desc` - Descending, from Z to A
```

```yaml
type: string
nullable: true  # <---
enum:
  - asc
  - desc
  - null        # <--- without quotes, i.e. null not "null"
```

במידה ונרצה לאפשר לקבל את הערך null. נגדיר אותו כחלק ממערך ה enum ובנוסף נגדיר nullable: true