# Подготовка на защита на ДР

**Visual Studio Code** – Редактор на програмния код (или средството използвано за реализация на програмния код) e **Visual Studio Code.**

**Бизнес правила –** Бизнес правилата могат да се определят като списъци с изявления, които дават критерии и условия, чиято цел е да информира какво може и не може да се извършва в системата.

**Бизнес логика –** Определя как да бъдат съхранявани, редактирани и създадени данните в системата, според дефинираните бизнес правила.

**Бизнес изисквания -**  Какво е нужно да бъде постигнато за да бъдат имплементирани зададените бизнес правила. Тоест те определят бизнес целите, обхвата на проекта и посоката на развитие на проекта.

**Проект –** Специфична задача за изпълнение със спецификации.

**Система –** Колекция от елементи или компоненти, взаимодействащи помежду си, според набор от правила, за да образуват единно цяло или да постигнат дадена задача.

**Функционални изисквания –** Описва поведението на системата или това, което системата трябва да прави.

**Софтуерна архитектура –** Софтуерната архитектура на системата задава организацията или структурата на различните подсистеми, компоненти или слоеве в разработения краен продукт като също и определя взаимоотношенията между тях.

**Архитектурен шаблон –** Архитектурния модел/шаблон изразява фундаменталната структурна организация или схема за софтуерните системи.

**Релационна база данни -** Релационна база данни е колекция от елементи (таблици) с данни с предварително дефинирани връзки между тях. Връзките позволяват на релационни бази данни да разделят и съхраняват данни в различни таблици, като същевременно свързват различни елементи от данни. Това позволява по-организирано структуриране и извличане на данните при работа с БД.

**Entity Relationship Diagram –** Диаграма на взаимоотношенията и обектите (ERD) e визуално представяне на различните обекти(таблици) в рамките на една система и как те са свързани един с друг.

**Таблица в БД –** Колекция от свързани данни в база от данни като всяка таблица притежава редове и колони. Колоните определят структурата на данните в таблицата, а редовете са самите попълнени данни(записи).

**UML Диаграми –** Това са диаграми, които имат за цел да създадат визуално представяне на система, заедно с нейните основни актьори, роли действия, класове, за да могат по-добре да се разберат, документира, промени или да се поддържа информация относно системата.

**Use Case Диаграми –** Use Case диаграмите или диаграмите на използване идентифицират основните функции на системата и връзката на основните актьори с тях.

**Sequence Диаграми –** Диаграмите на последователностите показват взаимодействията като последователност от съобщения.

**Уеб сървър –** Приложна програма, сървър, която позволява на даден компютър да предоставя информация на други компютри, като върната информация може да е статично съдържание (хоствано от файловата система на сървъра) или динамично (зависещо от вида на заявката се изпълнява някаква програмна логика)

**JSON –** Текстов формат, който се използва за сериализиране и предаване на структурирани данни през мрежова връзка. Използва се предимно за предаване на данни между сървър и уеб приложение, като алтернативата на JSON e XML

**Сериализиране –** Процесът, през който даден обект или структура от данни се превежда във формат, подходящ за прехвърляне през мрежа или съхранение ( например в буферен масив или файлов формат).

**API (Application Programing Interface) –** Серия от команди, инструкции, функции, ресурси които едно приложение, програма или библиотека предоставя на други такива като са дефинирани специфични правила, протоколи, които осъществяват тази комуникация. С прости думи, това е начин, който осъществява комуникация между различни софтуерни компоненти.

**Уеб услуга –** Серия от правила и технологии, които позволяват на два или повече компонента да комуникират помежду си в уеб пространството.

**REST –** REST е архитектурен стил за изграждане на приложно програмни интерфейси (API) в клиент-сървър модела. Може да се определи също и като стандарт, който ръководи проектирането и разработването на процеси, които ни позволяват да взаимодействаме с данни, съхранявани на уеб сървър. Комуникацията може да се осъществи с различни протоколи, но най-често е използван HTTP.

**URI –** Низ от символи, чрез които се идентифицира ресурс. Това е комбинацията на адреса на ресурса и неговото име: URL + URN = URI

**URL** – Низ от символи, чрез които се идентифицира адреса на ресурс

**URN** – Низ от символи, чрез които се идентифицира името на ресурс.

**REST API –** REST API организира ресурсите, хоствани в уеб сървър, в групи от уникални ресурсни идентификатори или (URI), които разграничават различните видове ресурси за данни в сървър. Клиент може да получи достъп до ресурс като извърши заявка до избрана крайна точка или URI чрез протокола от приложния слой HTTP. Заявката за да бъде успешна, трябва да се определи HTTP метод, чрез който се заявява какво точно действие се иска да се изпълни от сървъра като в REST архитектурния стил главните HTTP методи, които се използват са: GET – за прочитане на данни, POST – за създаване, PUT/PATCH – за актуализация и DELETE – за изтриване/премахване. Друго важно нещо, което трябва да се запомни при употребата на REST е че комуникацията между клиент и сървър е без запомняне на състоянията (**stateless**).

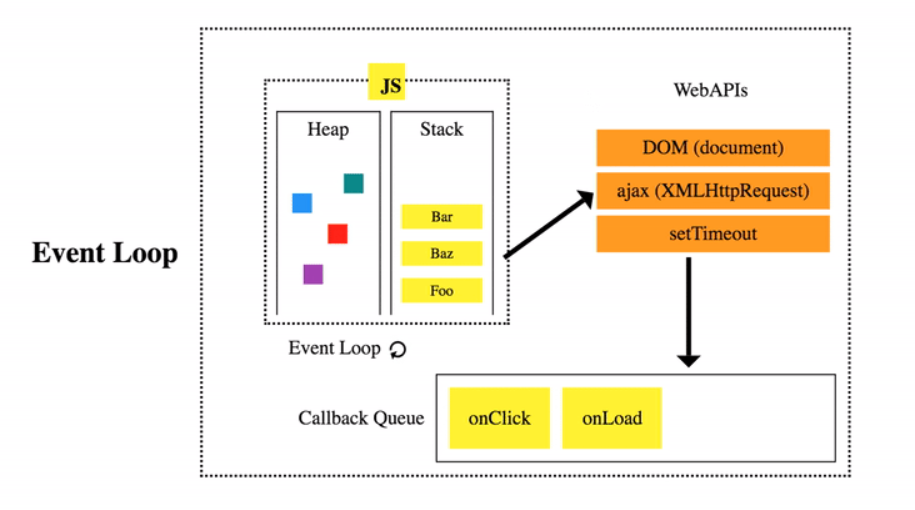
**Stateless** – Това е вид комуникация със сървър, чрез която състоянието на комуникацията не се съхранява на сървъра, тоест всяка заявка е изолирана и не зависят една от друга. Ако е нужно да се запазва състояние при този вид комуникация, то единствено трябва да се съхранява към клиентската страна.

**Софтуерна рамка –** Софтуерната рамка представлява универсална софтуерна среда, която е стандартизиран способ за разработката на софтуерни приложения, продукти и различни решения. За целта технологичните рамки могат да включват и допълнителни приложения, инструменти, библиотеки, приложно програмни интерфейси, като всички тези средства позволяват пълноценна разработка на програмна реализация.

**Библиотека -** Библиотеките са съвкупност от функции или обекти за изпълнение на различни задачи.

**Асинхронни функции -** Асинхронните функции в *Javascript* позволяват на програмата да стартира потенциално дълго изпълняваща се задача(от която не е ясно кога ще е върнат резултат) и в същото време да може да отговаря на други събития, докато тази задача се изпълнява.

**Цикъл на събитията (Event Loop) –** Цикълът на събитията e постоянно работещ процес, който позволява на Node.JS средата и Javascript да изпълнява не блокиращи операции за вход и изход независимо от факта, че Javascript е програмен език, който се изпълнява върху една нишка *(едно действие може да се изпълни само в даден момент)*. Този недостиг се преодолява чрез разтоварване на тези потенциално блокиращи операции(функции) върху системното ядро в случая на Node.JS или към приложно програмния интерфейс на браузъра в случай, когато работим само с Javascript (Front-end разработване).



**AJAX –** Асинхронен Javascript и XML е технология или съвкупност от технологии за комуникация със сървър от уеб приложение/уебсайт по асинхронен начин. Използва XMLHttpRequest обекта, който е вграден във всеки съвременен браузър, и чрез него е възможно да се създават заявки към сървър без цялата уеб страница да се презарежда наново. За трансмисия на данни AJAX използва XML и JSON текстовите формати.

**DOM (Документно обектен модел)** – Приложно програмен интерфейс, който третира HTML, XML и други езици за маркиране под формата на дървовидна структура като всеки възел от тази дървовидна структура може да се определи като обект, който притежава методи за манипулация на всички елементи в съдържанието на HTML документа.

С прости думи, това е приложно-програмен интерфейс, който позволява на програми и скриптове да получат достъп до съдържанието на обекти в HTML, XML-документи.

**Интерпретатор** – Транслиране, превеждане на програма в машинен код по един израз за дадено време

**Компилатор** – Сканира цялата програма и я превежда в машинен код наведнъж.

**World Wide Web –** Информационна система, която позволява на документи и други уеб ресурси да бъдат достъпвани чрез Интернет.

**Интернет** – Глобална компютърна мрежа, предоставяща различни информационни и комуникационни съоръжения, състояща се от взаимосвързани мрежи, използващи стандартизирани комуникационни протоколи.

**Интерпретируем език –** Програмен език, който извършва последователен анализ на командите от изходния код и ги превежда непосредствено в машинен език и изпълнява.

**Крос-платформена среда –** Е система или продукт, който може да работи върху различни платформи или операционни среди.

**Среда за изпълнение –** Това е средата, в която се изпълнява програма или приложение. Може също да се определи и като хардуерната или софтуерната инфраструктура, която поддържа работата на определената кодова база.

**Node.js –** За работа със сървърната или Backend средата е използвана многоплатформената среда за изпълнение Node.js. Node.js позволява да се създават сървърни и мрежови приложения чрез интерпретируемия език за програмиране Javascript.

**Javascript –** Скриптов или интерпретируем език за програмиране, позволяващ да се създават интерактивни уеб приложения/страници.

**Скриптов език –** Език за програмиране, който се изпълнява от машината чрез помощта на интерпретатор

**Маркиращ език –** Представляват набор от специални инструкции наричани тагове, предназначени за формиране на структура в текстовите документи и определяне на отношенията между различните елементи на тази структура. Примери за маркиращи езици за XML и HTML.

**HTML –** Това е стандартният маркиращ език, който се използва за създаване и структуриране на съдържанието на уеб страниците в Интернет.

**CSS –** Език за стилизиране използван за описание на представянето на документ, написан на език за маркиране като HTML и XML.

**Tailwind CSS –** CSS софтуерна рамка, съдържаща масивна колекция от различни помощни CSSкласове за продуктивно и бързо реализиране на дизайни в уеб страниците.

**Express.JS –** Софтуерна рамка, която позволява изграждане на приложно програмни интерфейси и уеб сървъри в средата за изпълнение на Node.JS по минималистичен и гъвкав начин.

**MySQL –** Система за управление на релационна база данни.

**Система за управление на база данни (СУБД) -** Програмна система, чието предназначение е да създава и управлява данните организирани в базата данни.

**Данни –** Неорганизирани и непрецизирани факти.

**Информация**  - Обработени и организирани данни представени в смислен контекст.

**Абстракция –** Техника, която прикрива ненужни подробности и показва или имплементира само важните?

**ORM –** Обектно релационния картографер е техника или абстракция, която улеснява или опростява работата с релационна база данни в програмен език, който поддържа обектно ориентирано програмиране. Различните библиотеки или софтуерни рамки, които се основават на обектното релационно картографиране поддържат различни класове и методи, които позволяват извличане, създаване и актуализиране на данни без да използваме директно SQL езика в програмния код.

**Sequelize –** Sequelize обектно релационен картографер, който улеснява работата с релационни база данни в средата за изпълниение Node.JS.

**Модул –**  Модулите в JavaScript са файлове, които съдържат общ, свързан код.

**Vue –** Това е JavaScript софтуерна рамка за изграждане на потребителски интерфейси и едностранични приложения.

**Single Page Application (SPA) –** Едностраничните приложения са уеб приложения, които взаимодействат с потребителя чрез динамично пренаписване на текущата страница, без да се зарежда изцяло нова страница от сървъра.

**Парадигма на програмиране –** Стил на програмиране. Съществуват множество програмни парадигми, но основните сред тях са: обектно ориентирано, императивно, функционално и декларативно.

**Императивно програмиране –** Това е програмна парадигма, описваща КАК програмата трябва да направи нещо чрез изрично посочване на всяка инструкция (или израз) стъпка по стъпка, което да промени състоянието на програмата.

**Декларативно програмиране –** Това е програмна парадигма, описваща КАКВО прави програмата, без изрично да уточнява нейния контролен поток. С прости думи това е програмиране по абстрактен начин.

**NPM –** Мениджър на пакети (зависимости) в средата за изпълнение Node.JS. Състои се от команден интерфейс чрез, който можем да изпълняваме различни операции като стартиране на различни скриптове или изтегляне на зависимости и също притежава онлайн база данни с публични и частни пакети наречена регистър на npm.

**Пакет (Package) –** Терминът „пакет“ се използва за описание на код, който е публично достъпен. Един пакет може да се състои от един или много файлове.

**package.json** – Съдържа метаданни на проекта и всички експлоатирани зависимости от регистъра на npm.

**Метаданни** – Данни, който описват други данни.

**Bcrypt –** Node.js библиотека, за изпълнение на хеш върху пароли.

**Хеширане –** Изпълнение на комплексен математически алгоритъм, който преобразува входен поток от данни в случайно генериран изходен низ с фиксирана дължина, с цел да се скрие скрие оригиналният низ. За разлика от криптирането, хеширането е еднопосочен процес, тоест оригиналното състояние на низа не може да се възвърне след изпълнение на хешираща функция върху него.

**Криптиране -** Промяната на входни данни от формата на четлив текст в някакъв неподлежащ на разчитане вид. За разлика от хеширането, криптирането позволява да се възстанови оригиналния вид на криптираните данни чрез изпълнение на декриптиране. Декриптирането се осъществява чрез помощта на частни и публични ключове .

**Автоматизирано тестване –** Автоматизираното тестване е похват при изследването на софтуерни продукти, при които тестовете се разработват и изпълняват посредством вече създаден софтуер за тази цел (като софтуера може да е библиотека или софтуерна рамка). Тестване е процеса на проверка на качеството на разработен софутер.

**Unit Тест –** Метод за автоматизирано тестване, който проверява дали индивидуалните единици от сорс кода работят правилно.

**Интеграционен тест –** Тестовете на ниво интеграция имат за цел да проверят взаимовръзките и съвместната работа на различните юнити или компоненти, когато са поставени в една среда.

**End-to-end тест –** Тестването от край-до-край извършва изследване на всички компоненти в системата за проверка на това дали тя работи оптимално при симулиран реален сценарии

**Маршрутизиране –** Механизъм, който насочва заявките от клиента към крайните точки, които могат да се определят кат URI адреси, които съдържат пътя до кода, който обработва определената заявка.

**JWT –** JSON уеб жетоните се използват най-често за идентифициране на удостоверен потребител. Създадени са от сървър и се използват в комуникацията между клиентската страна и сървърната за достъп до защитени крайни точки в приложно програмен интерфейс. Те съдържат JSON обекти, които притежават важна информация, която е нужна да бъде обменена за да се изпълни оторизацията като всеки жетон е дигитално подписан използвайки избрани криптографски алгоритми, чрез които се гарантира, че оригиналното съдържание на жетона не може да бъде променено от клиента или злонамерена страна.

**Регулярни изрази** – Последователност от символи, които описват шаблони в даден текст. Тези шаблони от своя страна се използват предимно за търсене и заместване на поднизове в даден символен низ.

# Рецензия въпроси

**Хеширани ли са паролите във вашето приложение? Какъв алгоритъм е приложен?**

* Да, паролите за всеки регистриран потребител в системата са хеширани. Като накратко ще се опитам първо да обясня какво представлява хеширането и след това ще представя как е осъществено то в моята имплементация.
  + Какво е хеширане - Изпълнение на комплексен математически алгоритъм, който преобразува входен поток от данни в шифрован изходен низ с фиксирана дължина, с цел да се скрият чувствителни данни. За разлика от криптирането, хеширането е еднопосочен процес, тоест оригиналното състояние на низа не може да се възвърне след изпълнение на хешираща функция върху него.
  + Хеширащият алгоритъм или хеширащата функция, която е използвана в реализацията е **Bcrypt**. Като е инсталирана библиотеката **Bcrypt** за средата Node, от публичния регистър за пакети на **npm.** Библиотеката Bcrypt предоставя методи за

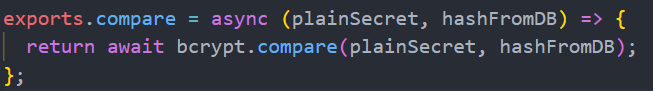
хеширане и също за сравнение на хеширана парола с такава, която е във вид на обикновен текст.

Методът за хеширане за извиква, когато се изпълнява регистрация на потребител. Един от недостатъците на хеширането е че даден вход връща един и същ хеширан резутат. Bcrypt обаче елиминира този проблем чрез техника, която прави хеша да е уникален, независимо от това дали е подаден един и същ низ, като тази техника се нарича подсоляване на хеша или salting преведено на английски. Подсоляването генерира случаен низ с фиксирана дължина, който се конкатенира към паролата, която ще бъде хеширана. При изпълнение на хеширащия алгоритъм, резултатът или хеша вече е уникален, дори когато подадения вход е един и същ.

Методът за сравнение compare(plainPassword, hash) се използва в системата, когато се осъществява удостоверяване на потребител.

Методът връща булеви стойности за истина или неистина при подадена валидна или грешна парола като самото стравнение се получава по следния начин.

1. Извличане на солта (salt) от хеша, който се намира в базата данни.
2. Конкатенация на извлечената сол – *(конкатенира се към нововъведената парола)*
3. *Хеширане на въведената парола заедно със солта*
4. *Върнат резултат (TRUE или FALSE) - (*ако пазената в база данни хеширана парола отговаря на тази, която е подадена от потребителя ще се върне истина и следователно, той ще получи достъп до останалите ресурси на системата.*)*

****

*Сравнение на парола във вид на обикновен текст със записа в база данни, който реално е хеш.*

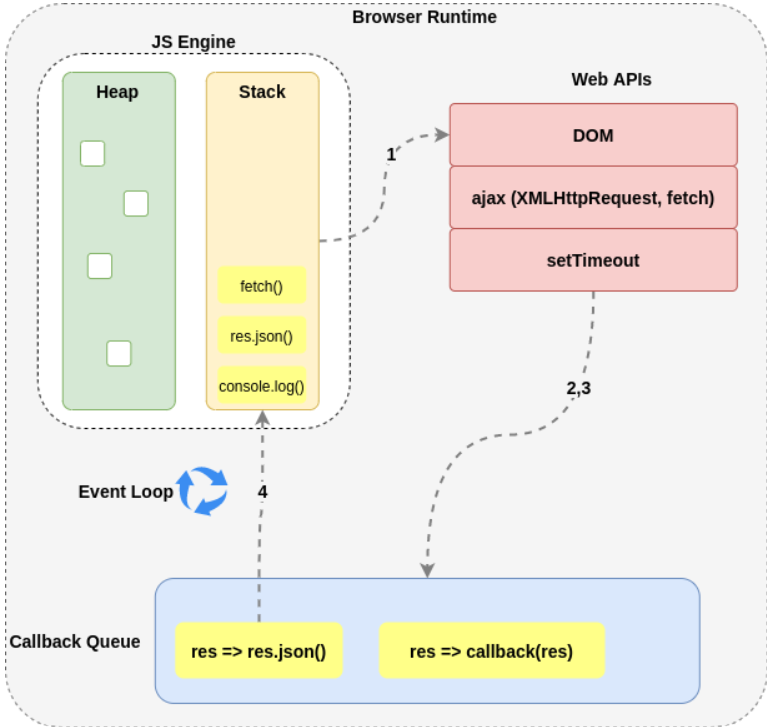
Картина, която съдържа текст

Описанието е генерирано автоматично

*Функция изпълняваща хеширането*

**Използвали сте асинхронни функции, каква е тяхната особеност?**

* Асинхронните функции в *JavaScript* позволяват на програмата да стартира потенциално дълго изпълняваща се задача(от която не е ясно кога ще е върнат резултат) и в същото време да може да отговаря на други събития, докато тази задача се изпълнява. Думата асинхронни е възможно да звучи подвеждащо, защото постигането на паралелизъм от JavaScript енджинът e невъзможно, тъй като JavaScript е език, който изпълнява програмата на една нишка, тоест той е еднонишков език и само един израз е възможно да се изпълни за даден момент. За да се избегнат недостатъците от това ***се използва*** Цикълът на събитията. Това e постоянно работещ процес намиращ се в средата на браузъра ( не е част от енджинът на Javascript), или ако се използва Node.JS се намира в C/C++ слоя на Node.JS, който позволява изпълнението на не блокиращи операции за вход и изход. Цикълът на събитията (Event Loop) разтоварва тези потенциално блокиращи операции(функции) върху C/C++ API в случая на Node.JS или към уеб приложно програмните интерфейси на браузъра в случай, когато работим само с Javascript (Front-end разработване).

****

JavaScript енджинът – съдържа heap, в който се осъществява разпределение или контрол на паметта.

Stack – Стек, чрез който се следи кой израз в javascript програма е на ред да се изпълни като работи на принципа LIFO – Last In, First Out или последен влязъл първи излязъл.

Извън Javascript Енджинът се намира средата на браузъра, където се намират Уеб приложно програмните интерфейси на браузъра (Web API), Опашката със задачи.

Web API – Контейнер, който съдържа допълнителни функционалности за JavaScript, като тук се намират също и всички приложни програмни интерфейси чрез който можем да осъществяваме асинхронни операции като например създаване на таймери, ХТТП заявки, създаване на слушатели на събития и тн. В този контейнер се прехвърлят всички изрази от програмата, която се изпълнява, които са с блокиращо състояние. При завършване на тяхното изпълнение, функциите, които трябва да се изпълнят след приключването на дългоизпълняващата операция се прехвърлят към опашката със задачи. Цикълът на събитията от своя страна проверява дали има синхрони изрази в стека на JavaScript Енджинът, ако няма той прехвърля съответната задача към стека и тя се изпълнява.

**Каква е разликата между валидиране и верифициране?**

* В контекста на софтуерното производство и изследване, валидацията и верификацията са независими процедури, които се използват заедно за проверка дали даден продукт, услуга или система отговаря на изискванията и спецификациите и че също изпълнява предназначението си. Основните разлики между тези два процеса са:
* Валидацията отговаря за проверката на крайният продукт за това дали той отговаря на истинските нужди и очаквания на крайният потребител. Тоест валидацията трябва да покаже, че програмата прави това, което потребителя желае.
* Верификацията е процес, който определя дали софтуерът е проектиран и разработен според определените изисквания. Верификацията трябва да покаже, че програмата работи коректно.

В контекста на работа с данни верификацията и валидацията имат различни значения в сравнение с областта на софтуерното изследване

* Валидиране на данни – свързано е с проверка на входните данни, за да се гарантира, че те съответстват на изискванията за данни на системата, за да се избегнат възникнали бъдещи грешки. Валидирането не засяга това дали въведените данни от потребител са напълно точни или верни.
* Верификация на данни – Процес, който извършва проверка, за да гарантира, че въведените данни съвпадат точно с оригиналния източник. Като верификацията на данни не отговаря за това дали данните са в коректен формат или в разумни граници. Това е работа на валидацията.

Примери за валидация:

Валидация на потребителско име на потребител чрез използването на регулярен израз:

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

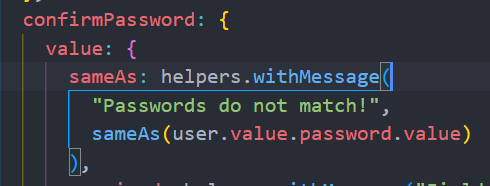
Резултат при въведени невалидни данни:

Картина, която съдържа текст

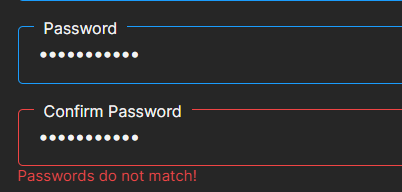
Описанието е генерирано автоматично

Пример за верификация:

Проверка на въведена парола



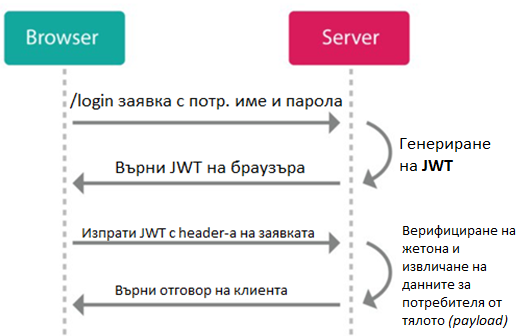
Резултат при верификация на парола:



**Използвали сте жетони за достъп в приложението си. Какъв е периода на валидността им?**

* + Какво са жетоните за достъп – Жетоните за достъп могат да се определят като обекти, които се използват за защитена трансмисия на претенции и данни между две страни за идентифициране на удостоверен потребител и да се разпознаят неговите роли и привилегии. Най-често тези две страни са клиент и сървър.
  + Съществуват различни видове жетони за достъп като в тази реализация се използват **JSON уеб жетони**.
  + JWT - Генерират се от сървъра и се използват в комуникацията между клиентската страна и сървърната за достъп до защитени крайни точки в уеб приложно програмения интерфейс. Те съдържат JSON обекти, които притежават важна информация, която е нужна да бъде обменена за да се изпълни оторизацията като всеки жетон е дигитално подписан използвайки избрани криптографски алгоритми, чрез които се гарантира, че оригиналното съдържание на жетона не може да бъде променено от клиента или злонамерена страна.

Жизнен цикъл на JSON уеб жетон:



* + Период на валидността им – В случай на откраднат жетон за достъп от злонамерено лице, което е извършило някаква уеб атака като (XSS и CRSF например) към клиентската страна на уеб системата, JSON уеб жетоните се задават с краткосрочно време на валидност като най често периода на валидността им продължава от 10 минути до половин час с цел да се намали периода от време в който злонамереното лице може да извършва заявки към защитените точки на сървъра с откраднатия жетон за достъп на потребител. Този краткосрочен период на валидност, обаче снижава потребителското изживяване, тъй като потребителят ще бъде подканван да се вписва в уеб системата по няколко пъти на час. Затова e честа практика JSON уеб жетоните за достъп е да се използват заедно с жетони за презареждане или (Refresh Tokens преведено на английски) като характерното за тях е че те се съхраняват в базата данни и са асоциирани с удостоверен потребител в системата и също са с по-дълъг период на валидност (от ден до седмица). Основната им цел е да презаредят изтекъл жетон за достъп (JWT) като се извършва заявка към сървъра за генериране на JSON уеб жетон…. като първо се извършва проверка за съществуващ запис на жетон за презареждане(refresh token) в базата данни, който отговаря на потребителя, който се нуждае от нов JWT. При намерен жетон за презареждане, сървърът генерира нов web token за достъп и го връща обратно на клиентската част на потребителя. В противен случай, ако такъв запис не съществува, потребителят е подканен да се впише наново в уеб системата.