

Work of Boldizsar Banfia  
Student number: 15363236  
Flash Card Application:  
Flashy

Git Repo:

[https://github.com/Boldizsarb/advanced\\_database](https://github.com/Boldizsarb/advanced_database)

Deployed at:

<https://odd-jeans-hare.cyclic.app/>

too look at pictures please use account:

[example@example.com](mailto:example@example.com)

pw: example

## Introduction

As a not native English speaker, upon arrival to the country I had to teach myself in English since I had not spoken a word before. During my research, seeking the most efficient way to learn it, I came across various techniques. Multiple high-volume study shows that besides many factors including habit, lifestyle, and attitude, one of the best methods to memorise anything is the retrieval practice with high repetition (FutureLearn 2022). Flash Cards are a very good representation of this. The hard part is to find the right one since most are pre-written and cost a fortune.

My application offers a solution to both problems, let alone the great accessibility on the web on any device. There are no pre-written cards, it let the user customize their cards without worrying about the learning curb or having to study something that they potentially would never use. The application is not subject bound, the only limit is the imagination of the user.

## System Overview

The application Flashy was structured following the MVC (Model, View, and Controller) architectural pattern (Geeks for geeks 2021). This not only makes the development faster but easier as well among many benefits such as good maintainability and expansion (Six Benefits of Using MVC Model for Effective Web Application Development 2016). The user side part was built with conventional technologies including HTML (ejs), CSS, and JavaScript. Note that all GUI components were designed individually solely using CSS and a minuscule amount of Bootstrap. On the server side under the hood, Node.js pulls the strings. It is a highly scalable, lightweight, quick, and data-intensive framework that runs on the Chrome v8 engine, which turns javascript code into machine code (Heller 2020).

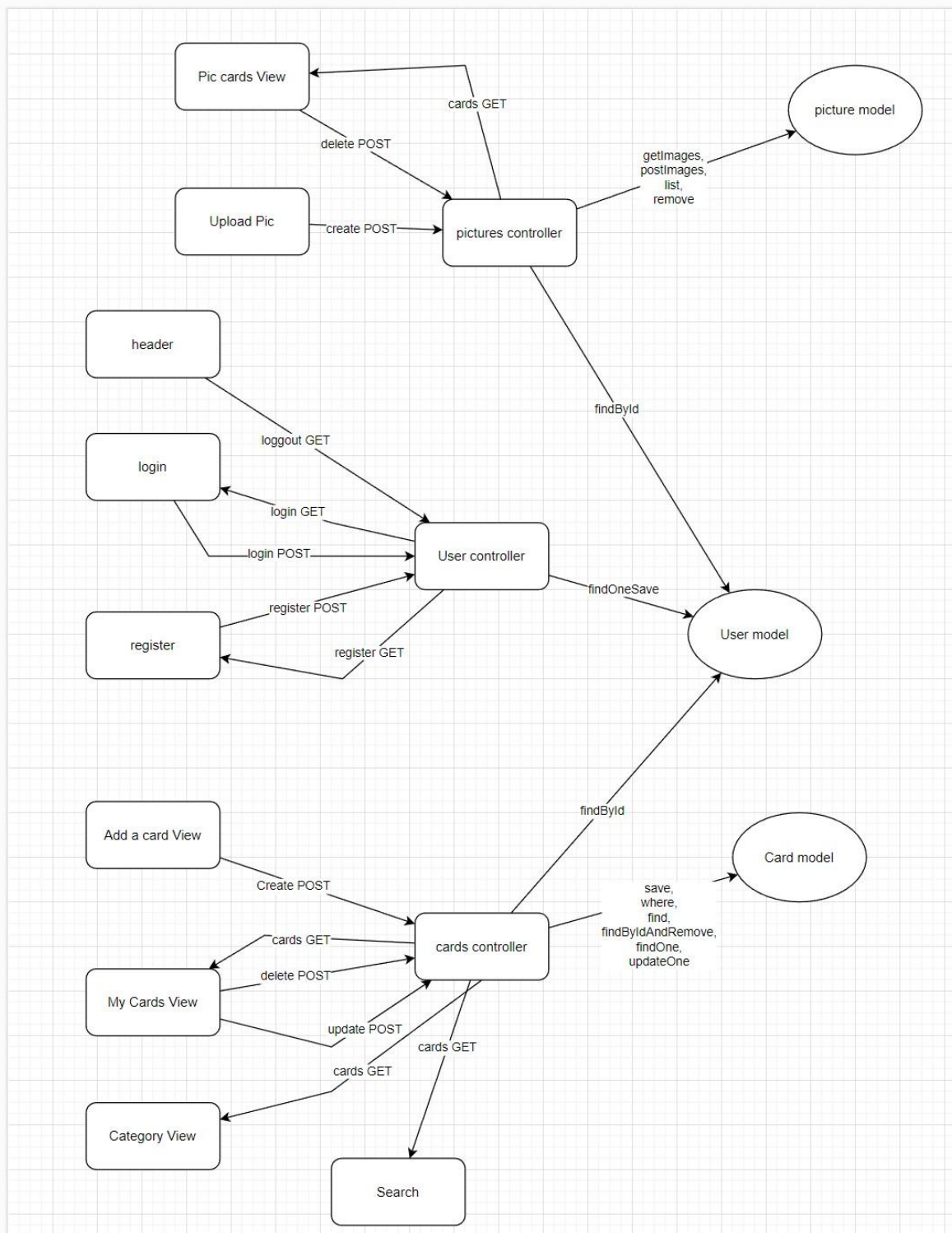
The database of choice was the non-relational database, also called NoSQL MongoDB. The underlying reason is the many advantages of NoSQL, one of the key points of which is scalability. NoSQL engines are built to scale up and take use of cloud computing. When we scale out or horizontally, we add resources to a single node (a computer or server). We can run a single database on several nodes. We can easily add and remove

nodes when we scale out. As a result, NoSQL is an excellent choice for the cloud. It can be utilising the scalability of the cloud benefits since it can expand out.

Another important aspect is the performance and flexibility, being not constrained to pre-defined rules (Integrant 2021) and enabling to creation of different data types or entries to the database on the fly.

The models and design of the content of the database as well as the controllers to govern the data were done by using mongoose.

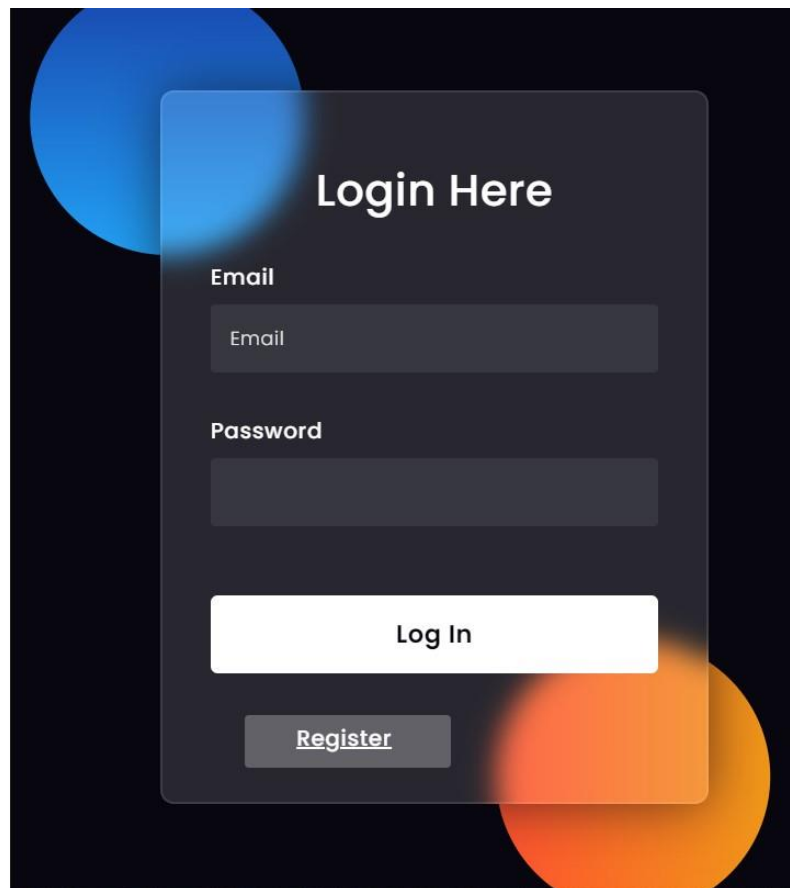
System diagram of collaborative effort of the the whole MVC structure.



## Users:

Starting with the users is a crucial part of the application since all users have their very own custom-designed data without having access to someone else's. The users can log in or register. In the case of a register, the program uses the hashing algorithm bcrypt that turns the password

into a long string before uploading that into the database. After login, the id of the user is stored in a session variable that enables access to each controller.



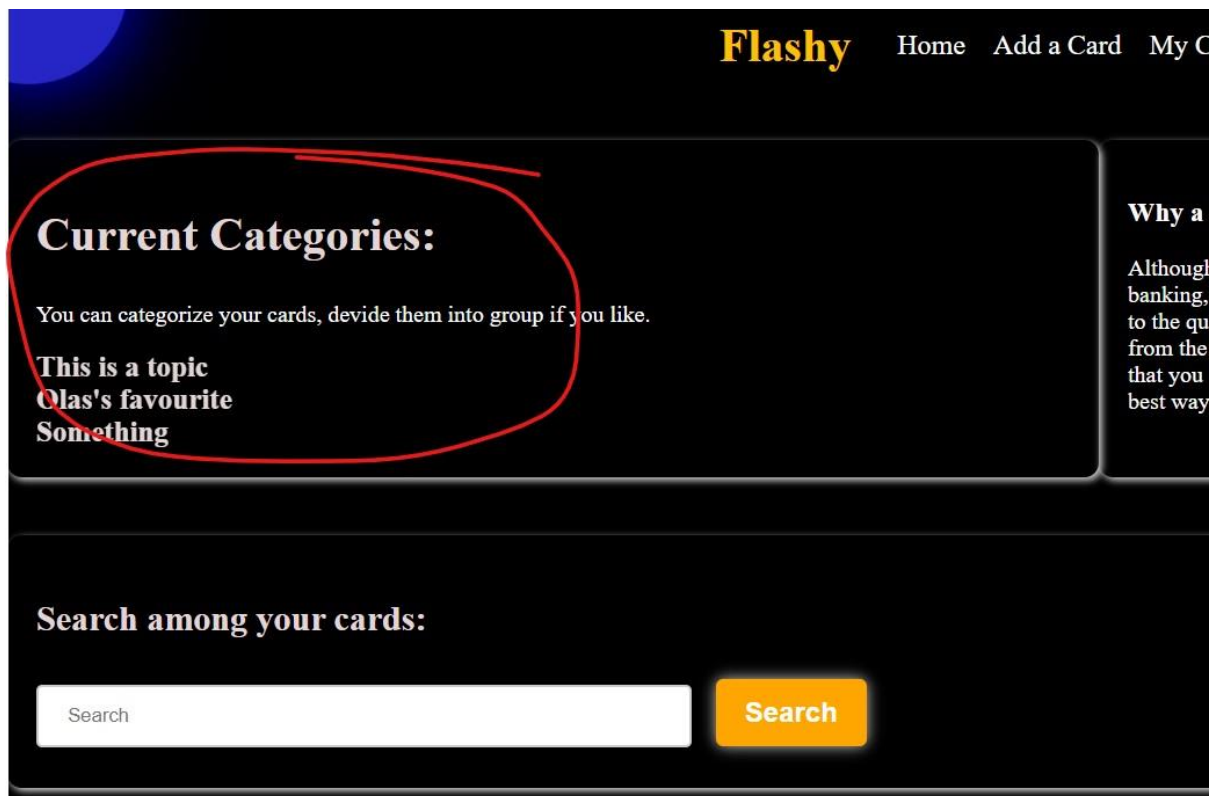
The main parts of the application are accessible from the navbar in the header which is incorporated into all pages.



### Categories:

First is the home page where the Categories and the Search are exclusively available.

Each card when created or edited can be placed into a category which facilitates the distinction between different subjects and topics. All existent categories are highlighted on the home page as shown in the picture. Click, and all the cards that belong to that particular category will be displayed.



### Search:

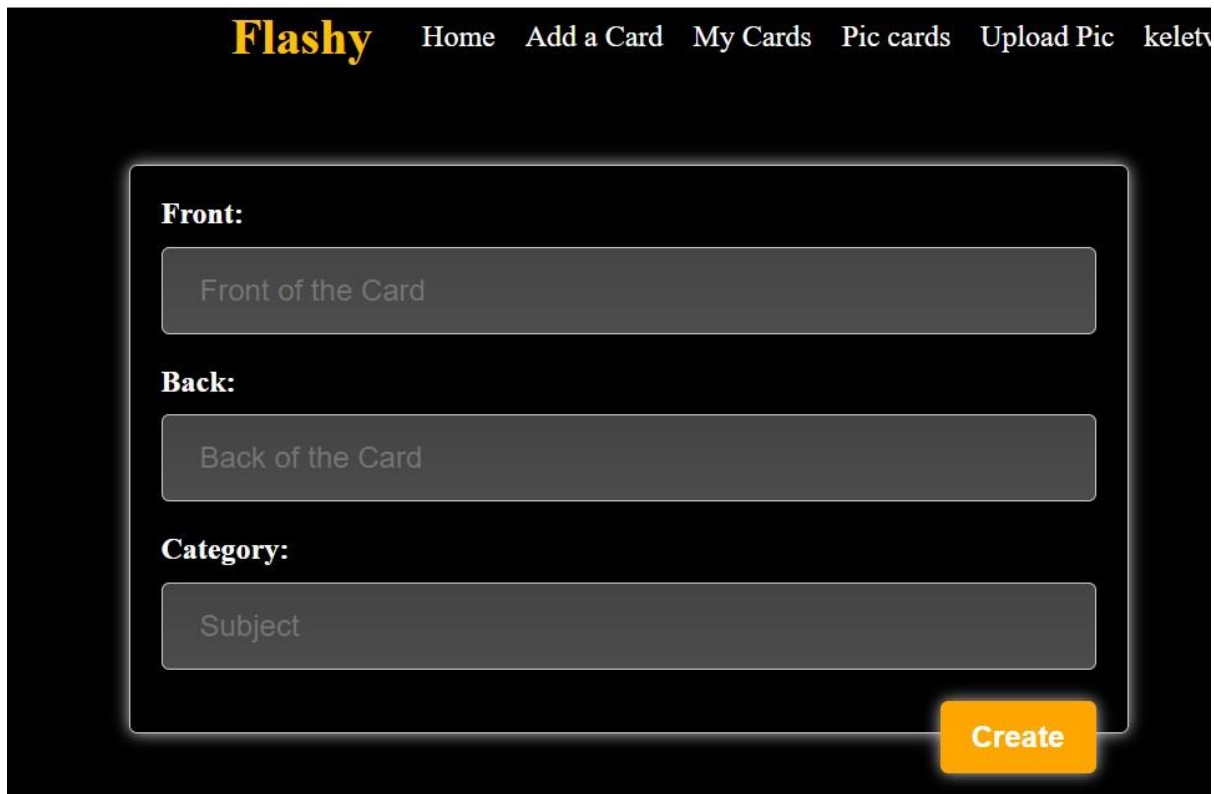
Search also opens from the home page. It works by querying all content of the cards both front and back and all the cards that contain the matching word will be displayed.



### Adding a Card:

Moving forward, next is Add a Card, which is self-explanatory. Users can add a limitless amount of text to the back and the front and can also

define its category of it. After clicking the create button the user gets redirected to the My Cards page.

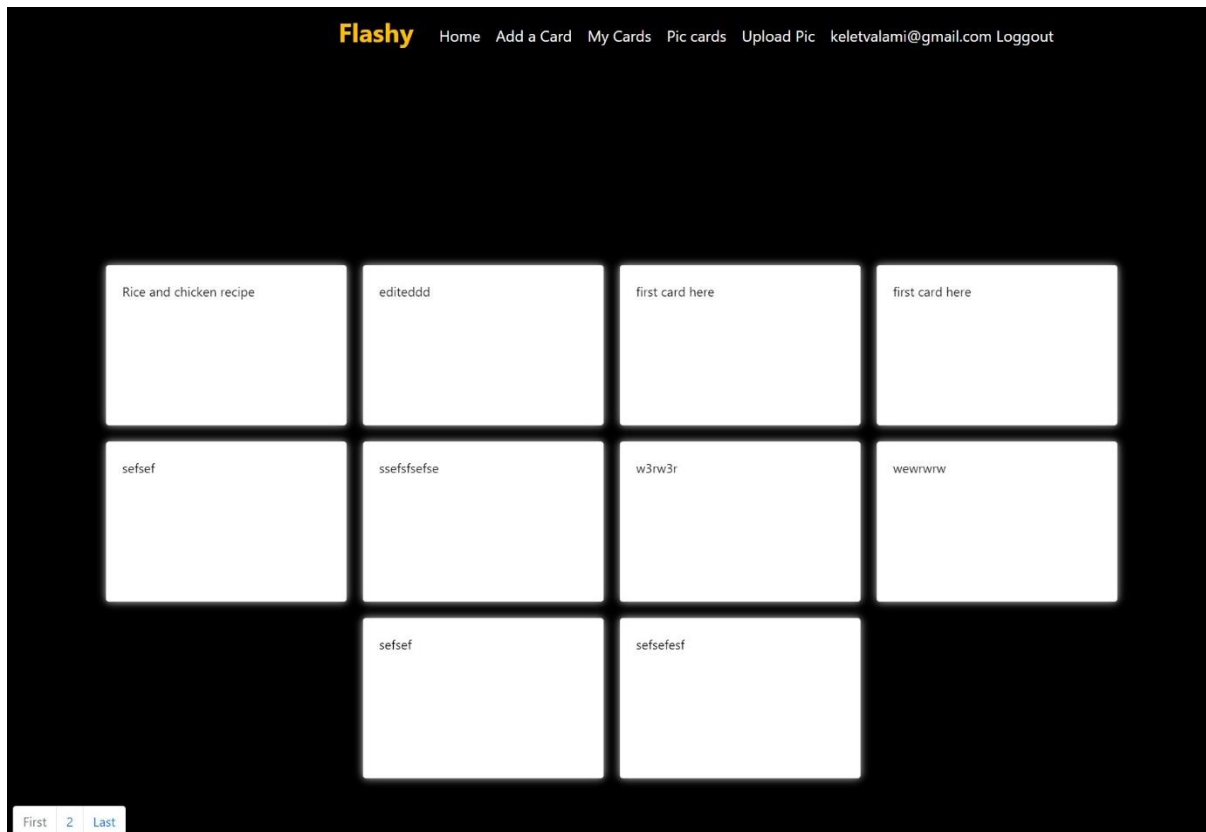


The screenshot shows a web application interface with a dark background. At the top, there is a navigation bar with the logo 'Flashy' in yellow and orange, and several menu items: 'Home', 'Add a Card', 'My Cards', 'Pic cards', 'Upload Pic', and 'keletv'. Below the navigation bar, there is a form for adding a new card. The form is enclosed in a rounded rectangle with a light gray border. It contains three text input fields: 'Front:' with placeholder text 'Front of the Card', 'Back:' with placeholder text 'Back of the Card', and 'Category:' with placeholder text 'Subject'. A yellow 'Create' button is located at the bottom right of the form.

### My Cards:

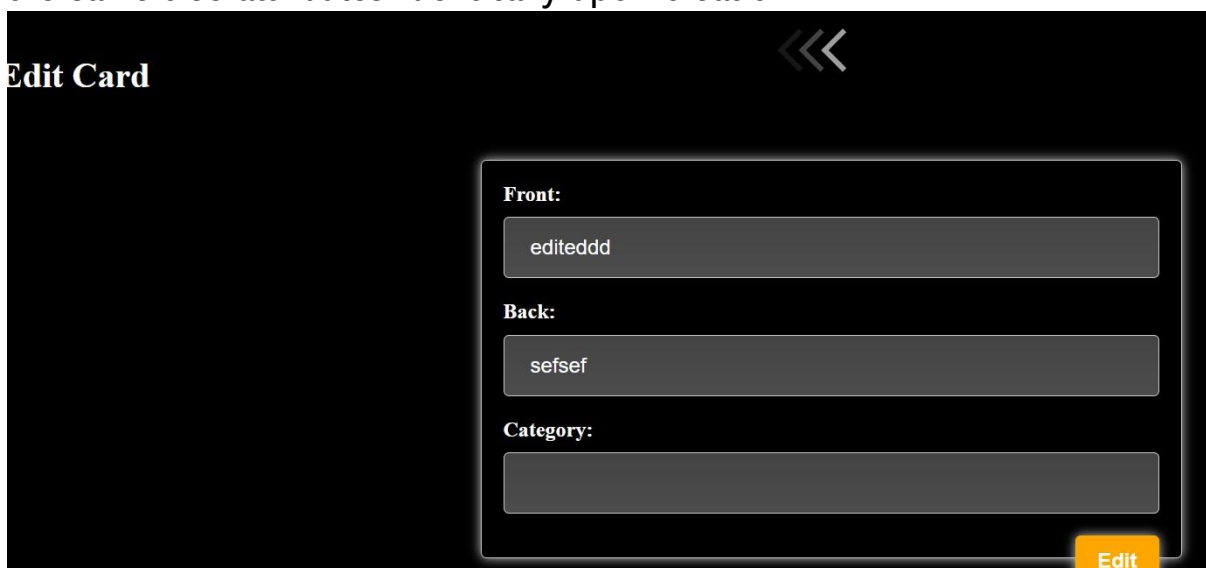
Expectedly the most used page of the whole application, where all the cards of the user get displayed. Due to the anticipated multitude of cards, the page was paginated with Mongoose with a bootstrap layout. What it entails is that not all cards will be loaded at once from the database, by using `skip()` and `limit()`. The current limit is 10 cards all at once on a single page.

The default view of each card is the front. It can be flipped, to reveal the back of the card just by simply clicking on them. The size of the cards is mediocre, and if it contains more text than is visible, a scroll bar will appear to help the overfitting rather than increasing the size. To not block the view, on hover the edit and delete button appears.



Deleting a card is as simple as pressing the delete button. Once it is clicked it deletes that document from the database.

Pressing on edit, the program grabs the individual object id and renders that object with its attributes to the editing page, where the user can edit the same tree attributes identically upon creation

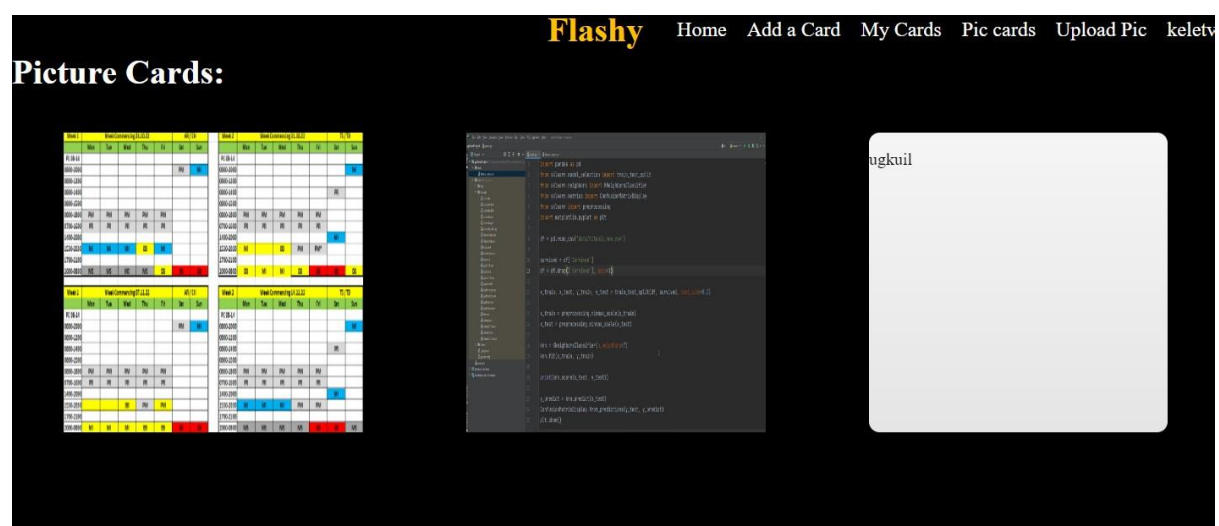


In My Cards, the first is always the one that was edited the last!



## Pic Cards:

Next is the Pic Cards, where all the cards containing a picture on the back gets displayed. Uploading a picture was implemented using a Node js middleware multi (npm 2019). I work with a smaller image, under 16 MB, that is stored directly in a document using BinData(binary data) in a BSON type, and saved into the file system (uploads folder) with only the image reference stored in the database. The layout is very similar with the slight difference being the lack of the edit button and the dynamically adjusting size of the cards.



The creation of the cards with a picture on them happens on the Upload pic. When uploading mime type makes sure that the uploaded file is an image, preventing any other type of file to enter the file system or the database (Webplatform.org 2003).

The screenshot shows the 'Flashy' application interface for creating a card with a picture. The navigation bar includes 'Home', 'Add a Card', 'My Cards', 'Pic cards', 'Upload Pic', and 'keletvalami'. The main heading is 'Create a card with a Picture'. Below this, there is a form with the following elements:

- Text:** A label above a text input field containing the word 'Name'.
- Upload Image** *Images only!*: A label above a file upload area.
- Choose file**: A button next to the text 'No file chosen'.
- Submit**: An orange button.

# Key Decisions

## MVC stack:

```
Advanced_database
|   .env
|   .gitignorer
|   app.js
|   README.md
|   report.md
|   package.json
├── models
|   |   Card.js
|   |   picture.js
|   |   User.js
├── controllers
|   |   cards.js
|   |   pictures.js
|   |   user.js
├── views
|   └── common
|       |   header.ejs
|       |   category.ejs
|       |   creatingUser.ejs
|       |   editingCard.ejs
|       |   forCard.ejs
|       |   index.ejs
|       |   loginUser.ejs
|       |   pictures.ejs
|       |   search.ejs
|       |   usersCards.ejs
|       |   viewPictures.ejs
├── public
|   ├── images
|   └── scripts
|       |   forCards.css
|       |   header.css
|       |   index.css
|       |   usersCards.css
|       |   viewPictures.css
└── uploads
```



## Database Design:

The whole application runs on one mongo database and three collections featuring the cards, images and users.

## Users collection:

The collection consists of the following values.

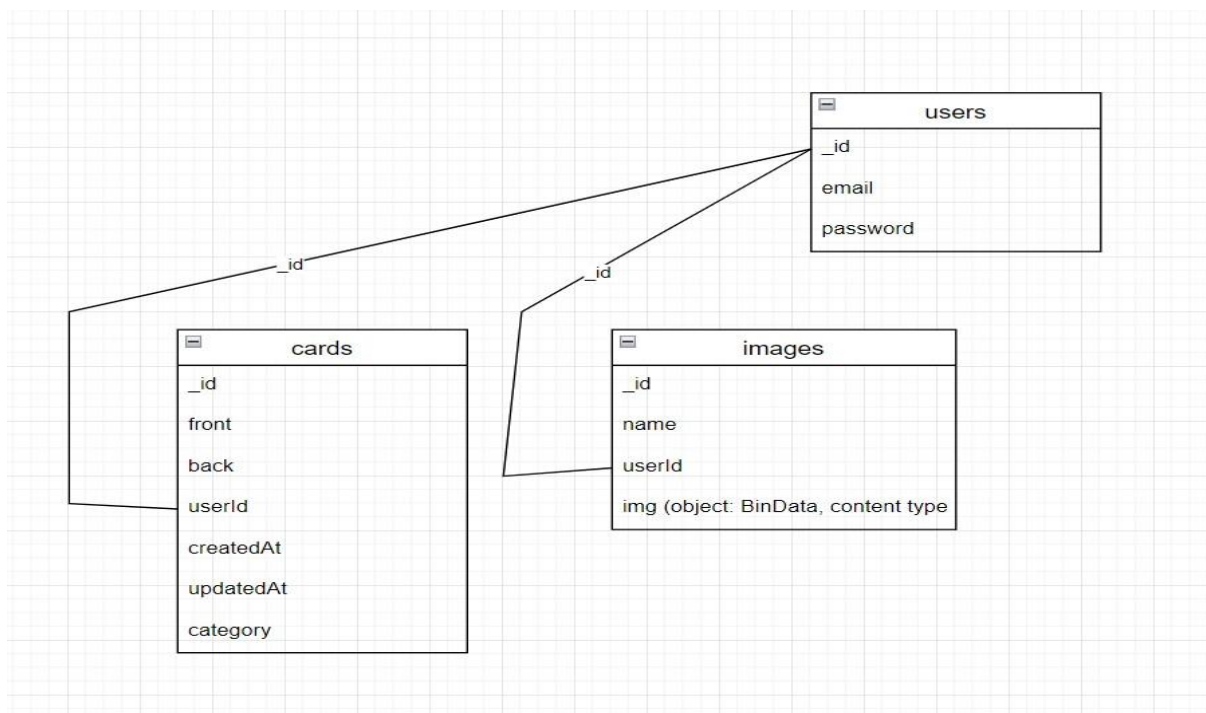
```

_id: ObjectId('63bfee67e1239d4598e74030')
email: "donboysz@gmail.com"
password: "$2b$10$XGtXwIUFCmZLLTtP8X.8be.TjMzRwtNzBu2ww9Mt7IKpqHzzaEGJm"
createdAt: 2023-01-12T11:26:31.263+00:00
updatedAt: 2023-01-12T11:26:31.263+00:00
__v: 0

```

The id plays a crucial role since it is passed to each card that the user creates, enabling the whole application to be user-specific! Emails and password for validation. Note that the password is in the already hashed form.

The class diagram demonstrates the relation between the collections.



### Cards collection:

```

_id: ObjectId('63c3466452d4594fdcdfd527')
front: "sefsefesf"
back: "sefsfsef"
userId: "63bfe265b6619b5bf0b3cd0c"
createdAt: 2023-01-15T00:18:44.397+00:00
updatedAt: 2023-01-15T00:18:44.397+00:00
__v: 0
category: "This is a topic"

```

Each card document has its front and back values that are compulsory and the hidden populated userId that again allow the entire program to be user specific. Then comes the time of creation and update. Note that whenever

a card gets updated it becomes the first in the My Card view. Lastly, the optional category value enables the user to aggregate all the cards within a certain subject.

Images collection:

```
_id: ObjectId('63c48f511db6f857ec3c20b9')
name: "fuck you "
userId: "63bfe265b6619b5bf0b3cd0c"
✓ img: Object
  data: BinData(0, 'iVBORw0KGgoAAAANSUgAAA4AAAAAGpCAYAAADcCFiAAAAAXNSR0
  contentType: "image/png"
  __v: 0
```

The document is similar to the text cards, each contains a front or name in this case the userId to personalise it and lastly the Bin Data representation of the image as well.

## Security and Scalability

### Security:

Although the whole application is built on personal data, the only sensitive data is the email address and password. Hence the password only enters the database in a hashed format, which was made possible by using Bcrypt. Bcrypt uses a 128-bit salt and encrypts a 192-bit magic value (Usenix 2008). Although it is not the safest option out there, it sure makes the attacks and brute force guessing hard (Compose 2016). Nonetheless, the security could be improved in any way by using biometric and or multi-factor authentication or outsourcing the login/signup process to bigger companies like google (login radius 2020). Besides this, as mentioned already all created or uploaded data is user-specific, so the only person who has access to them is the person who created them.

### Scalability:

Scalability is the ability to extend or contract the capacity of system resources to accommodate your application's changing use. This can relate to both growing and declining application utilisation (MongoDB 2021). In terms of scalability, our application's MongoDB database is already

sharded horizontally over three nodes inside our European Union deployment area. By dividing our data over different nodes, sharding improves database throughput. However, this is not the only way our database may be scaled. Vertical scaling refers to increasing the processing power of a single server or cluster. Both relational and non-relational databases can scale up, but eventually, there will be a limit in terms of maximum processing power and throughput (MongoDB 2021b). Historically, database scalability has been a key setback for big systems or applications with above-average throughput, with choices either restricted in number or prohibitively expensive to execute. MongoDB, on the other hand, provides a comprehensive set of scalability options .

## Conclusion and Reflection

This project is very dear to me, and the concept has profoundly helped my life was brought into life. No need to mention that if I made a flash card application, this is what it would look like. Nonetheless, due to insufficient time, the application is only a proof of concept. If I had to quantify the process, I would say around 30-35, 40% of the original plan is left to be implemented. It has its limitations. The GUI, even though every bit of it is custom and handmade, the responsiveness on smaller devices leave room for improvement. The user database could be expanded with the customer name to further tailor it to the comfort of the user. One point that is sort of obvious and will be rectified in the near future, is the mechanic of the cards with a picture on them. In their current state, the normal text cards and the ones with images do not mingle, completely separated. That will be changed, but it requires time! Another feature in mind that will be added later on to it, is a grading system for each card, where the user can grade himself based on his knowledge, which allows him to retrieve his performance and gives him a clear picture of where to make improvements and what areas are his strength. If had an opportunity to start over, probably I would prioritise the grading system over the layout of the application.

Overall, while significant progress has been made, there is still work to be done to bring the flashcard application to its full potential. Many conceptual targets were hit, and the application is fully mutable by any user and every bit of data will remain his/her digital property. Flashy does not depend on data to be useful, and though it could later be monetized it

is not only a product but a tool whose degree of usefulness solely depends on the user. In its current state perfectly represent a working beta, with a clear picture of what it will be like once it is done!

Fatal error in cyclic!!! After deploying the application uploading any file or picture will result in an error in the deployed version due to the incapability of cyclic of writing to the server any kind of static asset. It is not only immensely slow but limited! As, requested in teams the showcase that the application actually is working fine in local plus the picture upload.



The Pic cards view above and the flipped version below.

**Picture Cards:**

a cat

**Picture Cards:**

Crucial please use the Account: [example@example.com](mailto:example@example.com) pw: example

To see the cards with images. And please Clone the repo and run it locally to see the full program! Many time and effort was put in it! Thank you!

## References

- COMPOSE, 2016. *You May Get Pwned! At Least Protect Passwords with Bcrypt*. [online] [viewed 18 Jan 2023]. Available from: <https://www.compose.com/articles/you-may-get-pwned-at-least-protect-passwords-with-bcrypt/>
- FUTURELEARN, 2022. *Top 10 Study Tips for Effective Learning* [online] Available from: <https://www.futurelearn.com/info/blog/top-study-tips-for-effective-learning>
- GEEKS FOR GEEKS, 2021. *Benefit of Using MVC* [online] Available from: <https://www.geeksforgeeks.org/benefit-of-using-mvc/>
- HELLER, M., 2020. *What Is Node.js? The JavaScript Runtime Explained* [online] Available from: <https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html>
- INTEGRANT, 2021. *When to Use SQL vs. NoSQL* [online] Available from: <https://integrant.com/blog/when-to-use-sql-vs-nosql/>
- LOGINRADIUS, 2020. *Login Security: 7 Best Practice for Online Security | LoginRadius Blog* [online] Available from: <https://blog.loginradius.com/identity/login-security/>
- MONGODB, 2021a. *How to Scale MongoDB* [online] Available from: <https://www.mongodb.com/basics/scaling>
- MONGODB, 2021b. *How to Scale MongoDB* [online] Available from: <https://www.mongodb.com/basics/scaling>
- NPM, 2019. *Multer* [online] Available from: <https://www.npmjs.com/package/multer>
- SIX BENEFITS OF USING MVC MODEL FOR EFFECTIVE WEB APPLICATION DEVELOPMENT, 2016. *Six Benefits of Using MVC Model for Effective Web Application Development* [online] Available from: <https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/>
- USENIX, 2008. *Bcrypt Algorithm* [online] [viewed 12 Jul 2022]. Available from: [https://www.usenix.org/legacy/publications/library/proceedings/usenix99/full\\_papers/provos/provos\\_html/node5.html](https://www.usenix.org/legacy/publications/library/proceedings/usenix99/full_papers/provos/provos_html/node5.html)
- WEBPLATFORM.ORG, 2003. *Introduction to MIME Types · WebPlatform Docs* [online] [viewed 18 Jan 2023]. Available from:



[https://webplatform.github.io/docs/concepts/Internet\\_and\\_Web/mime\\_types/#:~:text=MIME%20types%20enable%20browsers%20to](https://webplatform.github.io/docs/concepts/Internet_and_Web/mime_types/#:~:text=MIME%20types%20enable%20browsers%20to)