Aleksandr Stepantsov

DEVELOPMENT OF A CENTRALIZED DATABASE BACKUP MANAGEMENT SYSTEM WITH NODE.JS AND REACT

Bachelor's thesis
Information Technology

2018

South-Eastern Finland
University of Applied Sciences

Author (authors)	Degree	Time
Aleksandr Stepantsov	Bachelor of Engineering	May 2018

Title

Development of a Centralized Database Backup Management System with Node.js and React

66 pages
0 pages of appendices

Commissioned by

South-Eastern Finland University of Applied Sciences

Supervisor

Matti Juutilainen

Abstract

The purpose of this thesis was to show the importance of data security for databases, explain the related challenges and risks and to provide a solution for the discovered problems. Such a solution is needed because databases are used to store many kinds of data and both individuals and businesses cannot afford the critical information to be completely lost.

The basics of cloud computing were described and it was shown how cloud solutions could be applied to the information security field. In the theoretical part, backup methods were studied for MySQL, PostgreSQL and MongoDB. Possible security measures and their implementations were also explored. The main tools that were used for building a solution were introduced.

Based on the study, a software solution allowing performing and managing database backups in a centralized manner was developed using Node.js and React. The entire process and the methods of development were explained. The application was built in a modular way, so it could be easily altered. As a result, a working solution was presented and tested. Possibilities for the further development of the application, such as support for the additional database engines and storage providers, were listed and described. This thesis was closely related to a huge number of other topics in the information security field and could be used for researching many of them.

Keywords

Node.js, JavaScript, database, backup, MongoDB, MySQL, PostgreSQL, React, web, RDBMS, cloud

CONTENTS

1 INTRODUCTION

We live in a world where information is an extremely valuable asset. Companies and individuals cannot afford to lose important data as such loss might cost them millions. Therefore, any enterprise with an IT infrastructure will eventually look into backup solutions.

There are various ways to backup the data. Cloud services are widely used for that purpose nowadays. However, there is a need for an automated solution with visualisation capabilities when it comes to an enterprise level.

A recent incident at GitLab.com is a great example of what could be caused by a simple negligence and insufficiency of backup procedures. An employee of GitLab Inc. had run a directory removal command on the wrong database cluster. The company had five database backup procedures set in place to be able to restore the data after such an emergency. However, none of them were reliably working. One of the employees of Gitlab Inc. happened to run a manual backup of that database cluster six hours prior to the incident. This coincidence allowed the company to restore most of the data. This emergency caused a few hours of downtime, loss of the data stored in the last six hours and probably cost the company a lot of money and customers. (GitLab Inc. 2017.)

The unfortunate situation explained above could have been prevented, if adequate backup procedures would have been in place. If such situation can occur even in a large company, many small businesses may be vulnerable, too. Any company should be prepared for incidents and there should be free comprehensive software solutions to provide help in achieving business continuity. The purpose of this work is to introduce available database backup methods, to explain how a centralized database backup management solution can be built and what tools can be used for the development.

I have been fascinated by software development since my childhood: building something new felt really interesting and important. I could easily forget about time and spend many hours in a row sitting and trying to solve a problem through programming. Therefore, I would like to apply my skills to the solution of this unexpectedly common problem.

2 DATABASES

Types of databases, database engines and differences between them will be discussed in this section. It is not a comprehensive overview of the information about databases, as only the topics relevant to the project will be covered.

2.1 Database types and differences between them

In general, two types of databases can be distinguished: relational (SQL) and non-relational (NoSQL). These are very different concepts that have their own benefits and drawbacks. Relational databases are often referred to as SQL databases because all of them use Structured Query Language to work with the data.

Relational databases have their roots in the 1970s and base on the relational model of data explained in a publication written by E. F. Codd. It was published in the Communications of the ACM magazine in 1970 (IBM 2018). In a relational model, the data is organized into tables that consist of columns and rows. Figure 1 shows a possible structure of the tables and relationships between them.

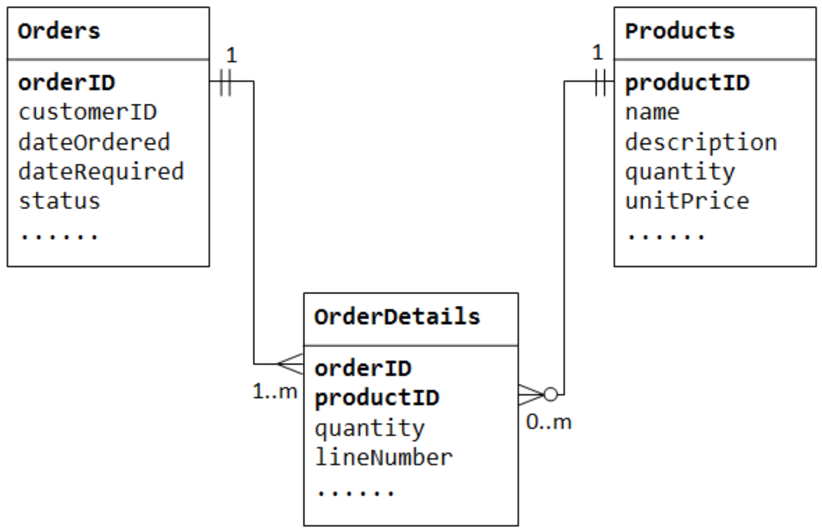


Figure 1. Relationships between tables in a relational database (ntu.edu.sg 2010)

Each row must be uniquely identified. Therefore, every table has a column called primary key which contains a unique value for each row in that table. Relations can be created between tables using foreign keys. For instance, a table can refer to another table by making one of its rows a foreign key. In this case, the foreign key will be the primary key of another table. (ntu.edu.sg 2010.)

Non-relational databases have existed since the late 1960s, but they weren’t very popular back in those days. Nowadays, the market started slowly moving towards the adoption of NoSQL databases, because they allow better scalability than normal relational databases. While there are many different types of non-relational databases, they have one thing in common: they can efficiently work with unstructured data, which is a huge advantage. (Leavitt 2010.)

2.2 Relational databases

This section will provide a brief overview of certain details related to relational databases.

2.2.1 Structured Query Language

Structured Query Language (SQL) is a language designed to access or manipulate data in a relational database management system. SQL was developed in the 1970s by two employees of IBM: Donald D. Chamberlin and Raymond F. Boyce. It was initially created for use with IBM’s System R, a relational database management system. SQL was officially standardized by ANSI and ISO in 1986. (Chamberlin 2012.)

Data retrieval and manipulation is done via statements that may contain multiple clauses. SQL supports predicates which are evaluated with three-valued logic. They are used to retrieve only a specific set of data that meets a provided condition. The three-valued logic has three truth values: true, false and unknown. (Chamberlin 2012.)

Several subsets can be distinguished within SQL:

- Data Manipulation Language (DML) allows retrieving and altering data in a table using such statements as `SELECT`, `INSERT`, `UPDATE`, `DELETE` and more.
- Data Definition Language (DDL) allows defining and altering data structures using such statements as `CREATE`, `ALTER` and `DROP`.
- Data Control Language (DCL) allows setting access rights in a database using such statements as `GRANT` and `REVOKE`.

These three subsets form the SQL language itself. (Lifewire 2017.)

2.2.2 MySQL

According to DB-Engines Ranking by solid IT (2018), MySQL is one of the most popular relational database management systems (RDBMS) in the world. It is currently owned by Oracle Corporation which is a parent company of MySQL AB. MySQL was initially released in 1995 by a Swedish company MySQL AB. There is a completely open-source version of it called MySQL Community Server which is released under the GNU General Public License and a commercial version, MySQL Enterprise Server. MySQL is well-documented and has a large community of supporters. (Data Science Central 2018.)

MySQL was written in C and C++ and officially supports many platforms, including various versions of Windows and popular distributions of Linux (Black Duck Software, Inc. 2018). Some of the reasons why it became so popular include high reliability, capabilities for scaling and security.

2.2.3 PostgreSQL

PostgreSQL is a widely used object-relational database management system (ORDBMS) written in the C language. The main difference from a normal RDBMS is that it has an object-oriented model. It is completely free and open-source under the PostgreSQL License. PostgreSQL is cross-platform and supports both Microsoft Windows and Unix-like systems. (The PostgreSQL Global Development Group 2018.)

PostgreSQL was developed as a successor of the Ingres project at the University of California and its development started in 1985 under the name of POSTGRES.

This project was ended in 1994 and released under the MIT license. In 1994, two students from Berkeley created Postgres95 which was based on POSTGRES but used the SQL query language interpreter instead of the POSTQUEL one. The project was renamed to PostgreSQL and its initial production-ready version was released in 1996. PostgreSQL is currently being maintained by PostgreSQL Global Development Group which consists of many people and companies. (The PostgreSQL Global Development Group 2018.)

Many people choose PostgreSQL over its competitors because it is open-source, supported by a diverse community, follows ACID principles and supports various replication methods (The PostgreSQL Global Development Group 2018). It is also one of the most advanced relational database management systems in terms of features. However, it is much less popular than MySQL and Microsoft SQL Server based on DB-Engines Ranking by solid IT (2018).

2.2.4 Other relational databases

There are many other relational database management systems that weren’t described above, but still have to be mentioned:

- Microsoft SQL Server – a popular RDBMS by Microsoft
- Oracle – the most popular RDBMS in the world as of April 2018 based on DB-Engines Ranking by solid IT (2018)
- SQLite – a local RDBMS that is embedded into an application
- MariaDB – a fork of MySQL made by its own creators
- Firebird – a RDBMS maintained by Firebird Project

These RDBMSs are widely used but will not be used for the implementation of this project. (solid IT 2018.)

2.3 Non-relational databases

Non-relational databases have started gaining popularity recently. Following Makris (2016), they can be classified into four categories:

- Key-value stores: Each value has a corresponding key (Redis, MemCached, Dynamo).
- Document stores: These consist of documents that represent compressed key-value pairs (MongoDB, CouchDB).

- Column family stores: These represent a sorted multidimensional map that allows to store key-value pairs (PNUTS, Cassandra).
- Graph stores: Data is stored within the edges, nodes and properties of a graph structure (Neo4j, HyperGraphDB).

However, the classification of NoSQL databases is not limited to these categories and each of them may be further divided into subcategories.

MongoDB document store will be used during the implementation phase of this project. Therefore, certain details about it will be provided next. MongoDB is a highly scalable and flexible NoSQL database engine. It is the most popular document store in the world based on DB-Engines Ranking by solid IT (2018). The development of MongoDB was started in 2007 by 10gen software. The name of the company was changed to MongoDB Inc. in 2013. Initially, it was planned to release the product using the Platform as a Service model. However, the company leaned towards the open source model in 2009. (ByteScout 2014.)

MongoDB stores data in JSON-like documents, so fields can be different in each document and it is not limited to a single model like RDBMSs. The document model can be mapped to objects in the application code. MongoDB has its own query language, which looks similar to JavaScript. Also, many advanced features are supported, such as aggregation, indexing and ad hoc queries. There are many different tools for each programming language that make it easy to work with MongoDB. (MongoDB, Inc. 2018.)

3 DATABASE BACKUP METHODS

This section will explain backup methods available for the databases covered in the project.

3.1 MySQL

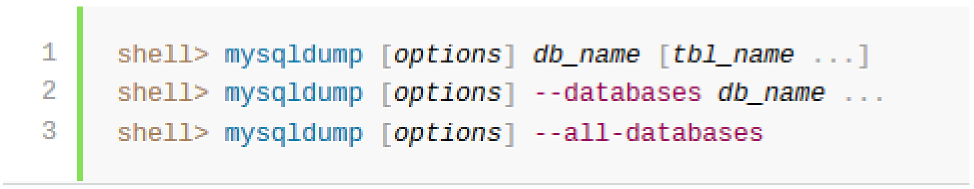
According to Oracle Corporation’s documentation (2018), the following backup methods are available for MySQL:

1) Backup using MySQL Enterprise Backup product (MySQL Enterprise Edition only)

This method allows performing physical backups of both entire database and selected tables. MySQL Enterprise Backup supports compressed and incremental backups. A huge advantage of this method is that restoring is much faster. Hot and warm backup methods are supported. Hot backup means that data can be modified while a backup is being performed. However, this backup technique is only supported if the InnoDB storage engine is used. Otherwise, a warm backup is performed which means that tables cannot be modified during the backup process.

2) Backup using mysqldump utility

This method provides capabilities for performing a logical backup of either entire database or selected tables. Logical backup means that generated output is a set of statements in SQL language that can be used to fully reproduce the database. It is possible to perform a hot backup for the InnoDB tables with this tool. Mysqldump is a command line utility. There are three main ways to invoke it as can be seen in the Figure 2.



```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
shell> mysqldump [options] --all-databases
```

Figure 2. Invocation syntax of mysqldump (Oracle Corporation 2018)

The first way is used to backup certain tables from one database, the second way is used to backup multiple databases and the third one is used to backup all databases. The `mysqldump` command also accepts many options. Only the options that will be used during the development of this project are going to be mentioned here. The list below shows these options:

- `--host`
  IP address or hostname of the destination server.
- `--port`
  Port where the destination server is running.
- `--user`
  Username for connecting to the MySQL server.
- `--password`
  Password for connecting to the MySQL server.
- `--all-databases`
  Backup all tables.
- `--databases`
  Allows to specify a list of tables to be backed up.
- `--single-transaction`
  Performs a backup in a single transaction.

Restoration of a backup is later possible by injecting a backup file into the database.

3) Backup by copying the storage engine files

MySQL Server must be stopped for a backup to be consistent (or a lock must be placed for read operations). InnoDB tables cannot be backed up using this method.

4) Delimited-text file backup

   There is a MySQL query which allows outputting the table's data to a file. This method is not very practical and output can only be saved on the server's host.

5) Incremental backup using binary log

6) Backup using replication slaves

   By using this method, data is replicated to the other server and then backed up from there (to avoid performance problems).

7) Backup by making a file system snapshot

The second method from the list above will be used in the practical part of this thesis. The other methods are mentioned but not explained in details for this reason.

3.2 PostgreSQL

There are three different approaches for backing up the data in PostgreSQL database. According to the The PostgreSQL Global Development Group's documentation (2018), the following methods are available:

1) Backup using pg_dump utility

   This utility allows performing a logical backup of PostgreSQL database (just like mysqldump for MySQL). It is possible to perform hot backups. This tool can be invoked in the following way:

   ```
   pg_dump [option…] [dbname]
   ```

   `pg_dump` command accepts many options, most important ones are listed below:

   - `--host`
     IP address or hostname of the destination server.
   - `--port`
     Port where the destination server is running.
   - `--username`
     Username for connecting to the PostgreSQL server.
   - `--password`
     Password for connecting to the PostgreSQL server.

- `--table`
  Specifies a table to be backed up, may be provided multiple times.
- `--dbname`
  Specifies a database to be backed up, may be provided multiple times.
- `--format`
  Specifies format of the output. Allowed values: `plain`, `custom` and `tar`.
- `--single-transaction`
  Performs backup in a single transaction.

   pg_restore tool can be used to later restore the database. It accepts all the parameters that were explained for pg_dump above and has the following syntax:

   ```
   pg_restore [option…] [filename]
   ```

2) File system level backup by making a file system snapshot

3) Continuous archiving (similar to the binary log in MySQL)

The first method will be used for the implementation of this project. The other methods are mentioned but not explained in details for this reason.

3.3 MongoDB

According to the documentation provided by MongoDB, Inc. (2018), the following backup methods are available for MongoDB:

1) Direct copying of the files

2) Backup by making a file system snapshot

3) Backup using mongodump utility

   This utility creates a binary export of the data stored in a database. It has the following invocation syntax:

   ```
   mongodump [options]
   ```

   There are many options available and the list below provides an overview of the most important ones:

   - `--uri`
     This option is used to specify a connection string for the MongoDB database. This string has the following format:

     ```
     mongodb://[username:password@]host1[:port1][,host2[:port2],...
     [,hostN[:portN]]][/[database][?options]]
     ```

- `--gzip`
  Compresses the output file(s).
- `--archive`
  Used to write the output to a single archive. Option `--out` allows to specify output as well but writes everything to separate files.

The mongodump utility will be used in the practical part of this thesis. The rest of the methods are not explained in details for this reason.

4 CLOUD SOLUTIONS

This section will give a brief overview of cloud technologies and, in particular, cloud-based storages. They will be widely used in the practical part of this project.

4.1 Overview

Certain initial concepts of what is currently called “cloud computing” were developed as early as in the 1960s. Back then, it was called Remote Job Entry (RJE). Tasks were sent to the mainframe computers remotely and then the output could be retrieved. In the 1970s many time-sharing solutions became available such as Multics and Cambridge CTSS. However, this approach wasn’t very popular. A lot of research related to time-sharing technologies was conducted in 1990s. However, the actual cloud computing only became available in 2006 when Amazon launched its Elastic Compute Cloud. Later on, more companies released their solutions: Google App Engine became available in 2008, Microsoft Azure came into existence in 2010. (Mourghen, 2018.)

Cloud solutions have many advantages over traditional approaches. Responsibility for the management and maintenance of the infrastructure lies with the cloud provider. It means that businesses can focus on the development of their own products and services instead of wasting time and resources on building an on-premises IT infrastructure. Moreover, cloud solutions provide high availability and have presence across multiple regions. If data center in one region will fail, then other data centers can temporarily compensate for the lost capacity. Cloud products also have outstanding scaling capabilities. It allows to use just as much capacity as needed on-demand and only pay for what is actually used. (Amazon Web Services 2018.)

In general, three types of cloud computing service models can be distinguished:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Each of these models has its own benefits and use cases. IaaS is the lowest-level solution out of three. Customer manages everything except for the hardware itself. Amazon EC2 is a good example of a service that uses such a model. Many IaaS providers also offer a library of images to make setup process easier for a customer. PaaS delivers an entire computing platform which may include operating system, development environment, databases and web servers. Users simply upload their applications to the platform and run them. However, there is less control from the user than in IaaS model. Google App Engine utilizes this approach. The last model is SaaS and it is very high-level. A user can subscribe to receive the access to a service offered by a provider. Everything is managed by the provider, user cannot manage underlying infrastructure or platform. This approach is widely used nowadays. (Amazon Web Services 2018.)

4.2 Cloud-based storages

Cloud-based storage is one of the services typically available from most cloud providers. This data storage model implies that users can upload their data into the logical pools without access to the underlying physical layer where the data is actually stored. It is widely adopted by many large companies. For instance, Netflix uses Amazon S3 for streaming the movies to their customers. There is no need to think about the security of the data yourself and the data can be replicated across many regions to ensure its safety. Moreover, storage space in object storage services, such as Amazon S3 and Google Cloud Storage, is usually billed on usage. While object storages are more business-oriented, there are many normal file storage services, too. Two object-oriented storage services will be covered in my work: Amazon S3 and Google Cloud Storage. (Amazon Web Services 2018.)

4.2.1 Amazon S3

Amazon S3 is an object storage service by Amazon Web Services. S3 is an abbreviation for Simple Storage Service. It was officially launched in 2006 and quickly gained high popularity. There are two interfaces available for creating, retrieving and managing the objects: REST (Representational State Transfer) and SOAP (Simple Object Access Control). Additionally, an object can be retrieved using BitTorrent protocol or HTTP GET interface. There is also a web interface available for manual object management. Objects are stored in user-created “buckets”. Each bucket must be given a unique name that will be used to identify it. (Amazon Web Services 2018.)

Amazon S3 offers three different storage types:

1) Amazon S3 Standard (S3 Standard)
2) Amazon S3 Standard-Infrequent Access (S3 Standard-IA)
3) Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA)

S3 Standard is designed to provide high levels of availability and durability for the frequently accessed data. Data is stored in three Availability Zones simultaneously and it is guaranteed to be safe even in case of destruction of one Availability Zone. S3 Standard-IA is designed for storing the infrequently accessed data. It has the same durability as S3 Standard but lower availability. Another difference is that S3 Standard-IA has minimum capacity (128KB) and minimum storage duration charge (30 days) set for the objects. Storage price and retrieval fee are reduced for this type of storage compared to the S3 Standard. S3 One Zone-IA is similar to the S3 Standard-IA, but data is only stored in one Availability Zone which results in a decreased redundancy. (Amazon Web Services 2018.)

Amazon S3 has built-in encryption capabilities with both automatically assigned encryption keys and custom encryption keys provided by a user. This feature removes the need for manual encryption and saves time and resources. Additionally, data transfer can also be encrypted by using endpoints with a SSL (Secure Socket Layer) encryption. (Amazon Web Services 2018.)

4.2.2 Google Cloud Storage

Google Cloud Storage is a direct competitor of Amazon S3 launched in 2010. It is an object storage service by Google aimed at enterprises. It follows the same principles as Amazon S3: objects are stored in buckets that have unique keys assigned by a user. Data can be stored, retrieved and managed using the REST API.

There are four storage classes available:

1) Multi-Regional
2) Regional
3) Nearline
4) Coldline

Multi-Regional storage is designed for the frequently accessed data that requires additional redundancy achieved by replicating the data across multiple regions. Regional storage is similar to the Multi-Regional one, but the data is stored in a single region. Nearline storage is designed for the data accessed less than once a month. Coldline storage is created for the data accessed less than once a year. (Google LLC 2018.)

5 SECURITY

Security should be a key consideration in any project. It is unacceptable to have data compromised or corrupted. Moreover, nowadays there are more dangers related to information security than ever. General information about common security measures will be given in this section.

5.1 Encryption

Encryption is a security measure aimed at the prevention of access by unauthorized parties through information encoding. Encryption itself doesn’t prevent the encoded data from being intercepted. However, this data will be useless unless decrypted with a valid key. In theory, any piece of encoded data can be decrypted without a key with enough computational resources. However, there is no such computational power currently available to crack the modern algorithms that are considered secure. There are two types of encryption: symmetric-key and public-key.

In symmetric-key method, the same shared key is used for both encryption and decryption. It means that if the encryption key is compromised, then the data can also be decrypted with it. However, this approach provides a couple of advantages: symmetric-key algorithms have a high protection level as long as the key is safe and they are inexpensive to process. The main problem of this method is in exchange of the shared key as it must be known by both parties. It can be solved with the use of the Diffie-Hellman key exchange algorithm. Implementations of symmetric-key algorithms include but are not limited to AES, Blowfish, 3DES. Figure 3 shows how symmetric-key encryption works. (IBM 2018.)

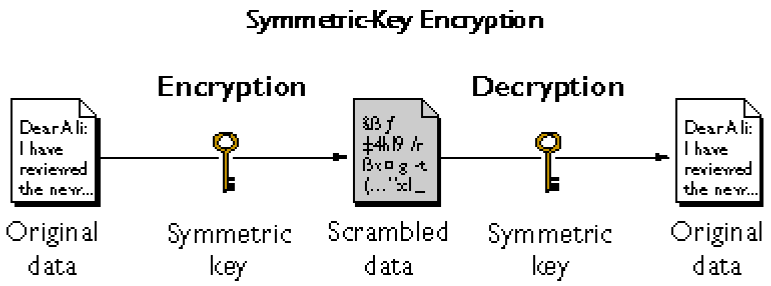


Figure 3. Symmetric-key encryption (IBM 2018)

Public-key encryption utilizes two keys: public and private. A public key is known to everyone and a private key is secret. If data was encrypted with a public key, then it will only be possible to decrypt it with a private one. It also works the other way around. Public key algorithms require much more computational resources than symmetric-key algorithms. Most implementations of such algorithms are based on Rivest-Shamir-Adelman (RSA) cryptosystem created in 1978. Figure 4 describes the principle used for public-key encryption. (IBM 2018.)

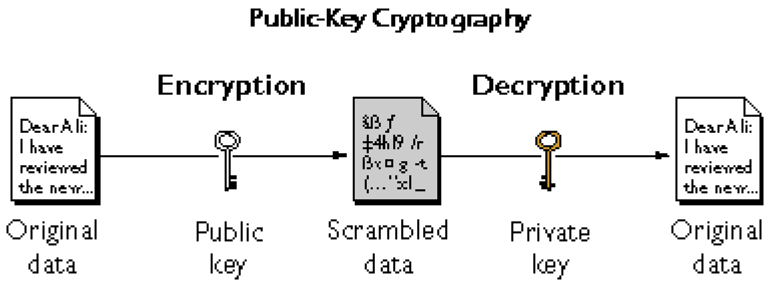


Figure 4. Public-key encryption (IBM 2018)

Choice of the encryption type mostly depends on what it will be used for. However, symmetric-key encryption is usually preferred, unless there is no other choice but to use public key encryption, due to performance considerations. While encryption is not directly used for backups in the practical part of this thesis, it is important to understand the main concepts. For instance, Amazon S3 provides built-in encryption capabilities for uploaded objects.

5.2 Hash functions

Hash functions allow to convert data of random size to data of a fixed size. Result of such conversion is a hash value. Original data cannot be reconstructed from a hash value. However, there are some exceptions when it comes to outdated hashing algorithms. Hash functions are widely used for verification of data integrity, message authentication and digital signatures. (SANS Technology Institute 2018.)

However, two different pieces of data may sometimes produce the same hash value and form a collision. Perfect hash function means that such a collision cannot occur because hash function will map every input to a unique hash value. In practice, it is impossible to achieve such a result for a random input. Therefore, a collision will inevitably occur after some time. The probability of this event may be reduced by using stronger hashing algorithms. (SANS Technology Institute 2018.)

Certain widely used hash functions are listed below:

1) MD5 is a 128-bit hash function that is no longer considered secure. Many vulnerabilities have been found over the years. This hash function is not

resistant to collision attacks. However, MD5 is still being used for the data integrity verification.

2) SHA-1 is a 160-bit hash function that is not considered secure since 2005. Collisions have been found in 2017. However, it is still being used for the data integrity verification.
3) SHA-2 is a family of 4 hash functions: SHA-224, SHA-256, SHA-384 and SHA-512. Each of them produces an output of 224, 256, 384 and 512 bits respectively. They are widely used and considered secure.
4) SHA-3 is a relatively new family of hash functions which is considered secure. However, it is not widely used yet due to the fact that SHA-2 is still secure.

Some of the outdated hash functions from the list above are still being used for the data integrity verification, because a collision is not very likely and these older functions are also less expensive in terms of consumption of computational resources. However, they should never be used for hashing critical information or passwords. Passwords should always be hashed with a strong algorithm and never be stored in plain text.

5.3 Security risks in web development

Security vulnerabilities are very common even in large projects. These flaws may affect the most critical systems even in such fields as medicine and finance. There are many well-known vulnerabilities that are still often present in modern applications. New security risks are being constantly discovered and a lot of them are still waiting to be found. Most common and publicly known vulnerabilities can be avoided by building an application using the best practices and continuous testing during the development process. (OWASP 2018.)

Certain common security risks are listed below:

1) Injection
Possibility for SQL and NoSQL injections is a serious problem. The goal of this attack is to get access to the database or corrupt it. Such attacks can be mitigated by sanitizing user input and using escaping techniques. According to the OWASP's research, it was the most critical vulnerability in 2017. (OWASP 2018.)
2) Cross-site scripting (XSS)
XSS allows an attacker to execute any JavaScript code in the victim's browser by adding a piece of code to a web page through some unsecure data input control. Filtering and escaping of raw input can be used to prevent that. (OWASP 2018.)
3) Cross-site request forgery (CSRF)

> CSRF is a commonly used technique for performing actions in an application where user is already authenticated. Third-party website may contain the code that will send a request to the web application. Functionality of a vulnerable application can be accessed by an attacker. (OWASP 2018.)

These three risks should be avoided in every web application. However, this list is not comprehensive as there are many other security risks. They are usually obvious and can be easily avoided simply by having a good application architecture. Such risks include broken authentication, broken access control, security misconfiguration. There are tools and services available that allow to scan for security vulnerabilities. (OWASP 2018.)

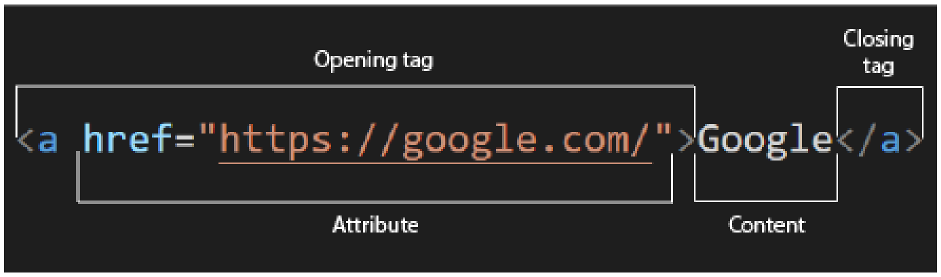
6 DEVELOPMENT TOOLS AND SERVICES

The purpose of this section is to give an overview of the main technologies, frameworks, tools and services that weren't mentioned in the previous chapters and will be used in the practical implementation of the project.

6.1 HTML

HTML (Hypertext Markup Language) is a markup language which is used to define the structure of a web page semantically. This markup is later retrieved from a web server and rendered by a web browser. It is the most important technology is web development. HTML is currently maintained by World Wide Web Consortium (W3C).

The language implements a series of tags that can be used to wrap different parts of the content in order to change its representation. HTML tags also accept attributes that are used for customization. Most elements consist of opening and closing tags. However, some self-closing tags are also available. Figure 5 shows a structure of an HTML tag. (Mozilla Corporation 2018.)



```html
<a href="https://google.com/">Google</a>
```

Figure 5. Structure of an HTML element

There were many versions of HTML specification created over the years. The history of HTML begins in 1993 when it was initially released (first specification). The latest version up to date is HTML5. It includes many new features which are absolutely necessary in the modern world, such as `<video>` and `<audio>` tags that allow embedding multimedia content into a web page natively. HTML5 is adopted by the latest versions of all modern web browsers. (W3C 2017.)

6.2 CSS

CSS (Cascading Style Sheets) is the second core technology in web development, and closely related to HTML. It allows to customize the style of HTML elements such as colors, fonts, and layout. CSS separates content of an HTML document from its presentation. CSS also makes it easier to maintain web pages. The latest specification is CSS3 and it is supported by all modern browsers. (W3C 2018.)

Before CSS was taken into use, all styling was applied to HTML elements through attributes. Therefore, styling options were very limited, and it was hard to find anything. Everything changed after the initial release and standardization of CSS (first specification) in 1993. This version only provided basic customization capabilities and it greatly differs from the latest CSS3 standard.

CSS represents a simple markup language. CSS stylesheet consists of a set of rules. Every rule includes one or more selectors (tag, class or id) and a declaration block itself. Declaration block is a set of properties with assigned values (declarations). Figure 6 gives a good overview of the structure of a CSS rule. (Wordpress.com 2018.)

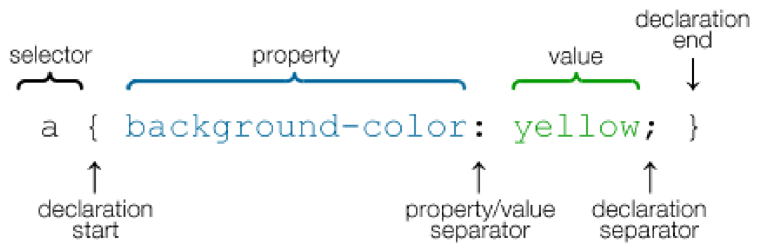


Figure 6. Structure of a CSS rule (Wordpress.com 2018)

CSS3 also provides strong capabilities for animation. It is now possible to implement many kinds of animations using pure CSS without JavaScript which makes developers’ life easier.

6.3 JavaScript

JavaScript is a dynamic, prototype-based, multi-paradigm, interpreted and an object-oriented programming language (Mozilla Corporation 2018). It is one of the core technologies of World Wide Web content production. JavaScript is usually used to make webpages more interactive and provide additional user experiences such as animations. Most websites use this technology, and all modern web browsers provide native support for it. There are many JavaScript engines (implementations). Each of them is based on ECMAScript specification, but some engines implement only part of the specifications and others have features beyond ECMA. (ECMAScript 2007.)

Event-driven, functional and imperative (including object-oriented and prototype-based) programming styles are supported by JavaScript. There are many advanced built-in features. JavaScript is often used in web development for the Document Object Model (DOM) manipulation. JavaScript engines were initially only implemented client-side in web browsers, but they are now embedded in many types of host software such as web servers and databases, and in non-web programs. (Mozilla Corporation 2018.)

It is important to note that JavaScript and Java are totally different languages (in terms of design). There are many different frameworks/tools available for JavaScript to make development easier: Node.js, React.js, Angular.js and others.

6.4 Node.js and Express

Node.js is an asynchronous event-driven JavaScript runtime which allows to build network applications at scale easily. In Node.js process never gets blocked as it is designed without threads. Therefore, it is a great choice for scalable systems. Moreover, Node.js provides many useful default libraries, such as HTTP which allows to create a webserver with just 2 lines of code. Nowadays, Node.js became very popular for backend programming in web. (Node.js Foundation 2018.)

Node.js provides the following command syntax for launching an application:

```
node [options] [V8 options] [script.js | -e "script" | - ] [arguments]
```

While most of the parameters are self-explanatory, `V8 options` requires further explanation. V8 is a JavaScript engine used by Node.js to run the code. Therefore, `V8 options` parameter specifies options for that engine.

According to npm, Inc. (2018), npm is a package manager for JavaScript which is frequently used with Node.js. There are thousands of packages available for use and they can be easily installed with npm. Application’s configuration for npm is stored in a package.json file. This file may contain such information as application name, author’s name, license, version number, description, homepage, entry point, scripts and dependencies. The following command line syntax is being used by npm:

```
npm <command> [args]
```

Most commonly used commands for npm are listed below:

- `init`: allows to create a package.json file interactively.
- `install`: installs specified packages. If no arguments are provided, then it will install all the packages listed as dependencies in a package.json file.
- `uninstall`: uninstalls specified packages.
- `run`: runs a specified script.
- `start`: executes a start script.

It is important to mention a few arguments for the `install` and `uninstall` commands:

- `-g` or `--global`: installs/uninstalls a package globally.
- `--save`: saves a package as a dependency or removes it. It will be added/removed in `dependencies` section of a package.json file.
- `--save-dev`: saves a package as a development dependency or removes it. It will be added/removed in `devDependencies` section of a package.json file.

Express is one of the most commonly used web frameworks for Node.js. It has many essential features such as routing, support for rendering engines and it is highly flexible with custom middlewares. Use of Express makes it easier for the developers to create backend and organize the code.

6.5 React, Redux and Blueprint

React is a framework for JavaScript that allows to easily build user interfaces. It is currently maintained (and also utilized) by such companies as Facebook and Instagram. MVC (Model-View-Controller) pattern is utilized in React.
Simply said, React provides capabilities to create UI for highly scalable and complex web applications that can change content over time without the reload of the page. It manipulates DOM in order to update the page's content. Browser's DOM is updated very efficiently as React computes the differences between an old data structure and a new one before updating the actual DOM. This feature is called Virtual DOM. (Krill 2018.)

State is an important concept in React. When the state is updated, React will re-render the component using the new state. The state is supposed to be immutable. React is often used together with Redux which allows to store the state of the whole application in an object tree within a single store. The use of Redux makes development significantly easier, reduces the amount of code that has to be written and makes the code less cluttered. There are three main concepts in Redux:

1) Actions: objects that are required to update the state in the store from the application. Action creators are functions that return actions. Actions can be easily dispatched using `dispatch` method. (Abramov 2018.)

2) Reducers: functions that change the state after the action is sent to the store (Abramov 2018).
3) Store: an object where an entire application's state is stored. Plus, it provides a couple of methods for manipulations with the store, such as `getState`, `dispatch` and `subscribe`. (Abramov 2018.)

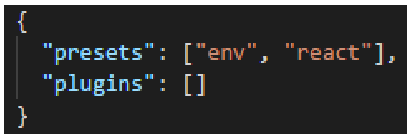
These concepts form the core of Redux.

Another useful JavaScript library is called Blueprint. It is a React-based toolkit for UI development. Blueprint contains many commonly used components for React and provides a huge amount of customization options. React will be used in combination with Blueprint components during the implementation stage of this project.

6.6 Babel and Webpack

There are many versions of ECMAScript (ES) standard which forms the core of JavaScript. The latest versions make development significantly easier by adding new features. However, developers of modern web browsers are slow at adopting these standards. The latest version of ECMAScript standard is ES8. However, only ES5 is fully supported by all modern browsers at the moment. Babel makes it possible to use newer specifications even in browsers that don't support them by transpiling the code to an older specification. Moreover, Babel supports presets and plugins that allow to perform custom manipulations with the code during the transpiling process.

Plugins and presets can be added through .babelrc file. Figure 7 shows how this file looks.



```
{
  "presets": ["env", "react"],
  "plugins": []
}
```

Figure 7. An example of .babelrc file contents

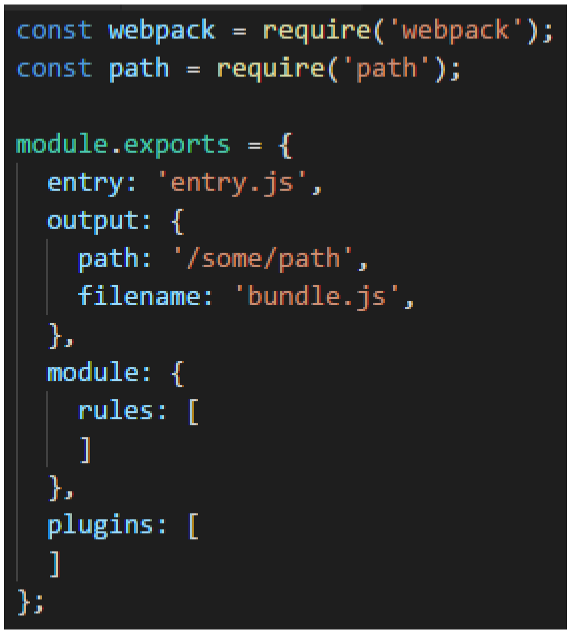
There are two presets and no plugins added in the example shown in the figure above. They will be used during the transpiling process later. The following command can be used to start the transpiling:

```
babel src -d dest
```

`src` is the directory where files to be transpiled are located and `dest` is a destination directory.

Another useful tool for the JavaScript development is Webpack. It is a module bundler for JavaScript. It means that Webpack processes all the files and dependencies of an application and combines everything into one or more bundles. It is an extremely useful tool that allows to include just one file into a web page instead of having to worry about all the files and dependencies. Custom plugins and loaders are also supported. Loaders allow Webpack to process more types of files than just JavaScript. (webpack.js.org 2018.)

Webpack's configuration is stored in a webpack.config.js file. It allows to specify the entry point and output filename, add loaders/plugins and more. Figure 8 shows the general structure of a Webpack configuration file.



```
const webpack = require('webpack');
const path = require('path');

module.exports = {
  entry: 'entry.js',
  output: {
    path: '/some/path',
    filename: 'bundle.js',
  },
  module: {
    rules: [
    ]
  },
  plugins: [
  ]
};
```

Figure 8. Structure of a Webpack configuration file

In the figure above, `entry` is the entry point for Webpack, `output` is an object that contains `path` and `filename` properties that define where the bundle will be saved.

There are two arrays: `rules` and `plugins`. Each of these arrays may contain objects that define rules for the loaders or plugins to be used during the bundling.

7 IMPLEMENTATION

The project's implementation was the most challenging aspect of this thesis. A web application had to be built that would satisfy basic requirements for a database backup management system. Development process is explained in this section.

7.1 Goals

It is very important for any project to have the goals defined early in development. The main goal of this project was to create an extensive web-based database backup management system. The application was created with Node.js and React.

The following features were implemented:

- Authentication
- Database management
- Scheduler
- Backup monitoring
- Manual backups
- Data integrity verification
- Compression
- Modularity
- Upload to cloud services

Backup processing for MySQL, PostgreSQL and MongoDB was supported in this project. Amazon S3 and Google Cloud Storage were used as cloud storages.

7.2 Preparing the environment

Visual Studio Code was used for the development of this project. It is a free, cross-platform and open source code editor developed by Microsoft. There are many useful plugins available for it to speed up the development. When it comes to choosing a hosting solution, Amazon Web Services (AWS) offers a great amount of flexibility and even includes a year of free use for a t2.micro computing instance. An Amazon EC2 instance was used for hosting a web server and Amazon S3 was used as a remote storage for backups.

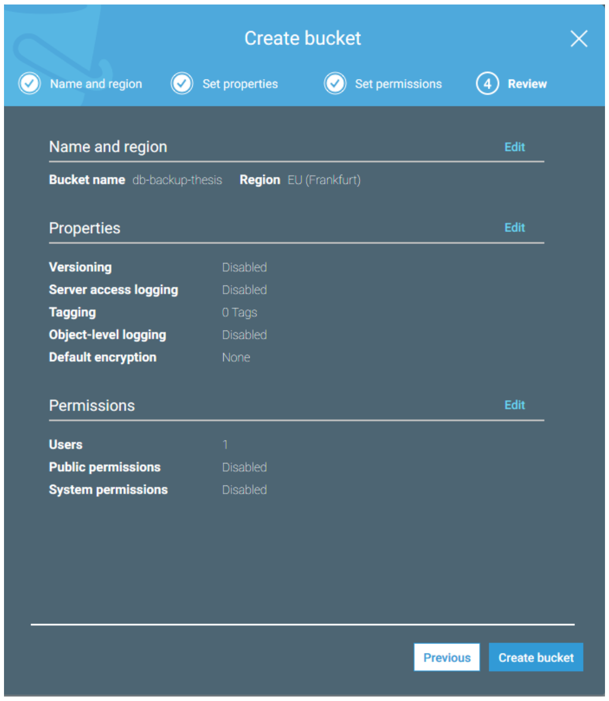
There was one t2.micro instance with a custom security group and Elastic IP assigned. This security group had all outbound connections allowed and inbound connections allowed on TCP ports 22 (SSH), 80 (HTTP) and 443 (HTTPS). Elastic IP is a static IP address that can be assigned to a resource on AWS. Ubuntu 16.04 was chosen as an operating system for the computing instance. An SSD volume with 30 GB of space was attached to the instance. A key pair was generated for authentication.

The following software was installed on the instance:

- Node.js 8.11.1
- npm 5.6.0
- nginx 1.10.3
- certbot 0.22.2
- mysqldump 10.13 (part of a `mysql-client` package)
- mongodump r3.6.4 (part of a `mongodb-org-tools` package)
- pg_dumpall 10.3 (part of a `postgresql-client-10` package)

All of these packages were installed using an APT package management tool. Some of the tools listed above require a description. Nginx is a web server and a reverse proxy. Mainly its reverse proxy functionality was used in the project. Certbot is a very useful tool for deploying SSL certificates from Let’s Encrypt. Let’s Encrypt is a free certificate authority. This tool also automatically configures the web server for use with a new certificate. The last three tools are used for performing backups on MySQL, MongoDB and PostgreSQL databases respectively.

The next step was to setup the cloud storage and to obtain credentials for programmatic access. A bucket called db-backup-thesis was created on the Amazon S3 platform. Figure 9 shows the settings of the bucket. Afterwards, a new IAM user was created on AWS with full access to the Amazon S3 service. It is possible to restrict the user’s access to just one bucket. However, for the purposes of testing in this project, full access is a great fit. Summary and permissions of the new IAM account are shown in the Figure 10. Access key was associated with the newly created IAM user. Access key ID and Secret key ID were recorded for the future use. Access key without Secret key ID is shown in the Figure 11.



Name and region

Bucket name	db-backup-thesis
Region	EU (Frankfurt)

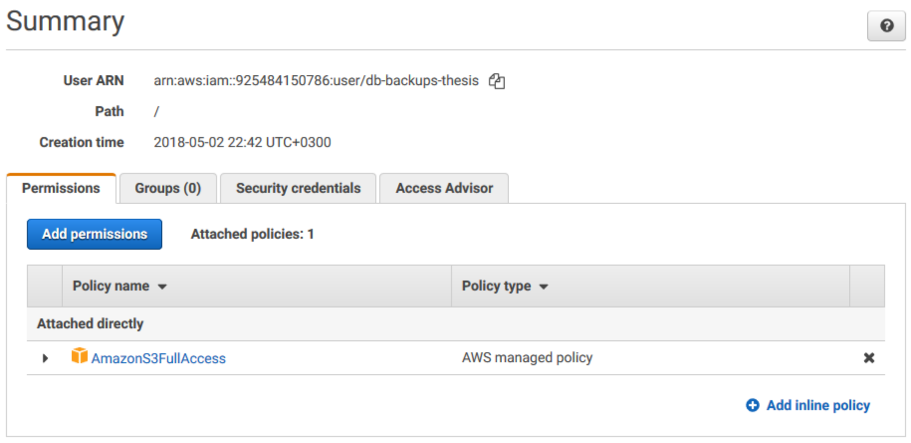
Properties

Versioning	Disabled
Server access logging	Disabled
Tagging	0 Tags
Object-level logging	Disabled
Default encryption	None

Permissions

Users	1
Public permissions	Disabled
System permissions	Disabled

Figure 9. Amazon S3 bucket configuration



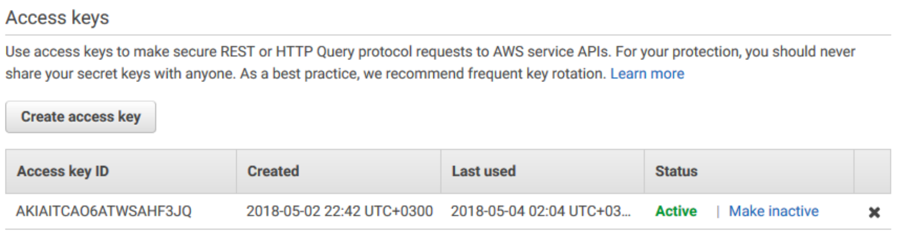
Summary

User ARN	arn:aws:iam::925484150786:user/db-backups-thesis
Path	/
Creation time	2018-05-02 22:42 UTC+0300

Attached policies: 1

Policy name	Policy type
Attached directly	
AmazonS3FullAccess	AWS managed policy

Figure 10. IAM user permissions



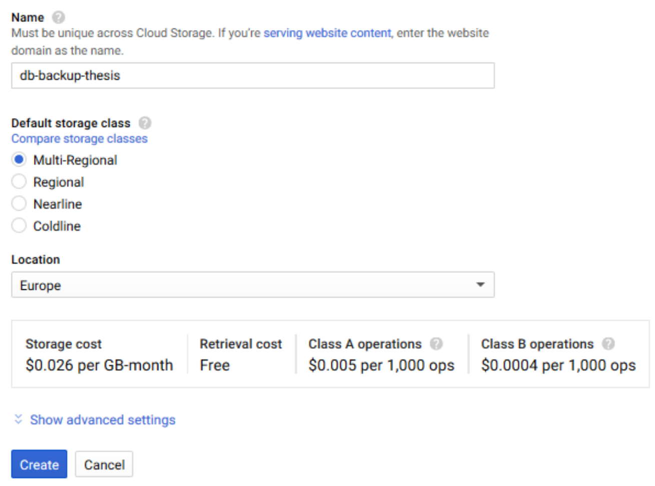
Access keys

Use access keys to make secure REST or HTTP Query protocol requests to AWS service APIs. For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation. Learn more

Access key ID	Created	Last used	Status
AKIAITCAO6ATWSAHF3JQ	2018-05-02 22:42 UTC+0300	2018-05-04 02:04 UTC+03...	Active | Make inactive

Figure 11. Access key for an IAM user account

Another bucket with the same name was created on Google Cloud Storage. Figure 12 shows the settings specified during the creation of the bucket. Google Cloud Platform uses service accounts for access management. Service account was created with full access to Google Cloud Storage and JSON key was generated and downloaded. This process is shown in the Figure 13.



Name

Must be unique across Cloud Storage. If you're serving website content, enter the website domain as the name.

db-backup-thesis

Default storage class

Compare storage classes

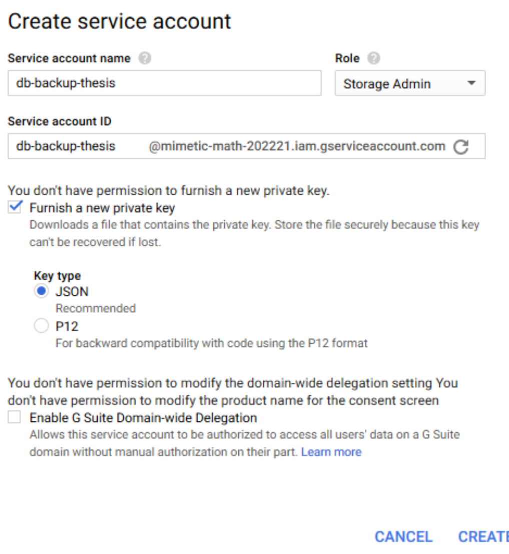
- [● Multi-Regional]
- ○ Regional
- ○ Nearline
- ○ Coldline

Location

Europe

Storage cost	Retrieval cost	Class A operations	Class B operations
$0.026 per GB-month	Free	$0.005 per 1,000 ops	$0.0004 per 1,000 ops

Figure 12. Creation of a Google Cloud Storage bucket



Create service account

Service account name: db-backup-thesis

Role: Storage Admin

Service account ID: db-backup-thesis @mimetic-math-202221.iam.gserviceaccount.com

You don't have permission to furnish a new private key.

[✓ Furnish a new private key]

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

Key type

- [● JSON] Recommended
- ○ P12 For backward compatibility with code using the P12 format

You don't have permission to modify the domain-wide delegation setting You don't have permission to modify the product name for the consent screen

☐ Enable G Suite Domain-wide Delegation

Allows this service account to be authorized to access all users' data on a G Suite domain without manual authorization on their part. Learn more

Figure 13. Creation of a new service account and generation of a key

A new directory was created for the project. The project was initialized with a `npm init` command. This command prompts the user input to receive general information about an application, such as name, version, description, author and license. Afterwards, it creates a new `package.json` file containing all this information.

7.2.1 Setting-up Babel and Webpack

Babel and Webpack play a very important role in this project. First of all, a few npm packages were installed with the following command:

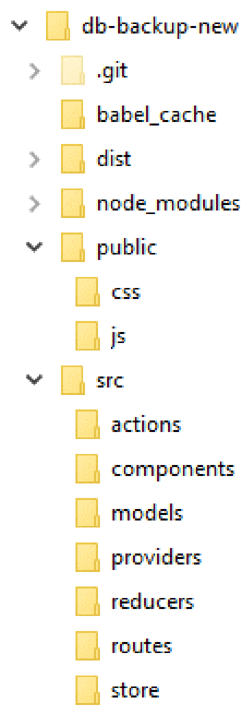
```
npm install --save-dev babel-cli babel-core babel-loader babel-preset-env
babel-preset-react babel-preset-stage-2 css-loader style-loader webpack
webpack-cli
```

The option `save-dev` specifies that the installed packages will be saved as development dependencies in the `package.json` file. The versions of the packages are listed below:

- babel-cli 6.26.0
- babel-core 6.26.0
- babel-loader 7.1.4
- babel-preset-env 1.6.1
- babel-preset-react 6.24.1
- babel-preset-stage-2 6.24.1
- css-loader 0.28.10
- style-loader 0.20.2
- webpack 4.1.0
- webpack-cli 2.0.10

The packages installed with this command require a more detailed description. The command-line interface (CLI) for Babel is provided by a `babel-cli` package. The core functionality of Babel is provided by a `babel-core` package. Three presets for Babel were installed: `babel-preset-react` contains bindings for React, `babel-preset-env` is the main preset that automatically determines plugins based on the targeted environment, `babel-preset-stage-2` provides some additional features that are in a draft stage and are not yet adopted by ECMAScript specification. Webpack and command-line interface for it are installed with `webpack` and `webpack-cli` packages. There are also three loaders for Webpack: `babel-loader` makes use of Babel while the code is being bundled, `css-loader` and `style-loader` allow loading CSS files within the code.

Directory structure used for this project has to be explained before moving onto the configuration of Babel and Webpack. Figure 14 shows the directories that were used throughout this project.



```
db-backup-new
  .git
  babel_cache
  dist
  node_modules
  public
    css
    js
  src
    actions
    components
    models
    providers
    reducers
    routes
    store
```

Figure 14. Directory structure

There were three autogenerated directories: `.git` created by Git, `babel_cache` created by Babel to store the cache and `node_modules` created by npm to store the packages. Source files of the project are located in a `src` directory, while public assets are stored in a `public` directory. Transpiled files will be stored in the `dist` directory.

Babel and Webpack have to be configured once they are installed. The configuration for Babel is stored in a `.babelrc` file which should be located in the root directory. This project has a very simple Babel configuration which just specifies three presets. Code 1 shows how Babel was configured in this project.

```
{
  "presets": ["env", "react", "stage-2"],
  "plugins": []
}
```
Code 1. Configuration in .babelrc file

Webpack's configuration can be stored in a `webpack.config.js` file in the root directory of the project. Code 2 shows the Webpack configuration used for this project.

```
const webpack = require('webpack');
const path = require('path');

module.exports = {
  entry: path.join(__dirname, 'src', 'app-client.js'),
  mode: "production",
  output: {
    path: path.join(__dirname, 'public', 'js'),
    filename: 'bundle.js',
  },
  module: {
    rules: [
      {
        test: path.join(__dirname, 'src'),
        loader: 'babel-loader',
        query: {
          cacheDirectory: 'babel_cache',
          presets: ['react', 'env', 'stage-2']
        }
      },
      {
        test: /\.css$/,
        loader: "style-loader!css-loader"
      }
    ]
  }
};
```

Code 2. Configuration in the webpack.config.js file

The entry point for the client part of the project is `/src/app-client.js` file. After going through three loaders, a bundle is saved as `/public/js/bundle.js`.

7.2.2 Installing global packages and creating the scripts

Two global npm packages were installed: `forever` and `nodemon`. The Forever utility automatically restarts the application in case of crashing. The Nodemon utility automatically restarts the application when any files are changed and it is very useful for testing purposes. The following command was executed:

```
npm install -g forever nodemon
```

Once the global packages were installed, four scripts were added to the `package.json` file. Code 3 shows these scripts.

```
"scripts": {
  "dev": "nodemon ./src/server.js 4000 --exec babel-node",
  "start": "forever start ./dist/server.js 4000",
  "build": "babel src -d dist --presets env,react,stage-2",
  "build-client": "node_modules/.bin/webpack -p"
}
```

Code 3. A section of package.json file containing scripts

The script `dev` starts a development server with Nodemon on port 4000 after transpiling the files with Babel. A production server is started with a `start` script which uses Forever to restart automatically. The server part of the application is built with the `build` script and the client part of the application is built with the `build-client` script which utilizes Webpack to bundle the files. The `start` script runs when `npm start` command is executed. All other scripts can be executed with `npm run <script name>` command.

7.2.3 Configuring the web server and main packages

Global and development packages were already installed. Nevertheless, the main packages had to be added still. The following command was executed to install these packages:

```
npm install --save @blueprintjs/core @google-cloud/storage aws-sdk bcrypt
better-queue body-parser connect-mongo express express-session formidable
mongoose node-cron passport passport-local react react-dom react-redux
react-router-bootstrap react-router-dom react-transition-group redux
```

The packages installed with this command, their versions and descriptions are listed below:

- @blueprintjs/core 2.0.0-rc.2
  Basis of the Blueprint framework
- @google-cloud/storage 1.6.0
  Package for interacting with Google Cloud Storage
- aws-sdk 2.212.1
  Package for interacting with Amazon Web Services
- bcrypt 2.0.1
  Library for the password encryption
- better-queue 3.8.6
  Advanced queue system
- body-parser 1.18.2

Middleware for parsing request body
- connect-mongo 2.0.1
  MongoDB session store
- express 4.16.2
  Web application framework
- express-session 1.15.6
  Session middleware for Express
- formidable 1.2.1
  Package for parsing form data
- mongoose 5.0.9
  Framework for interacting with MongoDB databases
- node-cron 1.2.1
  Task scheduler based on crontab
- passport 0.4.0
  Authentication middleware
- react 16.2.0
  Framework for building user intarfaces
- react-dom 16.2.0
  Package for rendering DOM
- react-redux 5.0.7
  Package that provides React bindings for Redux
- react-router-dom 4.2.2
  Package that provides routing for React
- react-transition-group 2.3.1
  Animation module used by Blueprint
- redux 3.7.2
  Predictable state container

All these packages were used during the development of this project. Once they were installed, a configuration file for the application was created. It was a JSON file called `config.json` and Code 4 shows its structure.

```json
{
    "mongoDBUri": "",
    "secret": ""
}
```
Code 4. Structure of a main configuration file

`mongoDBUri` property defines the connection string for MongoDB database and `secret` stores a secret string for the session management. Then, development of the server part of the application was started. A static HTML file `index.html` was created in `/public` folder and a CSS file `custom.css` was created in `/public/css` folder. The HTML file contained basic page markup and link to a new CSS file. Also, `bundle.js` file was linked there and one div container was created with the

ID `main`. This container is needed for React to place rendered content. A new file called `server.js` was created which was an entry point. Code 5 shows modules that were imported into that file.

```
import Path from 'path';
import Express from 'express';
import Mongoose from 'mongoose';
import BodyParser from 'body-parser';
import Config from './config.json';
```
Code 5. Modules and files imported into server.js file

Next step was to create the web server. Code 6 shows how it was done.

```
//Connecting to the database
Mongoose.connect(Config.mongoDBUri);
const port = process.argv[2] || 4000;
//Initializing Express and creating a server
const app = new Express();
var server = require('http').createServer(app);
//Trust proxy, so we can get the real IP behind reverse proxy
app.enable('trust proxy');
//Handling requests with body-parser
app.use(BodyParser.json());
app.use(BodyParser.urlencoded({ extended: true }));
//Hosting static assets
app.use(Express.static(Path.join(__dirname, '..', 'public')));
//Displaying static HTML page otherwise
app.get('*', (req, res) => {
    res.sendFile(Path.join(__dirname, '..', 'public', 'index.html'));
});
server.listen(port);
```
Code 6. Initializing database connection and creating a web server

At this point, the application was initialized and ready for the further development.

7.3 Routing

Routing was necessary for implementing the backend of the application. A web server should have been able to distinguish the routes in order to provide appropriate API functionality. Express has built-in routing capabilities. First of all, a new file `index.js` was created in `/src/routes`. Code 7 shows how this file was initialized.

```
import { Router } from 'express';
const router = new Router();

router.post('/login', function (req, res, next) {
    //Authenticate
});

export default router;
```
Code 7. Initialization of the router and a route example

Afterwards, the router was connected to Express by importing it from the `server.js` file. Path of all the API routes began with `/api`. Code 8 shows how it was done.

```
import Routes from './routes/index';
app.use('/api', Routes);
```
Code 8. Connecting router to Express

Table 1 shows all the routes that were eventually added to the application and their functions.

Table 1. Server-side routing for an application

Route	Purpose
/login	Performs authentication
/logout	Performs logout
/getuser	Retrieves information about an authenticated user
/createroot	Creates an initial root account
/settings/get	Retrieves settings
/settings/save	Saves settings
/dashboard/get	Retrieves dashboard data: all databases, destinations and schedules
/database/add	Adds a new database
/database/delete	Removes a database
/database/manualbackup	Performs a manual backup on the database

/destination/add	Adds a new destination
/destination/delete	Removes a destination
/scheduler/add	Adds a new schedule
/scheduler/delete	Removes a schedule

All of these routes accepted parameters in JSON that was parsed by `body-parser` middleware, except for `/destination/add` which accepted parameters as a `multipart/form-data` parsed by `formidable`. The reason for this design choice will be explained later. The next step was to create data models and then to implement application functionality.

7.4 Creating data models with Mongoose

Mongoose allows to easily create data models that can later be used for interaction with a MongoDB database. Six models were created for the application: `Backup`, `Database`, `Destination`, `Scheduler`, `Settings` and `User`. These models were created in the files `backup.js`, `database.js`, `destination.js`, `scheduler.js`, `settings.js` and `user.js` respectively and located at `/src/models`. Code 9 shows how one of these models was created.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
module.exports = mongoose.model('Scheduler', new Schema({
    database: { type: Schema.Types.ObjectId, ref: 'Database' },
    destination: { type: Schema.Types.ObjectId, ref: 'Destination' },
    rule: String
}));
```
Code 9. Scheduler model

The fields `database` and `destination` are linked to the `Database` and `Destination` models. These fields contain `ObjectId` of a document that is a part of another collection. This `ObjectId` can later be replaced with the document itself. This process is called population.

Similarly, other models were created. The `Database` model contained `name` and `engine` fields of the type `String` and the `options` field of the type `Object`. The

Destination model contained name and provider fields of the type String and the options field of the type Object. The Settings model contained the fields uniqueId and value of the type String. The User model contained the fields username, password and group of the type String. Finally, the Backup model contained the fields database and destination of the type String that were linked to the Database and Destination models respectively. The fields filename, type, status and log were also of the type String, field startDate was of the type Date and field hashes was of the type Object.

7.5 Implementing the authentication

Passport middleware was used for the authentication. Sessions were handled by an express-session module and stored in a MongoDB database with a connect-mongo module. The module bcrypt was used for hashing the password. Code 10 shows new imports added onto the top of the server.js file.

```
import ExpressSession from 'express-session';
import Passport from 'passport';
import PassportLocal from 'passport-local';
import Bcrypt from 'bcrypt';
import User from './models/user';
```
Code 10. Imports in the server.js file for authentication

Before implementing the authentication itself, session middleware had to be connected and session store had to be configured. Code 11 shows how it was done. The existing MongoDB connection created by Mongoose was used for the session store. Default session’s expiration time was set to one day for security reasons.

```
const MongoStore = require('connect-mongo')(ExpressSession);
app.use(ExpressSession({
    secret: Config.secret,
    resave: false,
    saveUninitialized: false,
    store: new MongoStore({ mongooseConnection: Mongoose.connection }),
    cookie: { maxAge: 86400000 }
}))
app.use(Passport.initialize());
app.use(Passport.session());
```
Code 11. Connecting the Passport, session middleware and session store

Local authentication strategy was connected to Passport and the methods for verifying, serializing and deserializing the user data were implemented. Code 12 shows the code used for that.

```
const LocalStrategy = PassportLocal.Strategy;
Passport.use(new LocalStrategy(function(username, password, done) {
    User.findOne({ username : username }, function(err, user){
        if (err) { return done(err); }
        if (!user) {
            return done(null, false, { message: 'Invalid username' });
        }

        if (Bcrypt.compareSync(password, user.password)) {
            done(null, user)
        } else {
            done(null, false, { message: 'Invalid password' });
        }
    });
}));

Passport.serializeUser(function(user, done) {
    done(null, user.id);
});

Passport.deserializeUser(function(id, done) {
    User.findById(id, function(err, user){
        if (err) {
            done(err)
        };
        done(null, user);
    });
});
```
Code 12. Local strategy for Passport

Also, four routes were implemented in the `/src/routes/index.js` file: `/login`, `/logout`, `/getuser` and `/createroot`. Authentication for local strategy was handled with the Passport's `authenticate` method. The route `/createroot` simply allowed adding an account to the database, but only if there are no other accounts available. Password was hashed with Bcrypt's `hashSync` method. The route `/getuser` was implemented to retrieve general information about the user's account: whether the user is logged in or not, username and group. Middleware was created to verify user's authentication status for restricted routes. Code 13 shows this middleware.

```
function isAuthenticated(req, res, next) {
    if (req.user) {
        return next();
    }

    res.json({
        error: true,
        message: "Access denied"
    });
}
```

Code 13. Middleware for verifying authentication status

This middleware was added to all the routes except for `/login`, `/logout`, `/getuser` and `/createroot`.

7.6 Schema for database engines and storages

In order to make it easy to add new database engines and storage providers to the application, a schema was implemented. It consisted of three files placed in the `/src/providers/` directory: `schema.js`, `server.js` and `types.js`. The file `types.js` had just a simple list of enums used to identify the field type in the schema. Three types were implemented for the purposes of this application. The first one allowed passing a string, the second one allowed passing a number and the third one allowed passing a JSON file. These types also affected user interface: the string and number types generated a simple text box, while the JSON file type generated a file input. Code 14 shows all these types.

```
const ENUM = {
    TYPE_STRING: 0,
    TYPE_NUMBER: 1,
    TYPE_JSONFILE: 2
}

export { ENUM };
```

Code 14. File types.js

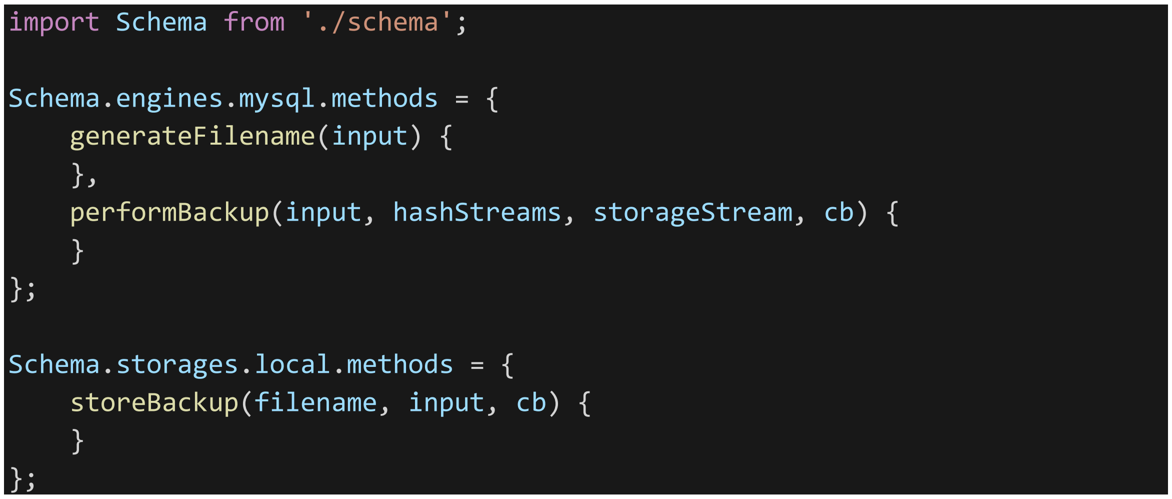
The file `schema.js` was shared between the client and the server and it contained the main information about the fields required to add a new database/storage. Code 15 shows a sample structure of the schema.

```
import { ENUM } from './types';

module.exports = {
    engines: {
        mysql: {
            name: "MySQL",
            fields: {
                hostname: { type: ENUM.TYPE_STRING, default: "", name:
"Hostname", description: "Hostname of your database" },
                port: { type: ENUM.TYPE_NUMBER, default: 3306, name:
"Port", description: "Port of your database", mix: 0, max: 65535 },
                username: { type: ENUM.TYPE_STRING, default: "", name:
"Username", description: "Username for your database" },
                password: { type: ENUM.TYPE_STRING, default: "", name:
"Password", description: "Password for your database", masked: true }
            }
        },
    },
    storages: {
        local: {
            name: "Local",
            fields: {
                path: { type: ENUM.TYPE_STRING, default: "/tmp", name:
"Path", description: "Backup path" }
            }
        },
    },
}
```
Code 15. A sample schema

The schema above is highly simplified as all engines, except for MySQL, and all storages, except for the local one, are omitted. Except for the `type`, every field had a couple of other properties. The property `default` allowed to specify a default value for this field, `name` and `description` were used to provide information about the field in the user interface. For the number type, two additional properties were available: `min` and `max` that allowed specifying minimum and maximum input value respectively. Additionally, every field had a `masked` property that allowed hiding sensitive information.

The `server.js` file was a server-side only file that contained the methods for performing backups. It simply extended the schema for the server use. Code 16 shows an example of the file structure.



```
import Schema from './schema';

Schema.engines.mysql.methods = {
    generateFilename(input) {
    },
    performBackup(input, hashStreams, storageStream, cb) {
    }
};

Schema.storages.local.methods = {
    storeBackup(filename, input, cb) {
    }
};
```

Code 16. Example structure of the server.js file

For database engines, two methods were available: `generateFilename` and `performBackup`. The first one is quite simple: it takes some input, generates a filename based on it and returns a string. The input parameter in all these methods is an object based on the `Backup` model. The next method is responsible for performing the backup and any other operations with the data stream, such as compression. It takes four arguments: `input` is the same as in the method above, `hashStreams` is an array containing writable streams for hashing functions where output stream from the method should be piped. Output stream is also piped into the `storageStream` which is responsible for writing the data to an appropriate provider. The last argument is a callback. It takes two arguments: the first one is a bool defining whether an error occurred or not during the backup and the second one is the backup log.

For storage engines, one method is available: `storeBackup`. This method defines storage logic and returns just a single storage stream which is later passed to the database engine’s `performBackup` method as the third argument. Backup data is later piped into that stream to be stored. The method `storeBackup` takes three arguments. The first one specifies a filename, the second one is the same input as used in database storage methods and the last one is a callback. Callback has exactly the same arguments as the one in the `performBackup` method for database engines.

7.7 Database and storage management

User interaction with the database management system was done via two routes: `/database/add` and `/database/delete`. For the first route, there were two mandatory parameters: `name` and `engine`. They were representing a unique name for the database and engine used respectively. Based on the `engine` parameter, the rest of the required fields were dynamically loaded from the schema and verified. Removal of the database required just one parameter: `databaseId`. This parameter had to be a valid identifier of the database. Four parameters were defined for both MySQL and PostgreSQL in the schema: `hostname`, `port`, `username` and `password`. On the other hand, MongoDB required just one parameter: `uri`. This parameter represented a connection string. When a new database was being added, a MongoDB document was created based on the `Database` model. Field `options` in this model is an object containing all the dynamically loaded parameters.

Two other routes were used for communication with the storage management module: `/destination/add` and `/destination/delete`. Unlike all other routes, `/destination/add` accepts input as `multipart/form-data`. There is a reason for that: some storage providers require a JSON key file to be used for authentication. Google Cloud Platform is one of such providers. As file cannot be provided as a parameter to a JSON object, this route uses form data instead. A module called `formidable` was used for parsing the form data. Code 17 shows how it was implemented.

```
import Formidable from 'formidable';
router.post('/destination/add', isAuthenticated, function (req, res, next){
    var Form = new Formidable.IncomingForm();
    Form.parse(req, function (err, fields, files) {
        if (err) {
            return res.json({ error: true, message: "An unexpected error
occurred" })
        }
        //The code for adding the destination is omitted
    }
});
```

Code 17. Use of formidable to parse the form data

After parsing, `formidable` provides three parameters: error, object with fields and object with files. Similarly to adding the database, all the fields were dynamically loaded from the schema based on the `provider` parameter. Another required parameter was the `name`. A new MongoDB document based on the `Destination` model was being created every time when a new destination is added.

7.8 Performing backups

As it was mentioned earlier, the schema contained a `performBackup` method for each database engine. In general, this method is supposed to perform two operations: backup and data compression. The `child_process` module was used to execute the backup tools for each database engine and gzip compression tool. While mongodump for MongoDB has built-in capabilities for compression, mysqldump and pg_dump for MySQL and PostgreSQL don't. Therefore, additional gzip module had to be used for that.

The `spawn` method from `child_process` module allows to spawn a process and provide command line arguments to it. Unlike `exec` method, it has protection from the injection of code that could have been used to execute another command. Exit code, standard output (`stdout`) and standard error (`stderr`) can be easily captured. Code 18 shows an example of how `child_process` module was used within `performBackup` method. A significant part of the code below is omitted.

```
import { spawn } from 'child_process';
let stderr = [];
const mysqldump = spawn('mysqldump', [
    '--all-databases',
    `--user=${input.database.options.username}`,
    `--password=${input.database.options.password}`,
    `--port=${input.database.options.port}`,
    `--host=${input.database.options.hostname}`,
    '--verbose']);
mysqldump.stdout.pipe(gzip.stdin);
mysqldump.stderr.on('data', data => {
    stderr.push(data);
});
mysqldump.on('exit', code => {
    stderr.push(Buffer.from(`Backup was completed with code ${code}.\n`))
    if (code == 0) {
        gzip.stdin.end();
        stderr.push(Buffer.from(`Starting compression...\n`))
    } else {
        cb(true, Buffer.concat(stderr).toString());
    }
})
```
Code 18. Use of the child_process module to run mysqldump

In the example above, mysqldump process is being spawned with `stderr` being written to a string and `stdout` being piped directly into a standard input (`stdin`) of gzip. Spawn of gzip is omitted due to it being done the same way. Once the backup is completed, resulting stream of data is piped into all the streams in a `hashStreams` array and a `storageStream` stream in order to perform hash calculations and store the data in an appropriate storage.

7.9 Hash calculations and storage

Once the backup was performed, hash values had to be calculated and the backup had to be stored. Four hash values were being generated for every backup: MD5, SHA-1, SHA-256 and SHA-512. These values were calculated using Node.js' built-in `crypto` module. A function `CalculateHashes` was implemented to create the writable streams for hashing and return an array of them. Code 19 shows how a stream could be created with a crypto module on the example of MD5 algorithm.

```
import Crypto from 'crypto';
let md5Val;
let md5 = Crypto.createHash('md5');
md5.on('readable', () => {
    const data = md5.read();
    if (data) {
        md5Val = data.toString('hex');
    }
})
```
Code 19. Creation of a hashing stream with crypto module

The resulting string was being converted to hexadecimal format. The result of `CalculateHashes` function was being passed as the second argument to the `performBackup` method. That function also had one argument for callback. This callback had to be called once all hash values were calculated in order to return the result.

The data was being stored at the same time as hash values were being generated. A `storeBackup` method of an appropriate storage provider was being used for that. This method had to return a writable stream that would later be passed as the third argument to the `performBackup` method. Code 20 shows an example of `storeBackup` method.

```
Schema.storages.local.methods = {
    storeBackup(filename, input, cb) {
        let stream =
fs.createWriteStream(`${input.destination.options.path}/${filename}`, {
flags: 'w' });
        stream.on('error', function(err) {
            cb(true, err);
        });
        stream.on('close', function() {
            cb(false, `File was successfully written to
${input.destination.options.path}/${filename}.\n`);
        })

        return stream;
    }
};
```
Code 20. storeBackup method for the local storage provider

Two more backup providers were also created in a similar way: Amazon S3 and Google Cloud Storage. Two modules were used for the interaction with the cloud providers' API: `aws-sdk` and `@google-cloud/storage`. Backup was considered complete once the callback passed as the third argument to `storeBackup` method was called. At this point, backup status, filename, log and hash values were being stored in the database.

7.10 Queue

A queue was implemented using the `better-queue` package in order to process each individual backup as fast and efficiently as possible. A new file `queue.js` was created in the `/src` directory. Code 21 shows the code used to create the queue.

```
import Queue from 'better-queue';
var BackupQueue = new Queue(function (input, cb) {
});
```

Code 21. Initialization of a queue

There were two parameters in a method used for processing a queue element. The first one defined the data provided for processing. This data was passed as the first argument to the `push` method provided by the queue object. An object based on the `Backup` model was meant to be passed as an input to a queue. The second argument was the callback. It was being called once the task was finished to make queue proceed to the next task. Function passed to the queue constructor was used to combine all the elements of backup logic together: backup processing, hashing and storage.

7.11 Manual backup and scheduler

Two options were available for performing backups: manual backup and the use of the scheduler. Manual backup feature was used to perform a one-time backup of the selected database to the selected destination. Route `/database/manualbackup` was used for performing manual backups. It accepted just two parameters: `databaseId` and `destinationId`. The first parameter was used to define the database for which backup will be performed and the second one was used to define the destination.

Firstly, database and destination documents were being retrieved from the database based on `databaseId` and `destinationId` provided. Then, a new backup object was being created from the `Backup` model.

```
import Backup from '../models/backup';
var localBackup = new Backup({
    database: database._id,
    destination: destination._id,
    filename: null,
    startDate: Date.now(),
    type: "manual",
    status: "queued",
    hashes: {},
    log: ""
});
```
Code 22. A new backup object

However, at this point the database and destination properties were of the `ObjectId` type. It wasn’t possible to pass this object to a queue, because these paths weren’t populated yet. Code 23 shows how the paths were populated.

```
Backup.findOne({_id: localBackup._id})
.populate('database')
.populate('destination')
.exec(function(err, backup) {
    BackupQueue.push(backup);
});
```
Code 23. Populating paths

After population, retrieved document could be passed to the queue for processing. Scheduler is another way of performing a backup. The `node-cron` package was used for implementing the scheduling capabilities. Three new methods were added to the `queue.js` file: `AddSchedule`, `RemoveSchedule` and `InitData`. Also, a global object `Tasks` was created to store relationships between an `ObjectId` of a scheduler object and an ID of a job created by `node-cron`.

The `AddSchedule` method accepted one argument: `schedule`. This argument represented an object based on the `Scheduler` model. Then, a new schedule was added based on the rule specified in that object. `RemoveSchedule` method allows to destroy a task using an `ObjectId` of a scheduler object. Code 24 shows how these two methods were implemented. A part of `AddSchedule` method is omitted.

```
import Cron from 'node-cron';
function AddSchedule(schedule) {
    Tasks[schedule._id] = Cron.schedule(schedule.rule, function () {
        //Start a new backup here in a similar way to manual backup
    });
}

function RemoveSchedule(id) {
    Tasks[id].destroy()
}
```
Code 24. AddSchedule and RemoveSchedule methods

The `InitData` method is simply called once on application's startup. It is required to load schedulers and queued backups that were started before application's shutdown but not processed yet.

Two routes were used for the management of the schedulers: `/scheduler/add` and `/scheduler/delete`. The first one accepted three parameters: `databaseId`, `destinationId` and `rule`. The rule parameter had to be a valid CRON expression. It was being validated with `validate` function from the `node-cron` module. After validation, this route would add a new document into the database based on the `Scheduler` model and schedule a new task with `AddSchedule` method. The second route was used to remove a scheduler based on the `taskId` property.

7.12 User interface

Once application's backend was implemented, a user interface had to be created for interaction with it. It was decided to use React as a framework for building the UI, Redux as a state container and Blueprint as a collection of commonly used React components. The Fetch API was used for communication with the backend. Entry point for the client side of the application was `app-client.js` file located in the `/src` directory. Redux was used to store the user state which had to be accessed all over the application. Therefore, basic setup was performed: `User` reducer was created and added to the root reducer using `combineReducers` method. Afterwards, a store was created to store the application state. Also, one action was implemented: `UserState`. React Router was also initialized for performing client-side routing.

Nine custom components were created. The `App` component was used to combine all other components together and perform routing. It was also used to retrieve user’s authentication status. The `Header` component was used to display application’s header, including navigation and information about authentication status. A simple footer was displayed using `Footer` component. It was important to make sure that sensitive data is not displayed right away. Therefore, `MaskedText` component was created. It allowed to display the text in a hidden state with an option to show the actual contents by pressing a button. This component was later used to hide database passwords. The `Home` component represented the main page of the application. It would then either display the `Dashboard` component or the `LoginForm` component, depending on whether user was logged in or not. Finally, the `Toaster` component was used to display notifications.

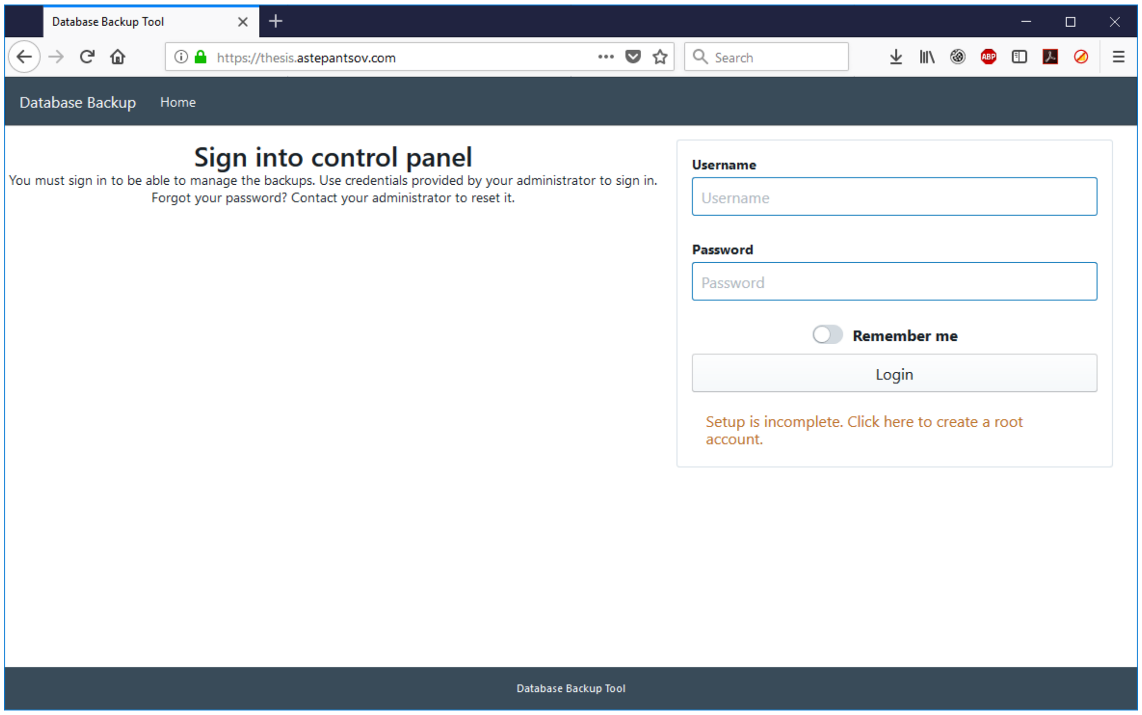
The main component of the application was `Dashboard`. It had four sections: Databases, Destinations, Scheduler and Latest Backups. Databases and Destinations sections were dynamically loading layout from the schema to display correct fields.

The Databases section allowed to view existing databases and all the information about them, such as name, engine and all the options specified in the schema. Additionally, records could be removed and manual backup could be performed through the menu button. The Destinations section allowed to view, add or remove the destinations. Name, provider and options were displayed. The Scheduler section was used for management of the tasks. Each task had a database, a destination and a rule displayed. Also, tasks could be removed. The Latest Backups sections contained general information about the backups: database, destination, filename, start date, type (manual or scheduled), status (queued, finished or failed). Additionally, backup log and hash values could be viewed. All dashboard data was being refreshed every ten seconds.

Client-side files were bundled into a single `bundle.js` file using `build-client` script that was created during the environment preparation phase. At that point, application was ready.

7.13 Results

For testing purposes, the application was deployed on an Amazon EC2 instance that was earlier described. Nginx was configured as a reverse proxy and DNS records were set for the domain name. The main page of the application can be seen in the Figure 15. It contains a login form and basic information. There is no account created yet, so the setup is considered incomplete.



Sign into control panel

You must sign in to be able to manage the backups. Use credentials provided by your administrator to sign in.
Forgot your password? Contact your administrator to reset it.

Username

Password

Remember me

Login

Setup is incomplete. Click here to create a root account.

Database Backup Tool

Figure 15. Application’s main page

The process of creating a root account is shown in the Figure 16. Once the root username and password are specified, the account is created.

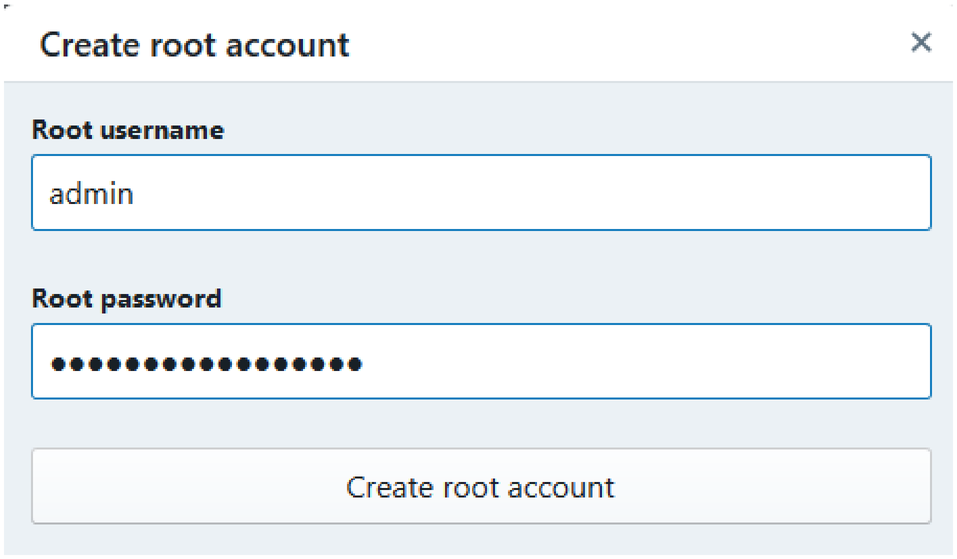


Figure 16. Creation of a root account

These credentials can be used to log into the application. A user gets access to the dashboard when logged in. It is shown in the Figure 17.

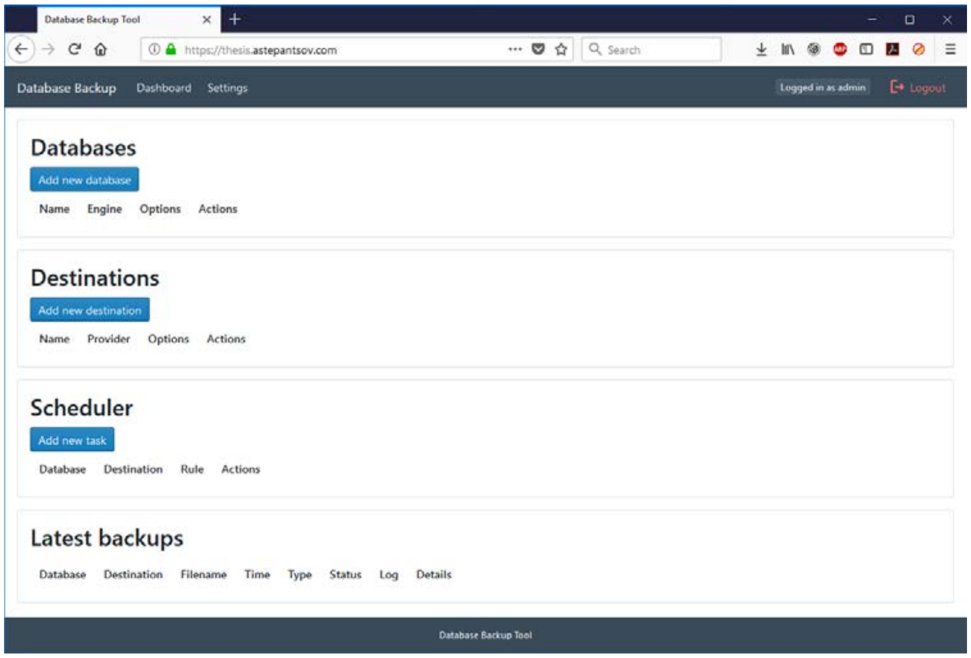
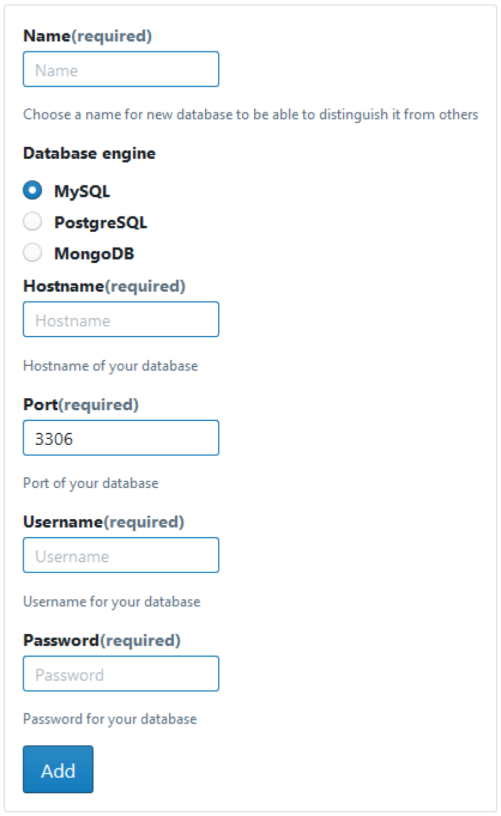


Figure 17. Empty dashboard

A new database, destination or task can be added using the appropriate buttons. Figure 18 shows how the panel for adding a new database looks.



Name(required)

Name

Choose a name for new database to be able to distinguish it from others

Database engine

[✓ MySQL]
PostgreSQL
MongoDB

Hostname(required)

Hostname

Hostname of your database

Port(required)

3306

Port of your database

Username(required)

Username

Username for your database

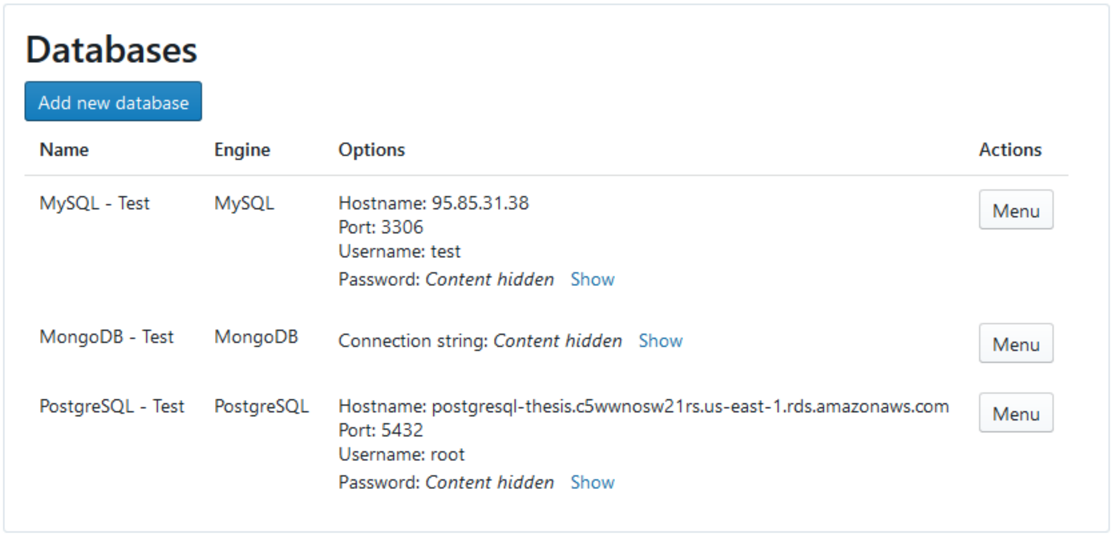
Password(required)

Password

Password for your database

Figure 18. Creating a new database record

Three new databases were added: one of each available type. The result is shown in the Figure 19.

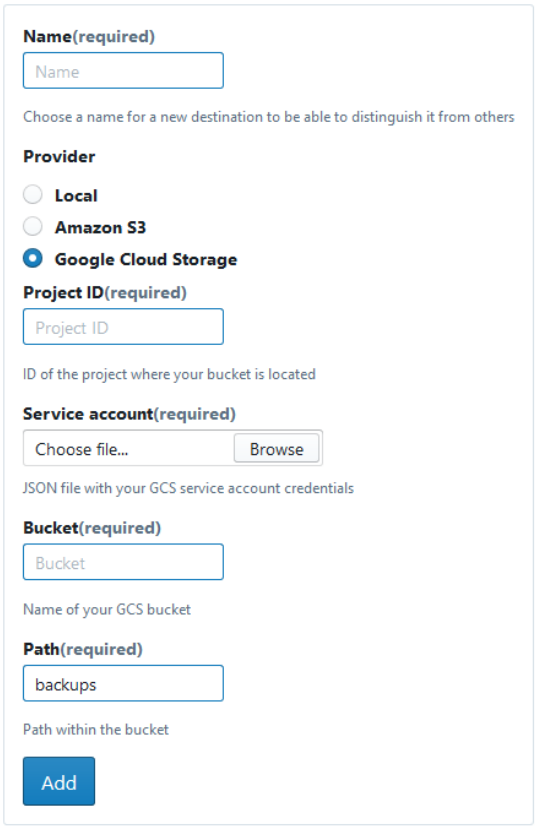


Databases

Name	Engine	Options	Actions
MySQL - Test	MySQL	Hostname: 95.85.31.38 Port: 3306 Username: test Password: *Content hidden* Show	Menu
MongoDB - Test	MongoDB	Connection string: *Content hidden* Show	Menu
PostgreSQL - Test	PostgreSQL	Hostname: postgresql-thesis.c5wwnosw21rs.us-east-1.rds.amazonaws.com Port: 5432 Username: root Password: *Content hidden* Show	Menu

Figure 19. List of databases

A panel for adding a new destination is displayed in the Figure 20.



Name(required)

Name

Choose a name for a new destination to be able to distinguish it from others

Provider

- Local
- Amazon S3
- [✓ Google Cloud Storage]

Project ID(required)

Project ID

ID of the project where your bucket is located

Service account(required)

Choose file... Browse

JSON file with your GCS service account credentials

Bucket(required)

Bucket

Name of your GCS bucket

Path(required)

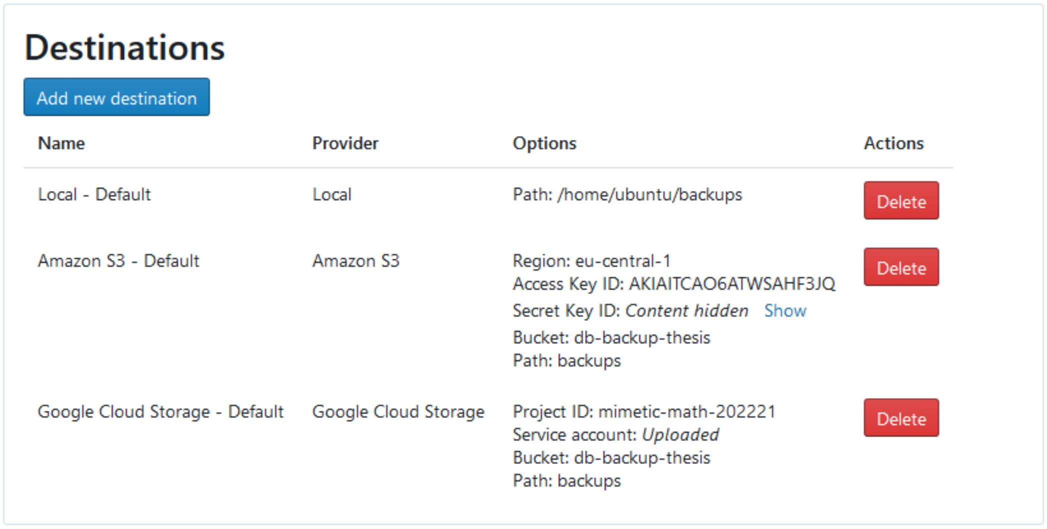
backups

Path within the bucket

Add

Figure 20. Creating a new destination record

One of the each type of destinations was added for the testing purposes. They can be seen in the Figure 21.



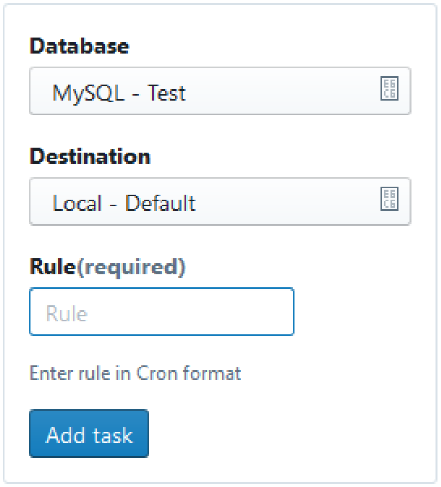
Destinations

Add new destination

Name	Provider	Options	Actions
Local - Default	Local	Path: /home/ubuntu/backups	Delete
Amazon S3 - Default	Amazon S3	Region: eu-central-1 Access Key ID: AKIAITCAO6ATWSAHF3JQ Secret Key ID: *Content hidden* Show Bucket: db-backup-thesis Path: backups	Delete
Google Cloud Storage - Default	Google Cloud Storage	Project ID: mimetic-math-202221 Service account: *Uploaded* Bucket: db-backup-thesis Path: backups	Delete

Figure 21. List of destinations

The task creation panel is shown in the Figure 22.



Database

MySQL - Test

Destination

Local - Default

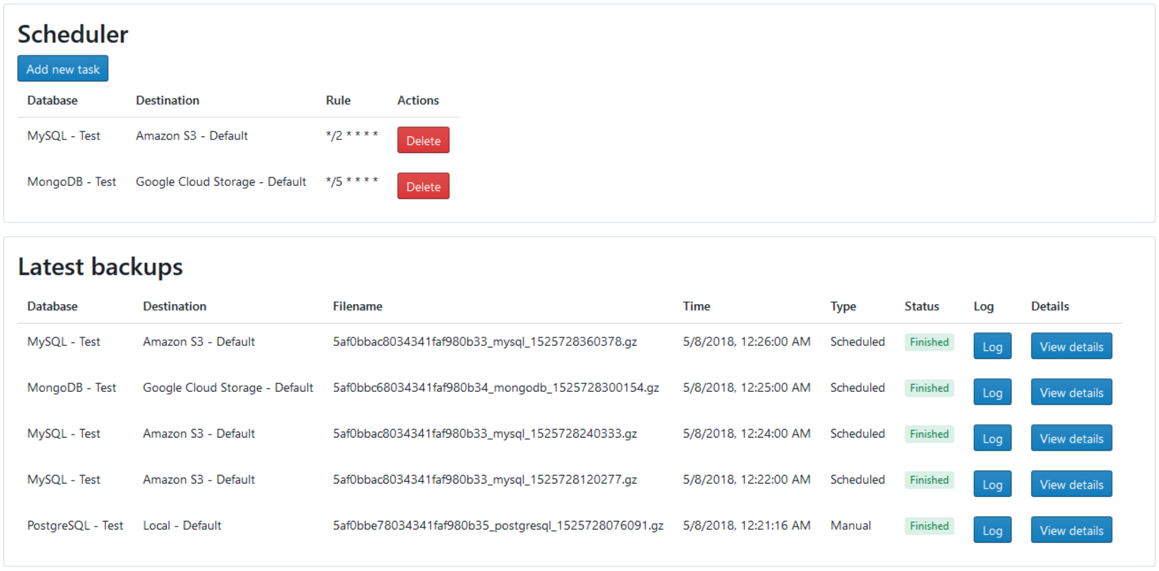
Rule(required)

Rule

Enter rule in Cron format

Figure 22. Creating a new task

Now, a couple of tasks were added and several backups were performed. The first task would backup the MySQL database to the Amazon S3 every two minutes. The second task would backup the MongoDB database to the Google Cloud Storage every five minutes. The PostgreSQL database would be backed up manually to the local storage. Figure 23 shows these tasks and backups that were performed.



Scheduler

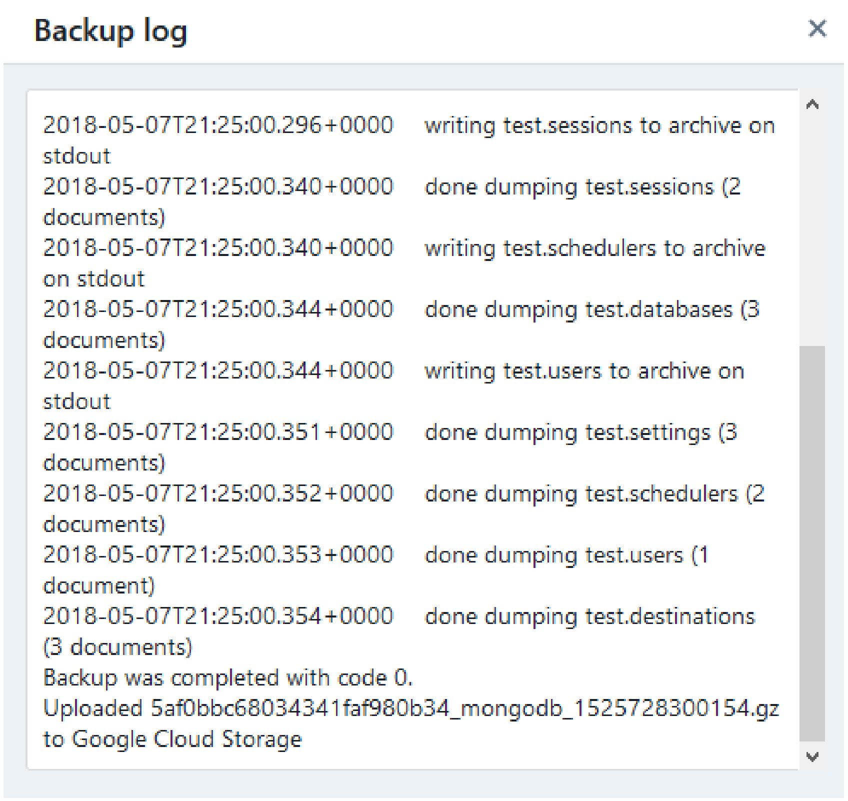
Database	Destination	Rule	Actions
MySQL - Test	Amazon S3 - Default	*/2 * * * *	Delete
MongoDB - Test	Google Cloud Storage - Default	*/5 * * * *	Delete

Latest backups

Database	Destination	Filename	Time	Type	Status	Log	Details
MySQL - Test	Amazon S3 - Default	5af0bbac8034341faf980b33_mysql_1525728360378.gz	5/8/2018, 12:26:00 AM	Scheduled	Finished	Log	View details
MongoDB - Test	Google Cloud Storage - Default	5af0bbc68034341faf980b34_mongodb_1525728300154.gz	5/8/2018, 12:25:00 AM	Scheduled	Finished	Log	View details
MySQL - Test	Amazon S3 - Default	5af0bbac8034341faf980b33_mysql_1525728240333.gz	5/8/2018, 12:24:00 AM	Scheduled	Finished	Log	View details
MySQL - Test	Amazon S3 - Default	5af0bbac8034341faf980b33_mysql_1525728120277.gz	5/8/2018, 12:22:00 AM	Scheduled	Finished	Log	View details
PostgreSQL - Test	Local - Default	5af0bbe78034341faf980b35_postgresql_1525728076091.gz	5/8/2018, 12:21:16 AM	Manual	Finished	Log	View details

Figure 23. Tasks and backup list

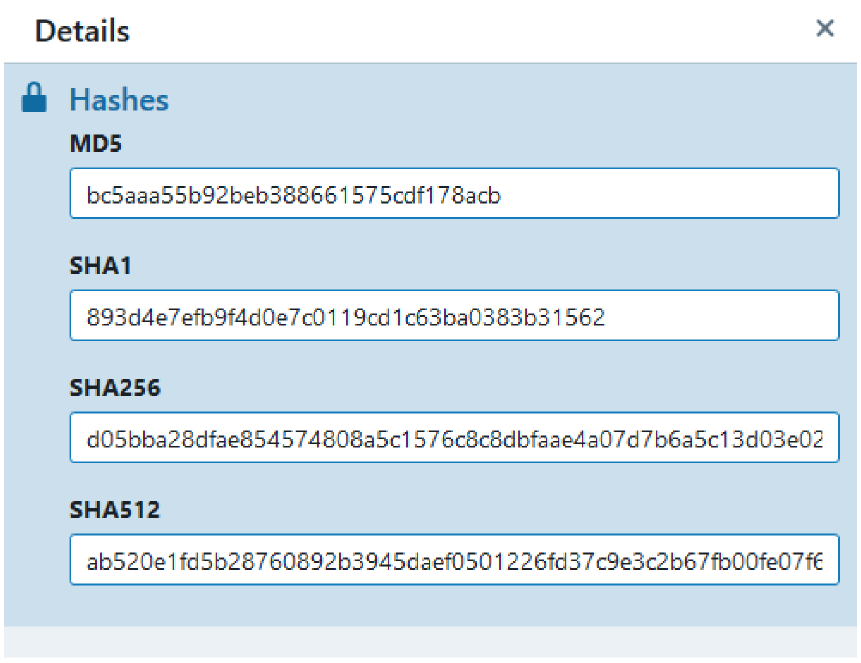
The Log button allows viewing the backup log. The contents of such a log can be seen in the Figure 24.



```
2018-05-07T21:25:00.296+0000    writing test.sessions to archive on
stdout
2018-05-07T21:25:00.340+0000    done dumping test.sessions (2
documents)
2018-05-07T21:25:00.340+0000    writing test.schedulers to archive
on stdout
2018-05-07T21:25:00.344+0000    done dumping test.databases (3
documents)
2018-05-07T21:25:00.344+0000    writing test.users to archive on
stdout
2018-05-07T21:25:00.351+0000    done dumping test.settings (3
documents)
2018-05-07T21:25:00.352+0000    done dumping test.schedulers (2
documents)
2018-05-07T21:25:00.353+0000    done dumping test.users (1
document)
2018-05-07T21:25:00.354+0000    done dumping test.destinations
(3 documents)
Backup was completed with code 0.
Uploaded 5af0bbc68034341faf980b34_mongodb_1525728300154.gz
to Google Cloud Storage
```

Figure 24. Backup log for a MongoDB database

The View details button allows seeing hash values for the generated file. Figure 25 shows this panel.



MD5
bc5aaa55b92beb388661575cdf178acb

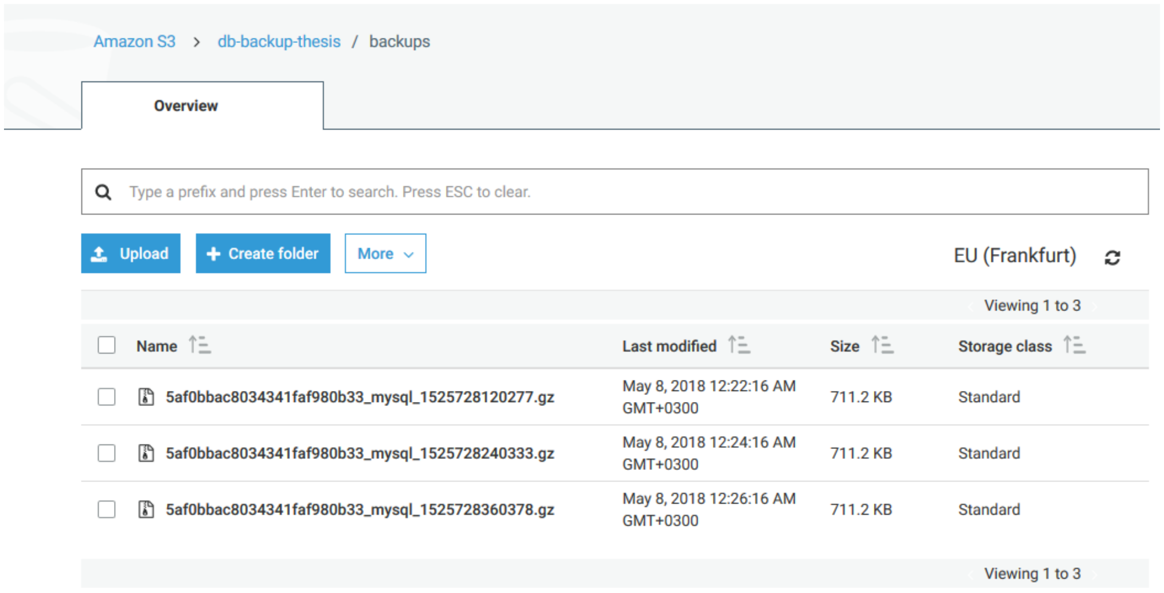
SHA1
893d4e7efb9f4d0e7c0119cd1c63ba0383b31562

SHA256
d05bba28dfae854574808a5c1576c8c8dbfaae4a07d7b6a5c13d03e02

SHA512
ab520e1fd5b28760892b3945daef0501226fd37c9e3c2b67fb00fe07f6

Figure 25. Hash values for a backup

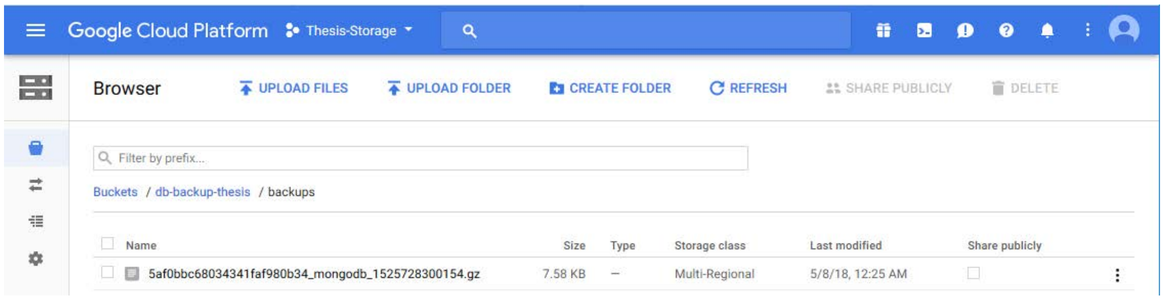
Figures 26 and 27 show the contents of the Amazon S3 and Google Cloud Storage buckets after performing the backups.



Amazon S3 > db-backup-thesis / backups
Overview
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
More
EU (Frankfurt)
Viewing 1 to 3

Name	Last modified	Size	Storage class
5af0bbac8034341faf980b33_mysql_1525728120277.gz	May 8, 2018 12:22:16 AM GMT+0300	711.2 KB	Standard
5af0bbac8034341faf980b33_mysql_1525728240333.gz	May 8, 2018 12:24:16 AM GMT+0300	711.2 KB	Standard
5af0bbac8034341faf980b33_mysql_1525728360378.gz	May 8, 2018 12:26:16 AM GMT+0300	711.2 KB	Standard

Figure 26. The contents of the Amazon S3 bucket



Name	Size	Type	Storage class	Last modified	Share publicly
5af0bbc68034341faf980b34_mongodb_1525728300154.gz	7.58 KB	–	Multi-Regional	5/8/18, 12:25 AM	

Figure 27. The contents of the Google Cloud Storage bucket

During this test session, everything went as planned. The application worked correctly and without any interruptions.

8 CONCLUSIONS

Insufficient data protection is one of the key issues in the modern world. The main goal of this work was to contribute to the solution of this problem. There are many different challenges related to the safety of information and I have tried to shed light upon some of them in this thesis. The goal was reached by successfully creating an expandable application for the centralized database backup management. All the planned features were fully implemented and the application performs its job well.

The theoretical part of this thesis provides a detailed overview of backup methods available for certain popular databases and security issues related to the backups. The role of cloud storage solutions in this matter is shown as well. This information is later used for carrying out the practical part of the project. I have attempted to write a decently detailed, but not too overwhelming by unnecessary details, explanation of the application's implementation process. The application was implemented with Node.js which is a really good choice for highly scalable products. React is a great framework for building modern and highly interactive user experiences.

The project can be extended in many different ways. First of all, support for more database engines and storage providers may be added. It will be extremely easy to accomplish due to the application's architecture. Certain secondary features can be implemented in the future, such as user management and advanced permission system. Another way to make the application better is by bringing additional capabilities for customization, such as optional fields in the schema and more field types. Moreover, there is a huge number of topics in the information security field closely related to this thesis.

REFERENCES

Abramov, D. 2018. Actions. WWW document. Available at: https://redux.js.org/basics/actions [Accessed 21 April 2018].

Abramov, D. 2018. Reducers. WWW document. Available at: https://redux.js.org/basics/reducers [Accessed 21 April 2018].

Abramov, D. 2018. Store. WWW document. Available at: https://redux.js.org/basics/store [Accessed 21 April 2018].

Amazon Web Services. 2018. Amazon S3 Features. WWW document. Available at: https://aws.amazon.com/s3/features/ [Accessed 14 April 2018].

Amazon Web Services. 2018. What is Cloud Computing? WWW document. Available at: https://aws.amazon.com/what-is-cloud-computing/ [Accessed 14 April 2018].

Bitnine Co, Ltd. 2016. Advantages of PostgreSQL. WWW document. Available at: http://bitnine.net/blog-postgresql/advantages-of-postgresql/ [Accessed 8 April 2018].

Black Duck Software, Inc. 2018. MySQL. WWW document. Available at: https://www.openhub.net/p/mysql [Accessed 14 April 2018].

ByteScout. 2014. MongoDB History and Advantages. WWW document. Available at: https://bytescout.com/blog/2014/09/mongodb-history-and-advantages.html [Accessed 14 April 2018.]

Chamberlin, D. D. 2012. Early History of SQL. Washington D.C.: IEEE Computer Society.

Data Science Central. 2016. History of MySQL. WWW document. Available at: https://www.datasciencecentral.com/profiles/blogs/history-of-mysql [Accessed 15 April 2018].

ECMAScript. 2007. Proposed ECMAScript 4th Edition – Language Overview. Available at: https://www.webcitation.org/5rBiWD4P6?url=http://www.ecmascript.org/es4/spec/overview.pdf [Accessed 9 April 2018].

ECRYPT. 2009. The Hash Function Zoo. WWW document. Available at: https://ehash.iaik.tugraz.at/wiki/The_Hash_Function_Zoo [Accessed 21 April 2018].

GitLab Inc. 2017. Gitlab.com database incident. WWW document. Available at: https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/ [Accessed 7 March 2018].

Google LLC. 2018. Google Cloud Storage. WWW document. Available at: https://cloud.google.com/storage/ [Accessed 14 April 2018].

IBM. 2018. Public key cryptography. WWW document. Available at: https://www.ibm.com/support/knowledgecenter/SSB23S_1.1.0.14/gtps7/s7pkey.html [Accessed 15 April 2018].

IBM. 2018. Relational database. WWW document. Available at: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/reldb/ [Accessed 9 April 2018].

IBM. 2018. Symmetric cryptography. WWW document. Available at: https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.14/gtps7/s7symm.html [Accessed 15 April 2018].

Krill, P. 2014. React: Making faster, smoother UIs for data-driven Web apps. WWW document. Available at: https://www.infoworld.com/article/2608181/javascript/react--making-faster--smoother-uis-for-data-driven-web-apps.html [Accessed 21 April 2018].

Leavitt, N. 2010. Will NoSQL Databases Live Up to Their Promise? Washington, D.C.: IEEE Computer Society.

Lifewire. 2017. What You Need to Know About Structured Query Language. WWW document. Available at: https://www.lifewire.com/what-is-sql-1019769 [Accessed 15 April 2018].

Makris, A., Tserpes, K., Andronikou, V., Anagnostopoulos, D. 2016. A Classification of NoSQL Data Stores Based on Key Design Characteristics. Madrid: Elsevier B.V.

MongoDB, Inc. 2018. mongodump. WWW document. Available at: https://docs.mongodb.com/manual/reference/program/mongodump/ [Accessed 14 April 2018].

MongoDB, Inc. 2018. MongoDB Backup Methods. WWW document. Available at: https://docs.mongodb.com/manual/core/backups [Accessed 11 April 2018].

MongoDB, Inc. 2018. What is MongoDB? WWW document. Available at: https://www.mongodb.com/what-is-mongodb [Accessed 14 April 2018].

Mourghen, Denis. 2016. Cloud Computing- Origin, Advantages & Limitations. WWW document. Available at: https://www.linkedin.com/pulse/cloud-computing-origin-advantages-limitations-denis-mourghen [Accessed 14 April 2018].

Mozilla Corporation. 2018. About JavaScript. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript [Accessed 21 April 2018].

Mozilla Corporation. 2018. HTML basics. Available at: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics [Accessed 5 April 2018].

Node.js Foundation. 2018. About Node.js®. WWW document. Available at: https://nodejs.org/en/about/ [Accessed 21 April 2018].

npm, Inc. 2018. npm-install – Install a package. WWW document. Available at: https://docs.npmjs.com/cli/install [Accessed 21 April 2018].

npm, Inc. 2018. npm-uninstall – Remove a package. WWW document. Available at: https://docs.npmjs.com/cli/uninstall [Accessed 21 April 2018].

npm, Inc. 2018. npm – javascript package manager. WWW document. Available at: https://docs.npmjs.com/cli/npm [Accessed 21 April 2018].

npm, Inc. 2018. package.json – Specifics of npm's package.json handling. WWW document. Available at: https://docs.npmjs.com/files/package.json [Accessed 21 April 2018].

ntu.edu.sg. 2010. A Quick-Start Tutorial on Relational Database Design. Available at: https://www.ntu.edu.sg/home/ehchua/programming/sql/Relational_Database_Design.html [Accessed 7 April 2018].

Oracle Corporation. 2018. Database Backup Methods. WWW document. Available at: https://dev.mysql.com/doc/refman/5.7/en/backup-methods.html [Accessed 11 April 2018].

Oracle Corporation. 2018. mysqldump – A Database Backup Program. WWW document. Available at: https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html [Accessed 11 April 2018].

Oracle Corporation. 2018. MySQL Enterprise Backup Overview. WWW document. Available at: https://dev.mysql.com/doc/refman/5.7/en/mysql-enterprise-backup.html [Accessed 11 April 2018].

Oracle Corporation. 2018. The Main Features of MySQL. WWW document. Available at: https://dev.mysql.com/doc/refman/5.7/en/features.html [Accessed 7 April 2018].

OWASP. 2017. OWASP Top 10 – 2017: The Ten Most Critical Web Application Security Risks. WWW document. Available at: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf [Accessed 16 April 2018].

OWASP. 2018. Cross-Site Request Forgery (CSRF). WWW document. Available at: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF) [Accessed 16 April 2018].

SANS Technology Institute. 2018. Hash Functions. WWW document. Available at: https://www.sans.edu/cyber-research/security-laboratory/article/hash-functions [Accessed 15 April 2018].

solid IT. 2018. DB-Engines Ranking. WWW document. Available at: https://db-engines.com/en/ranking [Accessed 7 April 2018].

The PostgreSQL Global Development Group. 2018. About. WWW document. Available at: https://www.postgresql.org/about/ [Accessed 14 April 2018].

The PostgreSQL Global Development Group. 2018. Backup and Restore. WWW document. Available at: https://www.postgresql.org/docs/8.3/static/backup.html [Accessed 13 April 2018].

The PostgreSQL Global Development Group. 2018. History. WWW document. Available at: https://www.postgresql.org/docs/8.4/static/history.html [Accessed 8 April 2018].

The PostgreSQL Global Development Group. 2018. pg_dump. WWW document. Available at: https://www.postgresql.org/docs/8.3/static/app-pgdump.html [Accessed 13 April 2018].

The PostgreSQL Global Development Group. 2018. pg_restore. WWW document. Available at: https://www.postgresql.org/docs/8.3/static/app-pgrestore.html [Accessed 13 April 2018].

W3C. 2017. HTML 5.2. WWW document. Available at: https://www.w3.org/TR/html/ [Accessed 5 April 2018].

W3C. 2018. What is CSS?. WWW document. Available at: https://www.w3.org/standards/webdesign/htmlcss#whatcss [Accessed 5 April 2018]

W3CSchools. 2018. JavaScript Versions. WWW document. Available at: https://www.w3schools.com/js/js_versions.asp [Accessed 21 April 2018].

webpack.js.org. 2018. Concepts. WWW document. Available at: https://webpack.js.org/concepts/ [Accessed 21 April 2018].

Wordpress.com. 2018. CSS Basics. WWW document. Available at: https://en.support.wordpress.com/custom-design/css-basics/ [Accessed 5 April 2018]