



# Senior Design Project

## “Bumpster”

### *Analysis Report*

Supervisor: İbrahim Körpeoğlu

Jury Members: Bülent Özgüç  
Uğur Güdükbay

Innovation Expert: Armağan Yavuz

Project Website: [www.bumpster.me](http://www.bumpster.me)

Group Members: Duygu Durmuş  
Gülsüm Güdükbay  
Doğukan Yiğit Polat  
Muhammed Çavuşoğlu  
Bora Bardük

November 6, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

## Table of Contents

<b>1. Introduction</b>	<b>4</b>
<b>2. Current Systems</b>	<b>5</b>
3.1 CrazyBump	5
3.1.1 Differences	5
3.2 Substance B2M	5
3.2.1 Differences	5
3.3 ShaderMap	6
3.3.1 Differences	6
<b>3. Proposed System</b>	<b>7</b>
3.1 Overview	7
3.2 Requirements	8
3.2.1 Functional Requirements	8
3.2.2 Non-functional Requirements	9
3.2.2.1 Compatibility	9
3.2.2.2 Performance	9
3.2.2.3 Adaptability	9
3.2.2.4 Reliability	9
3.2.2.5 Scalability	9
3.2.2.6 Flexibility	9
3.2.2.7 Maintainability	9
3.2.2.8 User Friendly Interface	10
3.2.3 Pseudo Requirements	10
3.2.3.1 Implementation Requirements	10
3.2.3.2 Economic Requirements	10
3.2.3.3 Language Requirements	10
3.2.3.4 Ethical Requirements	10
3.2.3.5 Privacy and Security Requirements	10
3.2.3.6 Confidentiality Requirements	11
3.2.3.7 Maintenance Requirements	11
3.2.3.8 Time Requirements	11
3.3 System Models	11
3.3.1 Scenarios	11
3.3.1.1 Scenario 1 - Create User	11
3.3.1.2 Scenario 2 - Login User	12
3.3.1.3 Scenario 3 - Generate Maps	12
3.3.1.4 Scenario 4 - Edit Model	13

3.3.1.5 Scenario 5 - View Models on Mobile	13
3.3.1.6 Scenario 6 - Capture Single Photo Via Mobile and Generate Maps	14
3.3.2 Use Case Model	15
3.3.3 Object and Class Model	16
3.3.3.1 Data	17
3.3.3.2 InputWrapper	17
3.3.3.3 Output Wrapper	18
3.3.3.4 User	19
3.3.3.5 Login Session	20
3.3.3.6 Map	20
3.3.3.7 Input Visual	21
3.3.3.8 Model	22
3.3.3.9 Saved Model	22
3.3.3.10 Database Manager	23
3.3.3.11 Neural Network	25
3.3.3.13 Input Function	26
3.3.4 Dynamic Models	27
3.3.4.1 State Diagrams	27
3.3.4.1.1 User State Chart	27
3.3.4.1.2 Model State Chart	28
3.3.4.2 Sequence Diagrams	29
3.3.4.2.1 Scenario 1 Sequence Diagram	29
3.3.4.2.2 Scenario 2 Sequence Diagram	30
3.3.4.2.3 Scenario 3 Sequence Diagram	31
3.3.4.2.4 Scenario 4 Sequence Diagram	32
3.3.4.3 Activity Diagram	33
3.3.5 User Interface	35
3.3.5.1 Mobile Application	35
3.3.5.1.1 Sign In Page	35
3.3.5.1.2 Sign Up Page	36
3.3.5.1.3 Main Page	37
3.3.5.1.4 Edit Page	38
3.3.5.1.5 Export Page	39
3.3.5.1.6 Camera Page	40
3.3.5.1.7 Profile Page	41
3.3.5.2 Desktop Application	42
3.3.5.2.1 Sign In Page	42
3.3.5.2.2 Sign Up Page	43
3.3.5.2.3 Main Page	44

3.3.5.2.4 Upload Page	45
3.3.5.2.5 Export Page	46
3.1 CrazyBump	47
3.1.1 Differences	47
3.2 Substance B2M	47
3.2.1 Differences	47
3.3 ShaderMap	48
3.3.1 Differences	48
<b>4. References</b>	<b>49</b>

# 1. Introduction

There are several types of mapping targeting to different aspects of material. Bump mapping is one of these techniques used in computer graphics which enables realistically rendering bumps and wrinkles on the surface of a model. Metalness, gloss and roughness maps are specular maps which make use of the reflectivity of the surface. In detail, Metalness map considers the amount of reflected light by evaluating materials as metallic or nonmetallic while gloss map specializes on how wide or narrow the reflected light appears on the surface and roughness map controls how smooth or rough the texture of surface is. Ambient map is a light map which includes calculations of the bounciness of ambient light on the surface. The appearance of textured surface can be obtained by simply using RGB or grayscale values to define dark parts as deepened and bright parts as heightened areas [1]. The shading effect and tiny details on the surface creates an illusion of depth which is useful for graphic designers and game artists to get realistic surfaces of their models.

The existing systems based on texturing software such as CrazyBump and nDo2 are good at exporting different kinds of maps from photographs or scanned images [2]. Getting good results is possible, but the accuracy is not high. The map types that will be generated in this software will include bump, metalness, roughness, gloss or ambient maps. The results can be customized with photoshop features of these texturing software programs but it also takes time to get realistic textured surface of a model. In addition, existing texture software programs are not available in mobile platforms so reaching wide range of customers and accessing benefits of these programs are impossible. Therefore, there is a need for creating bump maps in different platforms with higher accuracy which also reduces the necessity of additional customization and altering the maps created.

## 2. Current Systems

Currently available systems are described in following subsections.

### 3.1 CrazyBump

CrazyBump is a tool that generates normal map from an image. Users can change the intensity of the normal map, sharpen it and remove noise. It's useful for creating textures from a single image [3]. CrazyBump is offered at \$299 for professional use, \$99 for personal use and \$49 for students[4].

#### 3.1.1 Differences

- CrazyBump's detection capabilities are limited to Shape Recognition using mainstream image processing techniques (no machine learning elements involved). Our application will use Deep Learning to create 3D models. This allow us to have a few advantages over systems like CrazyBump like extracting metalness, roughness, gloss or ambient maps of the input from the given image, as well as more accurate bump map.
- Bumpster has the ability to use multiple images of the desired surface. This allows Bumpster to increase the accuracy of extraction.
- CrazyBump does not support multiple platforms, while the platform support Bumpster offers is: Mac, Windows, Linux, iOS, Android.
- CrazyBump's user interface is not user friendly, looks outdated and has limited model manipulation capabilities.

### 3.2 Substance B2M

Substance B2M (Bitmap2Material) is a tool that generates physically based rendering outputs from a single image based on its base color, metallic properties and roughness [5]. Substance B2M is offered at different prices based on the revenue of the customer [6]. Substance B2M is more comprehensive compared to CrazyBump, and it has a more user friendly interface.

#### 3.2.1 Differences

- Substance B2M also lacks the high-level model manipulation capabilities, instead provides low level controls which makes it hard to obtain the desired manipulation to the model.
- Bumpster has the capability to extract metalness, roughness, gloss or ambient maps of the surface in the images(s) as well.
- Bumpster promises more precise and accurate solution since it uses Deep Learning.

- Substance B2M does not support multiple platforms, while the platform support Bumpster offers is: Mac, Windows, Linux, iOS, Android.

### 3.3 ShaderMap

ShaderMap is a tool that generates various types of object maps from four images taken and inputted to the program in a special lighting condition. There are a number of default mapping support which includes: displacement from diffuse, albedo from diffuse, normal from displacement, ambient occlusion from 3d model, displacement from 3d model, normal from normal[7].

#### 3.3.1 Differences

- While ShaderMap has the capability to generate the types of mapping we offer in Bumpster, it lacks the capability to do so using a single image. This is because ShaderMap processes the image using image processing techniques, and not intelligently recognizing surfaces.
- ShaderMap requires the input of four photos taken in professional photography medium. Bumpster is capable of acquiring the required data in a single photo, but offers the option to use multiple images to further improve its results. Moreover, it does not require the photo to be taken in a photography studio with special equipment since it will recognize the materials.
- ShaderMap does not support multiple platforms, while the platform support Bumpster offers is: Mac, Windows, Linux, iOS, Android.

## 3. Proposed System

The sections below include the overview of the system, functional requirements, nonfunctional requirements, pseudo requirements and system models.

### 3.1 Overview

Our software will generate bump and physically based rendering maps using a specifically trained neural network. It will be able to identify the topological properties of the surface, as well as the material properties using a single photograph. Our product will be the used in the server to make the required operations, and communicate with the front-end applications on various platforms. We believe our software will provide rather large benefits for the industry, as the previous solution to this kind of topographical analysis required expensive methods which require personnel with expertise, expensive equipment, and time [8].

Since there are no other tools that extract physically based rendering (bump, metalness, roughness, gloss or ambient) maps with one or more photos using a neural network, this application will be beneficial in terms of accuracy and usability. After the extraction of the user selected mapping, the user will be able to obtain the resulting map and fit it onto the original photo to see the material properties or bump to create a realistic image. Then, the artists can tweak the chrominance, roughness and other properties of the final product and save it for later usage.



## 3.2 Requirements

The functional, non-functional and pseudo requirements are listed below in separate subsections below.

### 3.2.1 Functional Requirements

- The software will generate the physically based rendering maps from one or more photos taken by the user.
- The extracted maps will include glossiness, cavity, metalness and other physical properties from user specified photo.
- The application will be able to identify which parts of the photo are metal and other materials, using deep learning.
- The chrominance values of certain surfaces will also be determined by the network, which is an advantage over Bitmap2Material, since it is not able to determine only chrominance values.
- The data to train the neural network will be generated from computer rendered scenes in which the bump, metalness, roughness, gloss or ambient maps are extracted. This process is usually challenging, since these maps are created by graphic designers manually, and sometimes using special equipment.
- The software will contain a neural network of map-surface image pairs which will be calibrated.
- There will be a cross platform mobile application that will allow the user to send photo(s) of the textures in which they want map extraction to be performed on, display extracted maps, edit properties of extracted maps, browse an archive of saved maps, and export these maps.
- The input photo(s) will be sent to the server that performs the extraction, and then the server will send the map back to the application.
- Users can view the end map on their phone or PC and they can overlay the map with initial photo of texture to see the realistic image.
- Users can view the end map in many different angles in mobile and desktop applications.
- Users (especially graphic artists) will be able to tweak different properties of the generated map such as metalness and glossiness.
- Users will be able to save the extracted map.

## 3.2.2 Non-functional Requirements

### 3.2.2.1 Compatibility

Software will support the up to date versions of the mobile-desktop front end applications created for it. The application will be compatible with iOS, Android, Windows and MacOSx operating systems, because it will be developed in Unity, which is a cross platform engine.

### 3.2.2.2 Performance

Software needs to answer the peer request as fast as possible to ensure best possible response and move on to process future requests in time and this will be possible with the efficient algorithms and connections used between client and the server. Also, the database will be used efficiently and therefore inserting, deleting and updating it will be easy in terms of time cost.

### 3.2.2.3 Adaptability

The software will be adaptable to a certain degree and will support the addition of different technologies using the software such as new editing properties.

### 3.2.2.4 Reliability

Software should minimize the chance of failures like unterminating requests and crashes by handling all of the special cases. NULL values will be handled correctly in the database and error handlers for the server will be used.

### 3.2.2.5 Scalability

The software should be prepared for an increase in requests (users) and possible change at hardware scale which will provide processing power on a larger scale. The server-client architecture will be able to supply the increasing demand.

### 3.2.2.6 Flexibility

Because of the fact that users (especially graphical designers) always increase the realisticness of their works as their clients desire, their needs from these kinds of applications always increase to keep up with their needs and to make their lives easier.

### 3.2.2.7 Maintainability

The operating systems such as iOS, Android, Windows, MacOSx and Linux always improve via their updates. Our application will be able to keep up with the operating systems by the aid of its developers adding updates and also via the fact that the application is cross platform.

#### 3.2.2.8 User Friendly Interface

User friendly interface is one of the most important non-functional requirements of our software, because it builds a bridge between the computer graphics and image processing tools and the graphic designers. An easy and understandable GUI for the desktop and mobile app are our aim. This GUI will ensure the easy tweaking of the preferences of the map generated and even view it from different angles.

#### 3.2.3 Pseudo Requirements

The implementation, economic, language, ethical, privacy, security, confidentiality, maintenance and time pseudo requirements are listed below.

##### 3.2.3.1 Implementation Requirements

- The application will be available for iOS, Android, Windows, MacOSx and Linux platforms.
- The programming languages used in the application will be Python, C++ and C#.
- For version control, GitHub will be used.
- TensorFlow library will be used for the machine intelligence portion of the project.
- MySQL will be used for application database.
- Unity will be the environment in which the project will developed on, for the purpose of cross platform feature.

##### 3.2.3.2 Economic Requirements

The data which will be used to train the neural network will be acquired from trusted sources, as well as from software-rendered scenes which will automatically generate the corresponding bump map of the image during creation. We are planning to use Google's TensorFlow library, which is an open-source software library for Machine Intelligence [4].

##### 3.2.3.3 Language Requirements

The language of this application will be English, since it is a universally accepted and used language.

##### 3.2.3.4 Ethical Requirements

The users' photographs and scanned images will not be shared with any third-parties.

##### 3.2.3.5 Privacy and Security Requirements

- The photos that users upload are seen only the users themselves.

- The client part's communication with the server will be secure in order to preserve the privacy and the security of the users' data.
- User's password and usernames are kept secure in the database, where no unauthorized person can access.

#### 3.2.3.6 Confidentiality Requirements

- The maps generated in the application and the settings of every user will be kept confidential.
- The source codes of the application will be kept confidential.

#### 3.2.3.7 Maintenance Requirements

The neural network contained within the software will be updated and extended to recognize more types of materials and surfaces.

#### 3.2.3.8 Time Requirements

The application will be completed and tested by May 2018.

## 3.3 System Models

### 3.3.1 Scenarios

#### 3.3.1.1 Scenario 1 - Create User

**Use Case Name:** CreateUser

**Actors:** User

**Entry Conditions:**

- User opens the app.
- User chooses to create a user account.

**Exit Conditions:**

- User creates an account.
- User closes the app.

**Main Flow of Events:**

- User opens the app.
- Bumpster checks if a user is logged in, displays sign-in/sign-up screen.
- User presses on Create Account button.
- User is redirected to sign-up screen.
- User inputs account information.
- User submits the account details.

- User waits for a response.
- Bumpster informs user about the success of the sign-up operation.
- If a successful sign-up occurs then scenario ends, else user may choose to re-enter the details and gets redirected to the sign-up screen again.

### 3.3.1.2 Scenario 2 - Login User

**Use Case Name:** LoginUser

**Actors:** User

**Entry Conditions:**

- User opens the app.
- User is on login screen or user is already logged in.

**Exit Conditions:**

- User logs in.
- User closes the app.

**Main Flow of Events:**

- User opens the app.
- Bumpster checks if a user is logged in, displays sign-in/sign-up screen.
- If user already entered his/her details before, Bumpster logs in automatically. Login flow ends here in this case.
- User enters credentials and presses on Login button.
- Bumpster tries to log-in with provided details and informs user about the success of the sign-in operation.
- If a successful sign-in occurs then login flow ends, else user may choose to re-enter the details and gets redirected to the login screen again.

### 3.3.1.3 Scenario 3 - Generate Maps

**Use Case Name:** GenerateMaps

**Actors:** User

**Entry Conditions:**

- User is logged in.
- User have chosen bitmap image(s) to generate PBR maps of.
- User chooses to generate PBR maps of the image(s).

**Exit Conditions:**

- Maps are generated.
- Maps could not be generated.
- User closes the app.

**Main Flow of Events:**

- User presses generate maps button.
- User is redirected to the results page, if a failure occurs then user is informed about the failure and gets redirected to the image selection scene.
- User views the rendered result that is generated with the PBR maps.

#### 3.3.1.4 Scenario 4 - Edit Model

**Use Case Name:** EditModel

**Actors:** User

**Entry Conditions:**

- User is logged in.
- User have the resulting PBR maps and is on the results screen.

**Exit Conditions:**

- User is done with editing.
- User closes the app.

**Main Flow of Events:**

- User interacts with UI tools to tweak the resulting image.
- User gets instant feedback by examining the rendered preview.
- User saves the tweaked model.

#### 3.3.1.5 Scenario 5 - View Models on Mobile

**Use Case Name:** ViewModelsOnMobile

**Actors:** User

**Entry Conditions:**

- User opens the app
- User logs in if not logged in
- User chooses to view models.

**Exit Conditions:**

- User closes the app.
- User decides to do something else.

**Main Flow of Events:**

- User opens the app.
- Bumpster checks if a user is logged in.
- User browses through his/her previously saved model.
- User presses on the model which he desires to view.
- User is redirected to the rendered preview screen.

### 3.3.1.6 Scenario 6 - Capture Single Photo Via Mobile and Generate Maps

**Use Case Name:** CaptureSinglePhotoViaMobileAndGenerateMaps

**Actors:** User

**Entry Conditions:**

- User opens the app.
- User chooses to generate maps with a single photo.

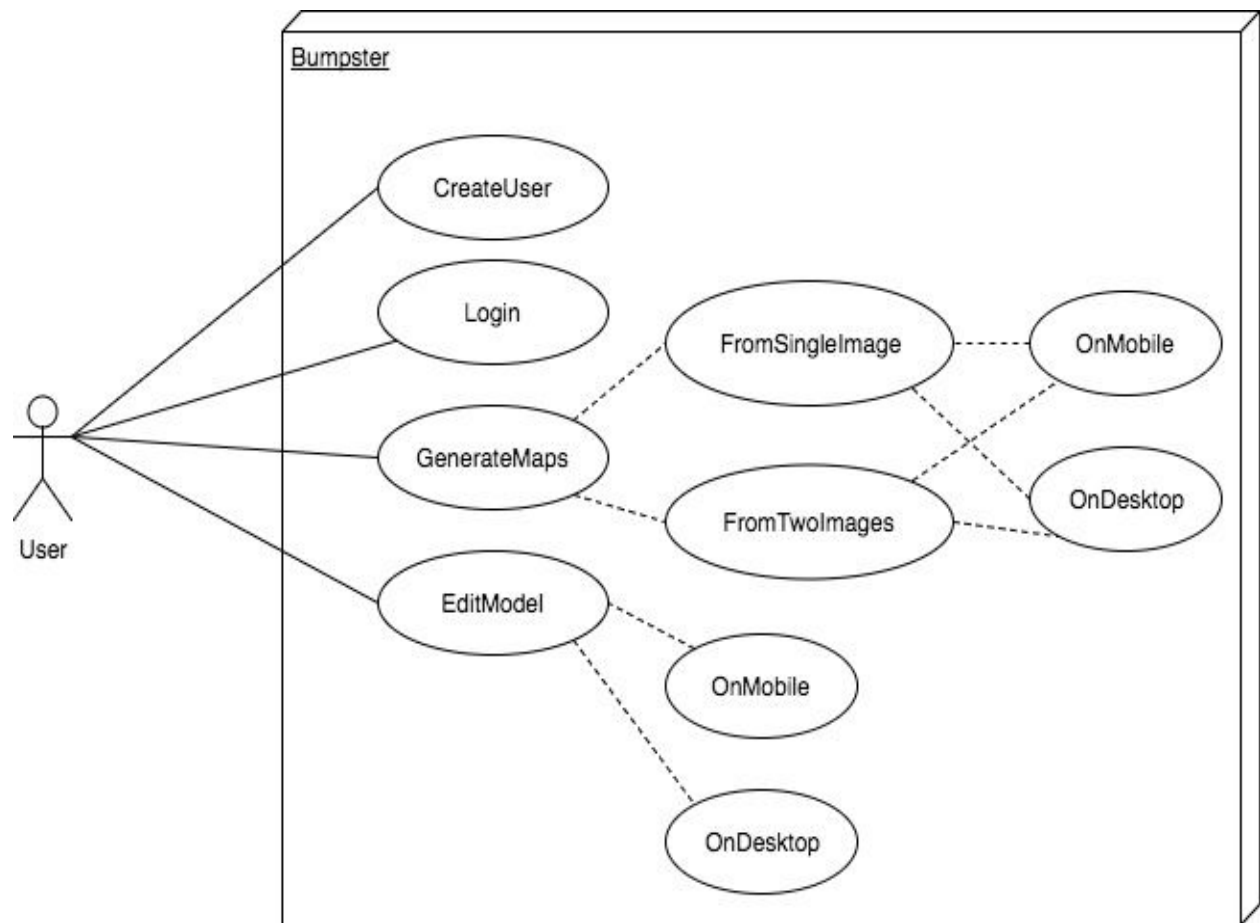
**Exit Conditions:**

- User exits the app.
- User decides to do something else.

**Main Flow of Events:**

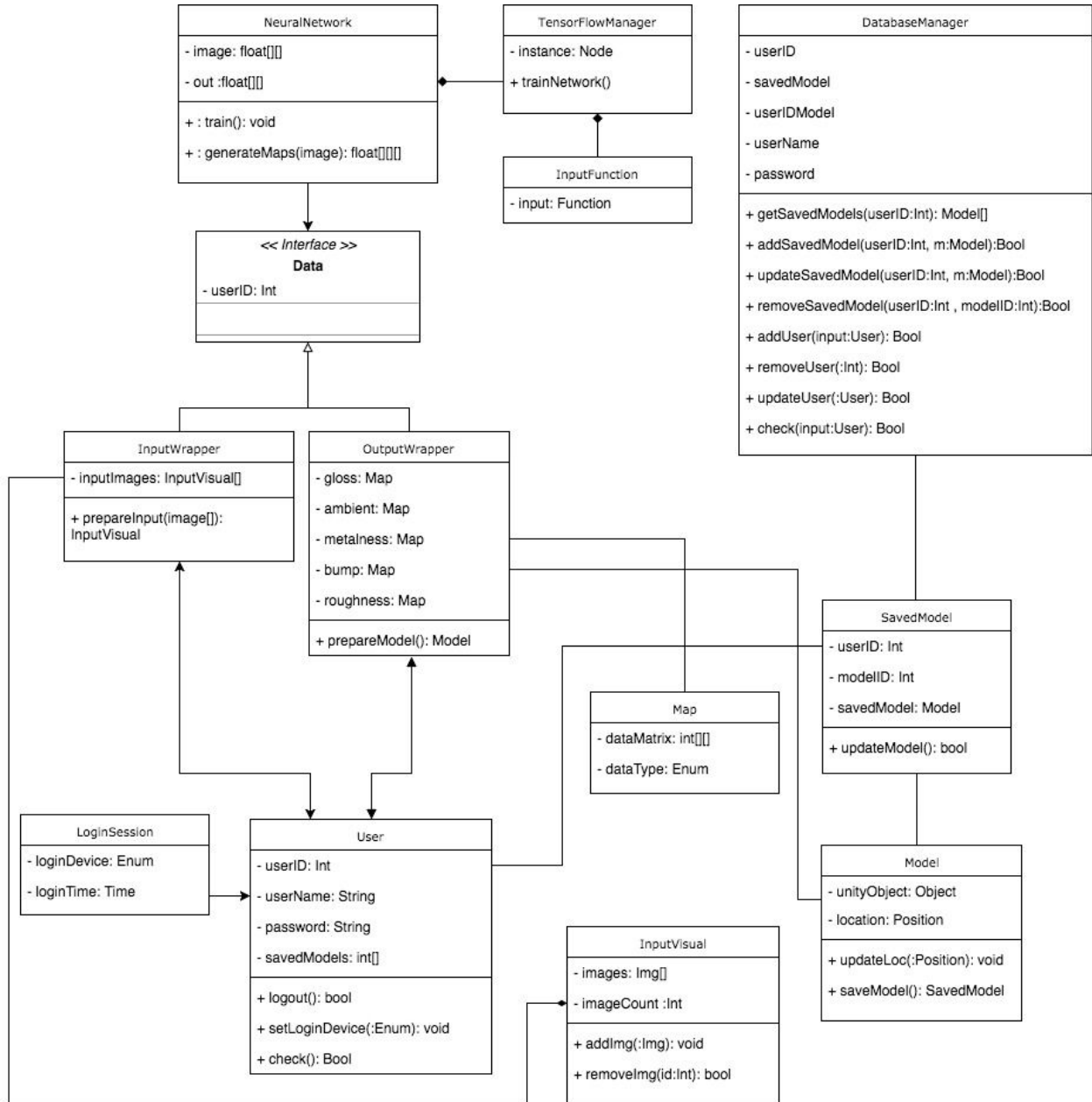
- User opens the app.
- User chooses to create a new model.
- User chooses to take only one photo from the phone camera.
- User takes a nice photo and proceeds to the map generation flow.

### 3.3.2 Use Case Model



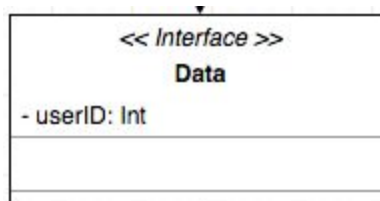


### 3.3.3 Object and Class Model



UML Diagram

### 3.3.3.1 Data



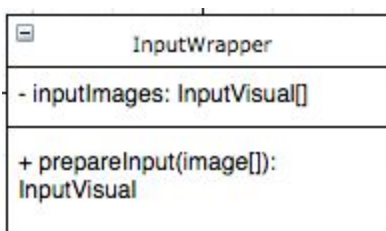
#### Main Features

This class represents the parent class of `InputWrapper` and `OutputWrapper`, which are used to carry information between server and client of Bumpster. We chose to implement this as an interface, because we believe there should be a contract of how data should be created since data in Bumpster is a very fundamental aspect of our project, thus should be made standard using certain rules defined in this interface.

#### Attributes

**userID: Int** This attribute is used to define a part of data which is in consensus over all types of data, being which user it belongs to. It is stored in the form of an integer.

### 3.3.3.2 InputWrapper



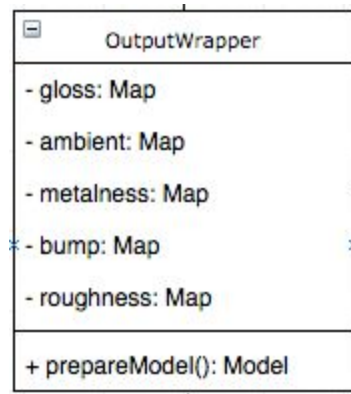
#### Main Features

This class represents the data being carried from user to the server. This class may also carry additional information about the input being created, which will be clarified in the design process of this project. Getter and setter method are also not included in this model as it is trivial. This class uses the interface **Data** to implement some core rules that should be present in all types of data.

#### Attributes

**inputImages: inputVisual[]** This variable represents the the of photos taken by the user to be processed in the server. **inputVisual** is also a wrapper class for the set of images and relevant data required to process the input.

### 3.3.3.3 Output Wrapper



#### Main Features

This class represents the data being carried from server to the user. It provides a cleaner way to transfer various types of information easily, also may be modified to perform various operations on them as well. This class uses the interface **Data** to implement some core rules that should be present in all types of data.

#### Attributes

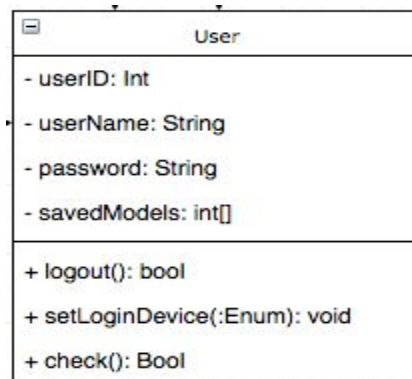
**roughness: Map** This variable carries the information called *roughness* of the generated output. We defined a class called **Map** which carries the 2D matrix of every data point generated, and some other relevant information. Every entry in that matrix will represent the roughness of that data point.

**bump: Map** This variable carries the information called *bump* of the generated output. We defined a class called **Map** which carries the 2D matrix of every data point generated, and some other relevant information. Entries in this map's matrix will represent the unit height of every point when looked bird-eye view.

**metalness: Map** This variable carries the information called *metalness* of the generated output. We defined a class called **Map** which carries the 2D matrix of every data point generated, and some other relevant information. 2D matrix of Metalness will basically contain the amount of reflectivity of every data point.

**ambient: Map** A physically-based rendering term *Ambient occlusion* of the generated output will be stored in this map. We defined a class called **Map** which carries the 2D matrix of every data point generated, and some other relevant information. 2D matrix of ambient occlusion will basically contain the amount of large scale occluded light of every data point.

### 3.3.3.4 User



#### Main Features

User class represents the user object which is required to access the features of Bumpster. Every user is stored in the database. Because a user can use both their mobile and desktop devices to access Bumpster, we made a user model to adapt all the possible cases of use.

#### Attributes

**userID: Int** This variable carries the distinctive feature of every user, which is their identification number in the system.

**userName: String** To be more user friendly, we allow users to set and login using the username they desire. This variable will be used to login to system and show the user-related data.

**password: String** Password which the user will login with.

**savedModels: Int[]** This variable will contain the id numbers of the saved models. Since we will also store these variables in database, only the ID and not the object itself can be used to retrieve them.

#### Operations

**logout(): Bool** This operation will destroy the login session object connected to the user. It will not have any parameters because logout object is the user itself, and output will be a boolean depending on the success of logout operation.

**check(): Bool** This operation will create the login session object connected to the user. It will not have any parameters because login object is the user itself, and output will be a boolean depending on the success of login operation.

**setLoginDevice(:Enum): void** This operation set the login device of the user. Since we have a unified user model for every platform, we need this operation to set user's distinction between them. This is because the environment will provide different operations for, ie, desktop and mobile users.

### 3.3.3.5 Login Session

LoginSession
- loginDevice: Enum
- loginTime: Time

#### Main Features

LoginSession class represents the current login session of the user. It is essential to the project since it is mainly used to make the distinction between different devices supported by Bumpster, which the user might use to login to the outer system.

#### Attributes

**loginDevice: Enum** There are a list of different login devices, pre-enumerated to the outer system. These devices may be, and not limited to, desktop and mobile devices.

**loginTime: Time** This variable is used to store when user logged in to the system. This information may be used for various purposes, like auto-logout.

### 3.3.3.6 Map

Map
- dataMatrix: int[][]
- dataType: Enum

#### Main Features

Map class is used to store the various kinds of information returning to user from server, and wrapped by OutputWrapper class for a convenient and tidy system design.

#### Attributes

**dataMatrix: int[][]** This attribute is a 2D integer matrix, carrying data of every datapoint returned by the system.

**dataType: Enum** Since we return various types of data, we are using pre-enumerated data to decide which data goes where in associated OutputWrapper

### 3.3.3.7 Input Visual

InputVisual
- images: Img[] - imageCount :Int
+ : addImg(:Img): void + : removeImg(id:Int): bool

#### Main Features

InputVisual class is used to construct the data which will be used while communicating with the server containing neural network. It is essential to note that this object model will expand during the design phase of the project. It is designed to support sending more than one images to neural network and improve accuracy if possible.

#### Attributes

**images: Img[]** This attribute stores single/multiple images to be sent to the server.

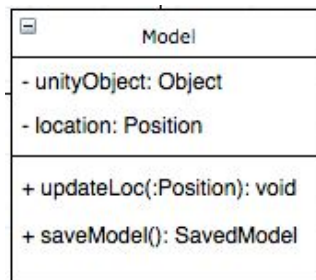
**imageCount: Int** This variable stores the amount of images which will be sent to the server.

#### Operations

**addImg(Img): void** This operation will add a newly taken or pre-selected image from user's storage. It takes an image as a parameter (Multiple images can be batch-inserted, but this will be done by repeatedly calling this operation). It will not generate any data output.

**removeImg(Img): bool** This operation will delete an image from images attribute mentioned. As parameter, it will take the index of the image in the images attribute. It will return the result of the remove operation the form of a boolean.

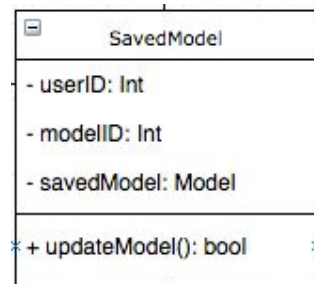
### 3.3.3.8 Model



#### Main Features

Model class is the model which will be generated by using the data coming from the server into a human viewable and visually manipulatable form. Since we are planning to use Unity Development Platform, this object will use the data structures defined in unity. Since there are many Unity-related attributes and operation related with this model, it will not be defined in this report.

### 3.3.3.9 Saved Model



#### Main Features

Model class is the model which will be generated by using the data coming from the server into a human viewable and visually manipulatable form. Since we are planning to use Unity Development Platform, this object will use the data structures defined in unity. Since there are many Unity-related attributes and operation related with this model, it will not be defined in this report.

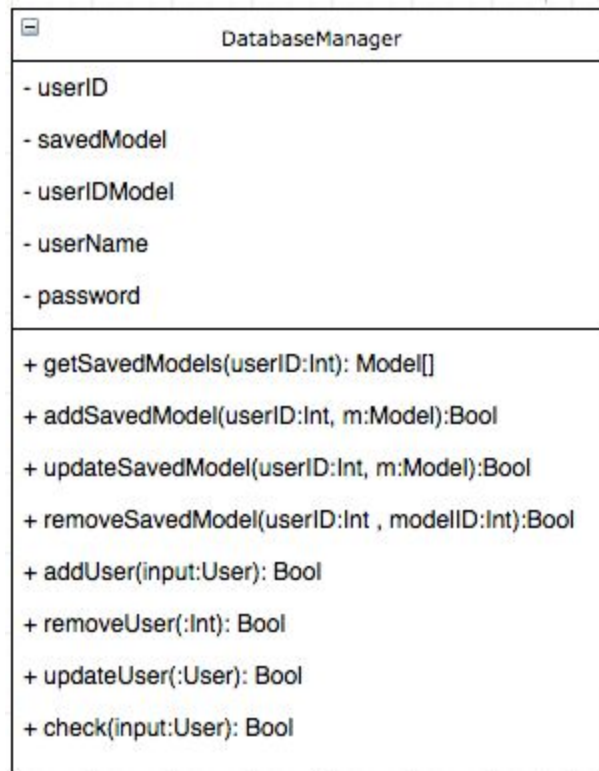
#### Attributes

**userID: Int** This attribute stores the associated user of the model to be saved. It will then be retrieved from the database accordingly.

**modelID: Int** This attribute stores the associated ID of the model to be saved. It will then be retrieved from the database accordingly.

**savedModel: Model** This attribute stores the data files required to re-construct the model in the exact way it was during the user was saving the model.

### 3.3.3.10 Database Manager



#### Main Features

DatabaseManager class will contain the database data structures which will be used to store various objects that need to be accessed retrieved during Bumpster is used.

#### Attributes

**userID: Int** This attribute stores the userID associated user.

**userName: String** This attribute stores the userName associated with each user.

**password: String** This attribute stores the password associated with each user.

**savedModel: Model** This attribute stores the data files required to re-construct the model in the exact way it was during the user was saving the model.

**userIDModel: Int** This attribute stores the user ID of the saved model.



## Operations

**getSavedModels(userID:Int): Model[]** This operation return all the saved models of the user to be view in the user interface. It will take user's id as its parameter and return the model array, containing all the saved model(s).

**addSavedModel(userID:Int , m:Model): bool** This operation will add a saved model for the desired user. It will take the ID of user which the saved model will be added, and the model data which will be added. It will return a boolean determined by the success of the operation.

**updateSavedModel(userID:Int , m:Model): bool** This operation will update a saved model for the desired user. It will take the ID of user which the updated model will be added, and the model data which will be added. It will return a boolean determined by the success of the operation.

**removeSavedModel(modelID:Int): bool** This operation will remove a saved model from user's stored saved models. It will take the ID of the model to be removed. It will return a boolean determined by the success of the operation.

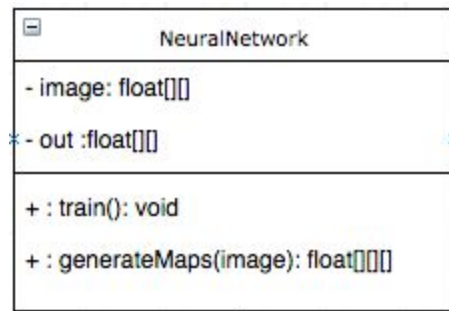
**addUser(input:User): bool** This operation will add a user to the database. It will take the user object as its parameter. It will return a boolean determined by the success of the operation.

**removeUser(:Int): bool** This operation will add a remove a user from the database. It will take the user's ID as its parameter. It will return a boolean determined by the success of the operation.

**check(:User): bool** This operation will check if the user with specified username and password exists. It will return a boolean determined by the success of the operation and login will be performed.

**updateUser(:User): bool** This operation will add modify the desired attributes of a user stored in the database. It will take the newly generated user object as its parameter. The ID of the user to be modified will be the same the the ID stated in the parameter. It will return a boolean determined by the success of the operation.

### 3.3.3.11 Neural Network



#### Main Features

`NeuralNetwork` will be the class that connects the deep neural intelligence portion of Bumpster to the database and client portions. It includes "image" and "out" attributes and "train()" and "generateMaps(image)" functions.

#### Attributes

**image: float[][]** This attribute stores input image.

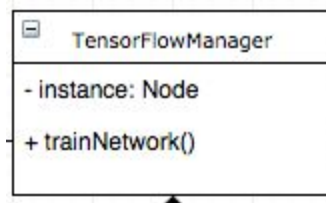
**out: float[][]** This attribute stores the current map that is being generated.

#### Operations

**train(): void:** This operation is called once, to train the neurons and the whole neural network in order to ensure that it detects regions of metals and extracts the five map types. The training operation is done via the pre-rendered images and their corresponding maps.

**generateMaps(image): float[][][]:** This operation does the core work of the project. The maps of the input image are generated via this function. The output is an array of 2D float array, which includes all the map types.

### 3.3.3.12 TensorFlow Manager



#### Main Features

TensorFlow Manager will be the class that wraps the TensorFlow library functions that will be utilised.

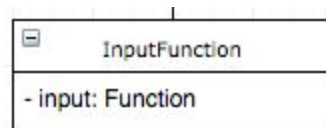
#### Attributes

**instance: Node** This attribute is one neuron instance.

#### Operations

**trainNetwork(): void** This function trains the network.

### 3.3.3.13 Input Function



#### Main Features

InputFunction will be the class that stores the input function that enables the network to be trained.

#### Attributes

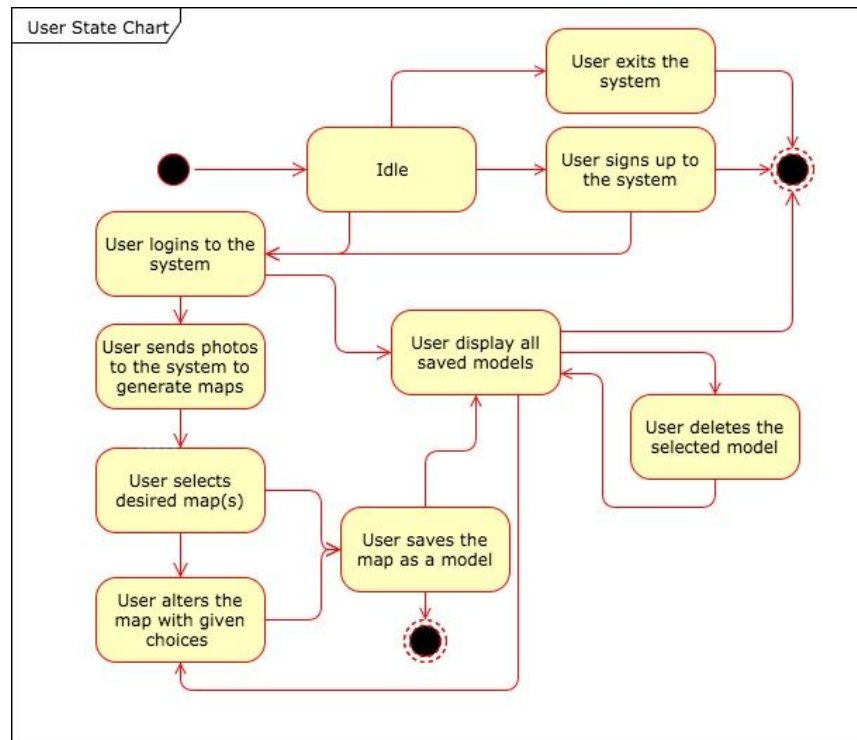
**input: Function** This attribute stores the input function.

### 3.3.4 Dynamic Models

This section includes the state diagrams, sequence diagrams and activity diagrams of Bumpster.

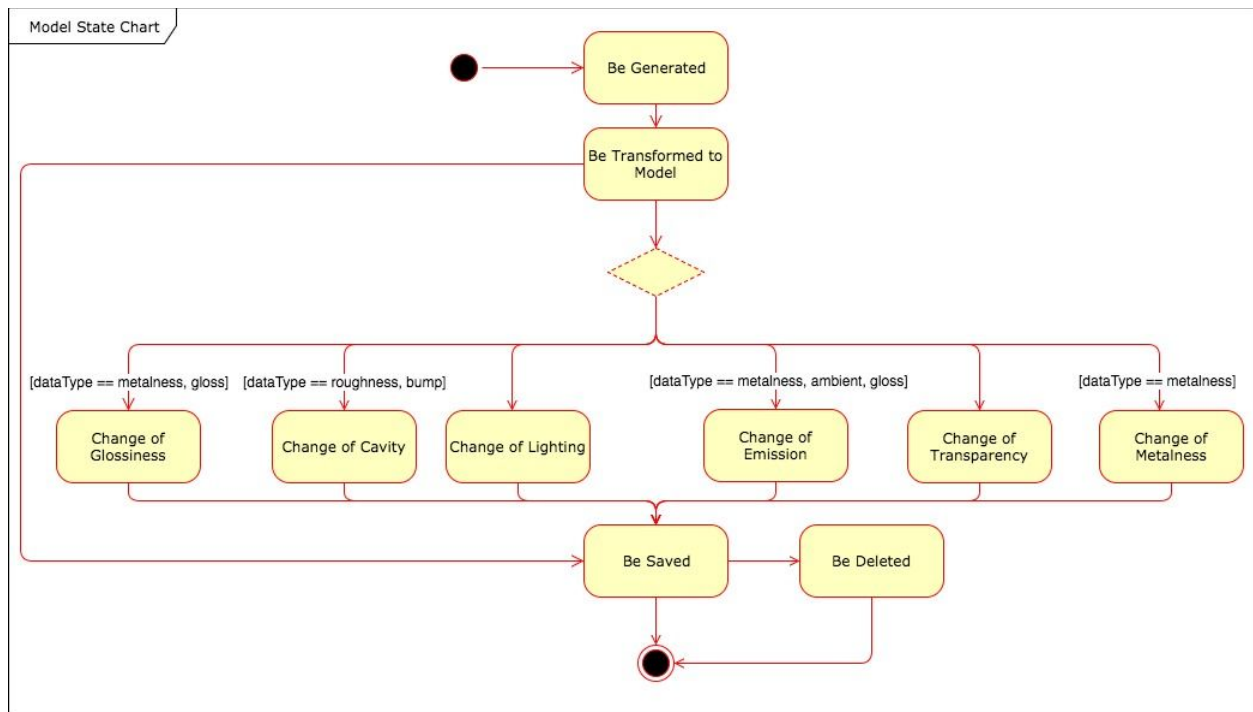
#### 3.3.4.1 State Diagrams

##### 3.3.4.1.1 User State Chart



The user state chart generally describes the capabilities of the user. The user signs up to the system if he/she does not have an account in the system before. It is necessary for user to have an account for saving the information belong to the user such as the generated models he/she has. The user logs in to the system if he/she has an account. After uploading photograph(s) to the system, the system generates 5 different maps (bump, ambient, gloss, metalness and roughness). Desired maps are chosen by the user to save them as a model or edit them. Presented features such as glossiness, emission etc. can be used for editing the model. Then, the user can again save the updated version of the model. In addition, the user can display all the saved models in his/her account in order to make changes on his/her models or delete the one he/she wants. At any point, the user can exit the system.

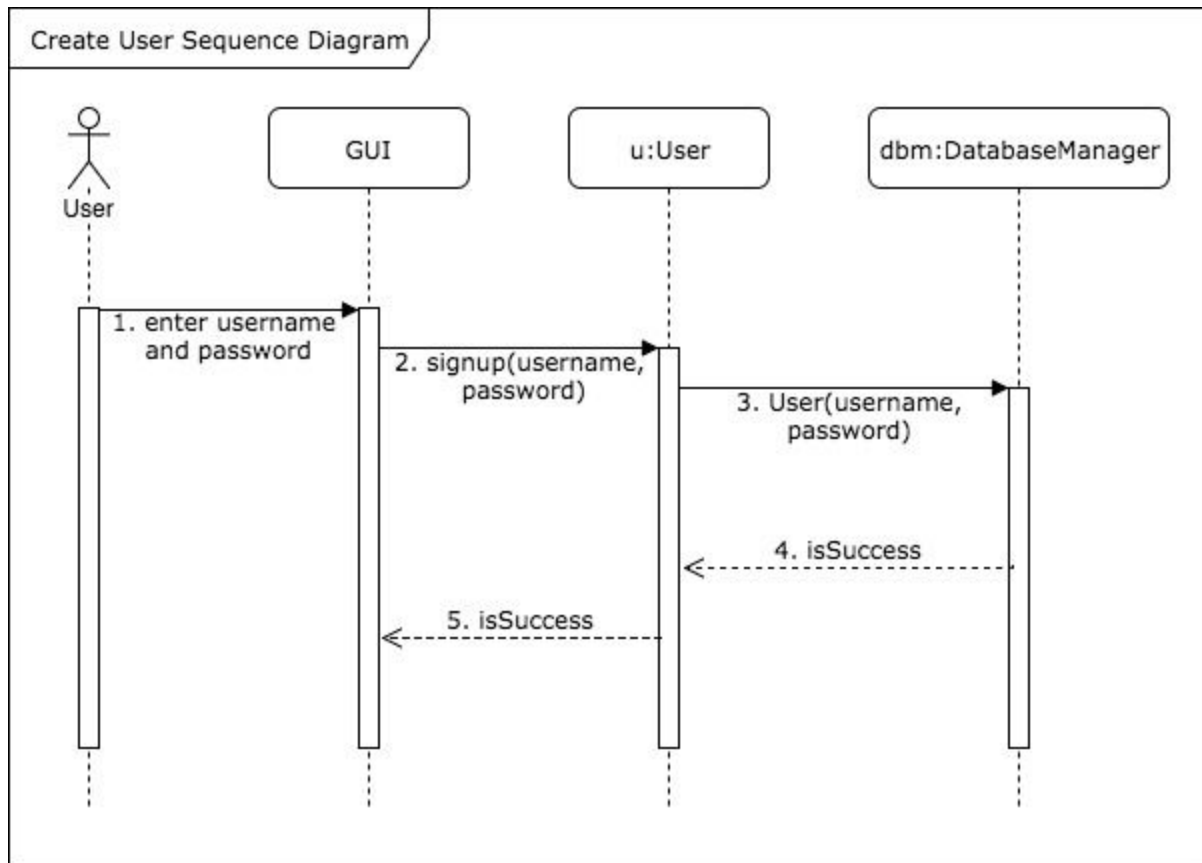
### 3.3.4.1.2 Model State Chart



The model state chart shows the model description from the point of view of a map. First, a map is needed to be generated independent from its type. 5 different maps are generated for the user by using the uploaded photo(s) of the user. Then, the map is transformed to model to be displayed to the user. The user can make several changes on the model based on the type of the map. For metalness map, editing the model in terms of glossiness, lighting, emission, transparency and metalness is possible. For ambient map, editing is similar to metalness map except change of metalness. The user can edit the model in terms of glossiness, lighting, emission and transparency for roughness map. For bump and roughness map, change of cavity, lighting and transparency can be done on the model. The user can save the updated version of model or original version after the map is transformed. The saved models can also be deleted by the user.

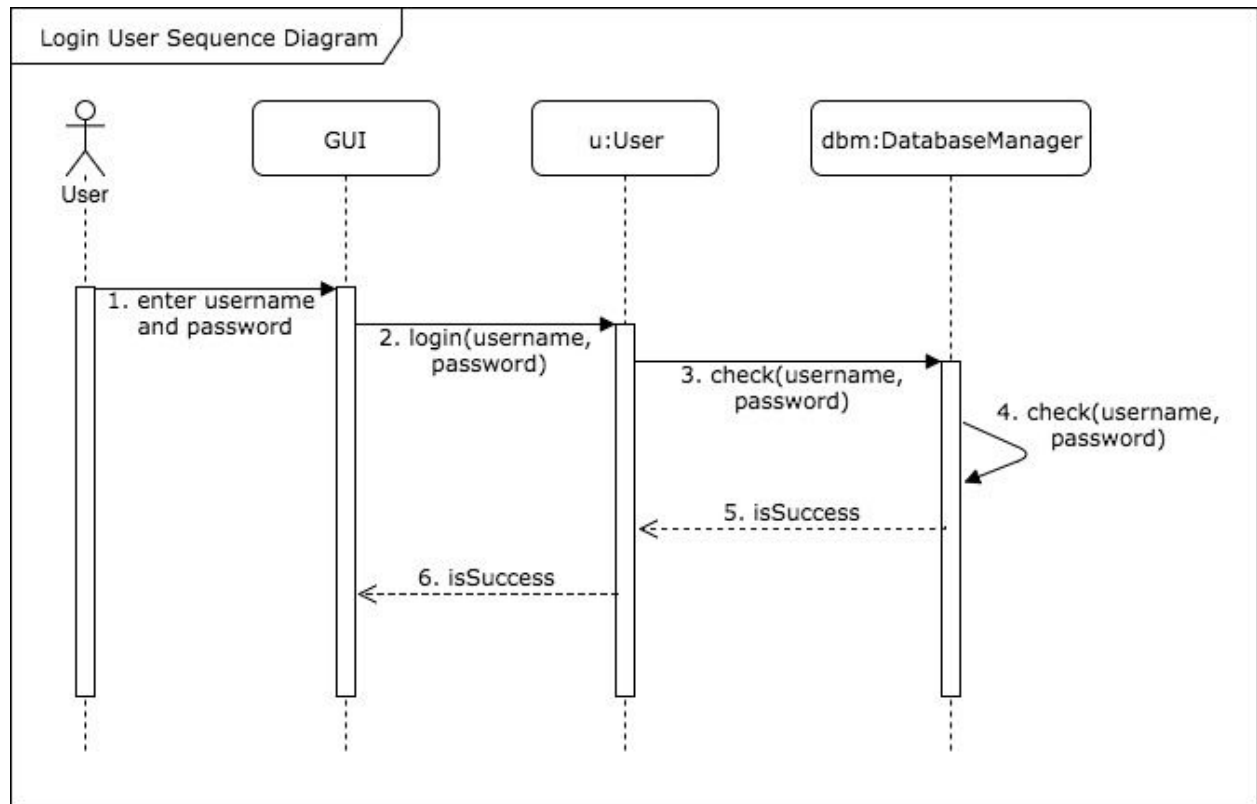
### 3.3.4.2 Sequence Diagrams

#### 3.3.4.2.1 Scenario 1 Sequence Diagram



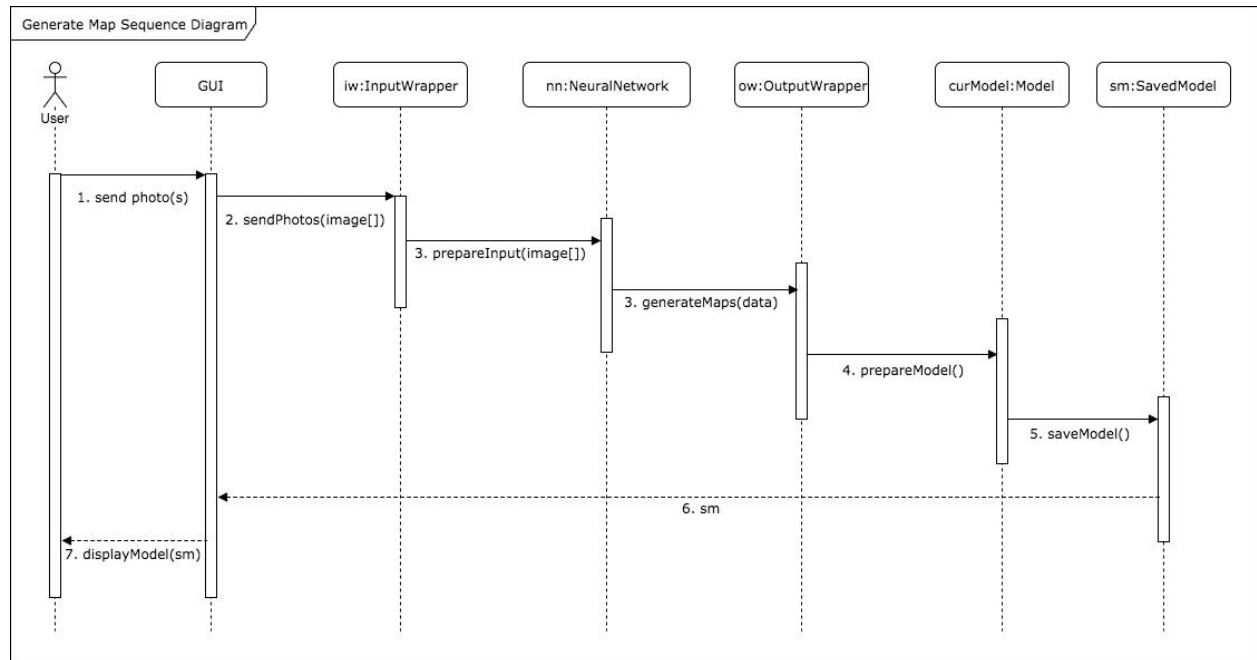
The sequence diagram describes the scenario which a user wants to create an account. The user fills the required blanks (username and password) for signing up to the system. The information given by the user is passed to the DatabaseManager. The DatabaseManager creates the user by saving the received account information. The success response is transmitted back to the User and then GUI to inform the user about success of account creation.

### 3.3.4.2.2 Scenario 2 Sequence Diagram



The sequence diagram shows the scenario of logging to the system. Similar to scenario 1, the user enters the username and password in order to sign in to the system. Then, the received information is transmitted to DatabaseManager for checking the correctness. The DatabaseManager checks if the user exists or if the password is correct for given username. The success response is transmitted back to the User and then GUI to inform the user.

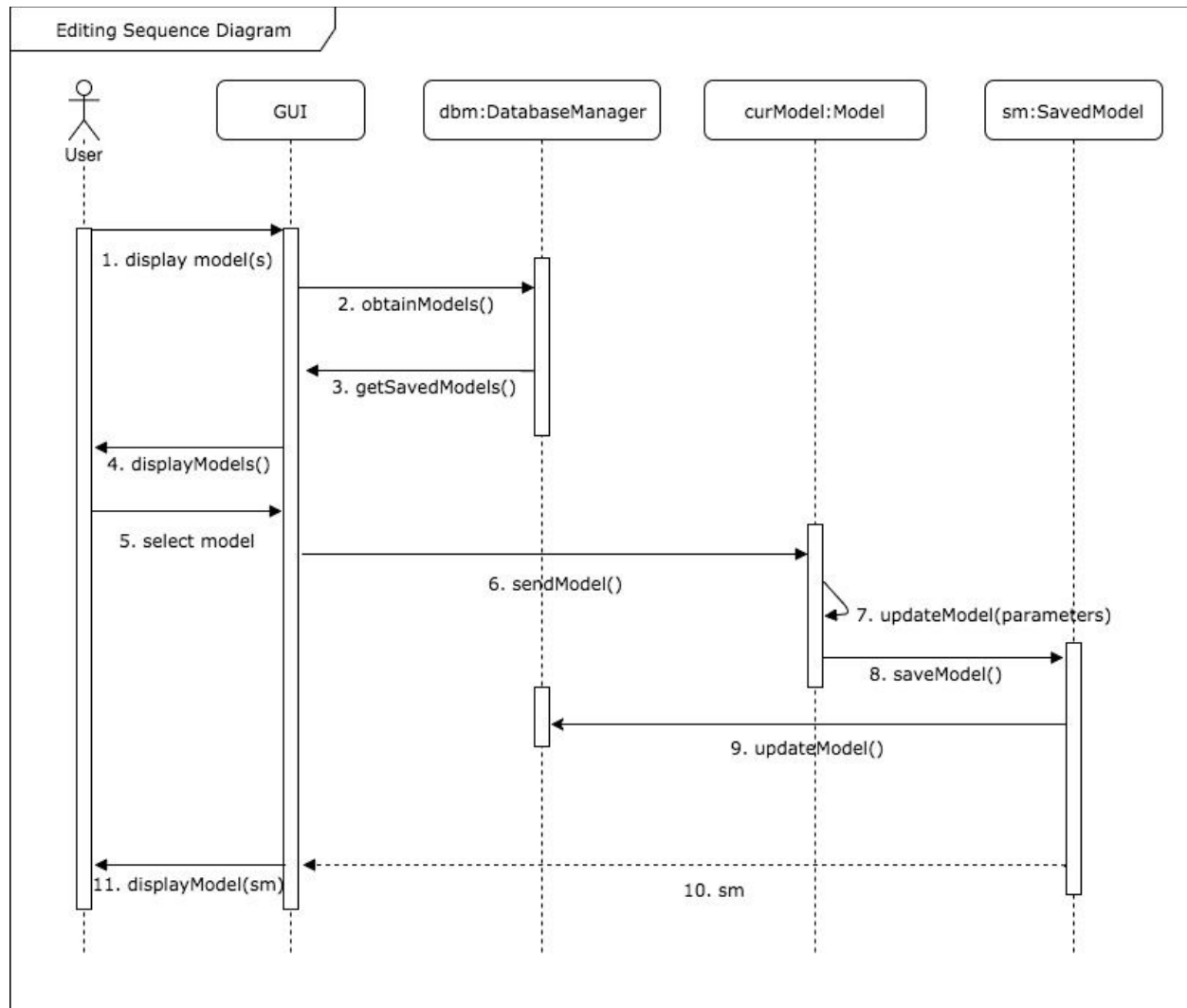
### 3.3.4.2.3 Scenario 3 Sequence Diagram



The sequence diagram describes the generation of map in the system. The user uploads photo(s) to the system. Then, the received photos are transferred to the InputWrapper by GUI. The InputWrapper transmits the image arrays to the Neural Network. The Neural Network generate 5 different maps from the given arrays and transfer them to the OutputWrapper. The OutputWrapper convert maps into models and transmit them to the Model. The models are saved to the system in SavedModel. The saved models are transmitted back to GUI to display them to the user.

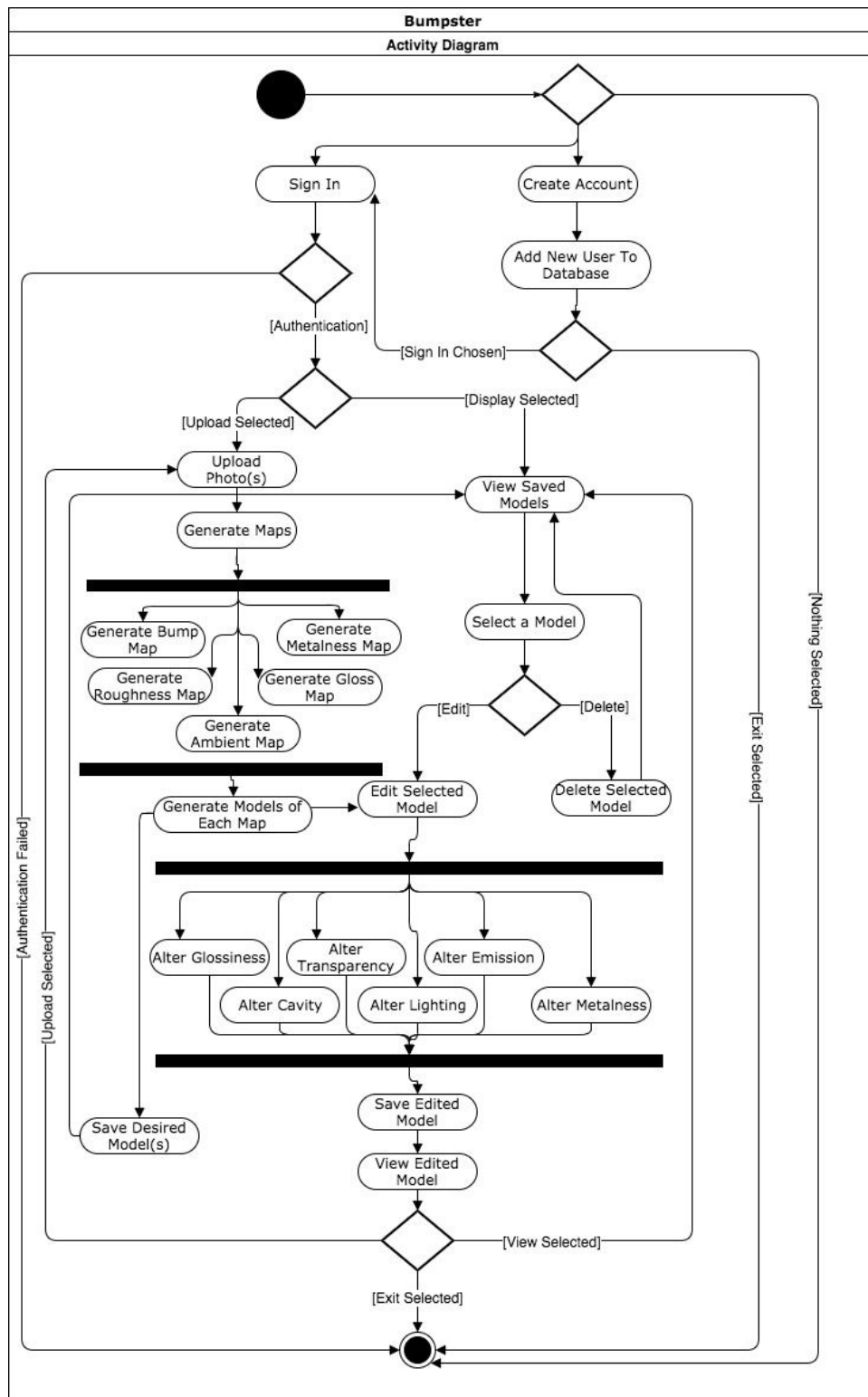


### 3.3.4.2.4 Scenario 4 Sequence Diagram



The sequence diagram shows the editing of models in the system. The user clicks display model button in order to look through his/her saved models. The GUI transmits the request to the DatabaseManager. The DatabaseManager transmits back the saved models to the GUI. The GUI displays the received models. The User selects a model among saved models. The selected model is transmitted to the Model. The received model is updated with given parameters in the Model and saved in the SavedModel. The saved version of the model is transmitted to the DatabaseManager by the SavedModel. The DatabaseManager updates the model in the database. The SavedModel also transmits back the new version of the model to the GUI to display the model to User.

### 3.3.4.3 Activity Diagram



The activity diagram shows all of the steps that could be taken by the system and the user throughout all of the parts of Bumpster. There are some parallel lanes that show that more than one activity can be performed at the same time.

First of all, the application can be entered by first signing in or creating an account. If a user chooses to create an account, the system adds a new user to the database. After the user is successfully added to the database, the system can be exited or a sign in operation can be performed. After sign in operation is requested, if authentication fails, the system is exited. If authentication is performed, there are two choices: upload photos or view previously saved model of the current signed in user. If upload is selected, after the system receives the photo(s), the bump, metalness, gloss, roughness and ambient maps are generated using the neural network. Then, the models of each map are generated and the desired models can be saved. Also, the selected models can be edited. If display is selected, the saved models are displayed and when a model is selected, the user can select to edit it or delete it. If the model is deleted, the system returns to the saved models view.

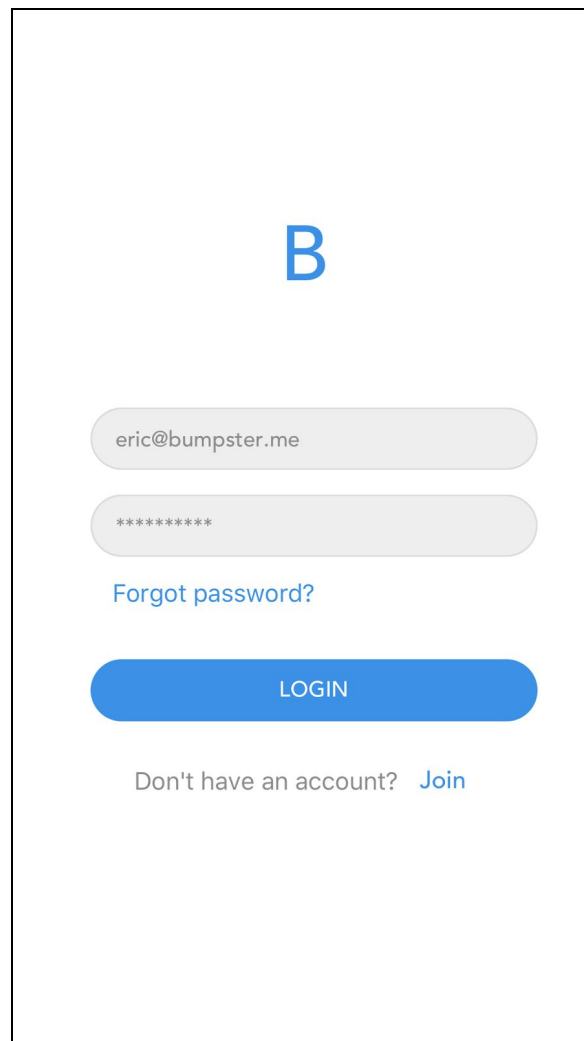
If a model is selected to be edited, in a parallel manner, the glossiness, transparency, cavity, lighting, emission and metalness can be changed. After they are changed, the edited model can be saved. The saved model can be viewed and then if upload is selected, the system will return to upload photo(s) activity. The user can also choose to view selected models. Then, the user can exit the system.

### 3.3.5 User Interface

This section includes the user interface mock-ups for the mobile and desktop application. Both the mobile and desktop application will be compatible with Android and iOS (for mobile application) and Windows and MacOSx (for desktop application).

#### 3.3.5.1 Mobile Application

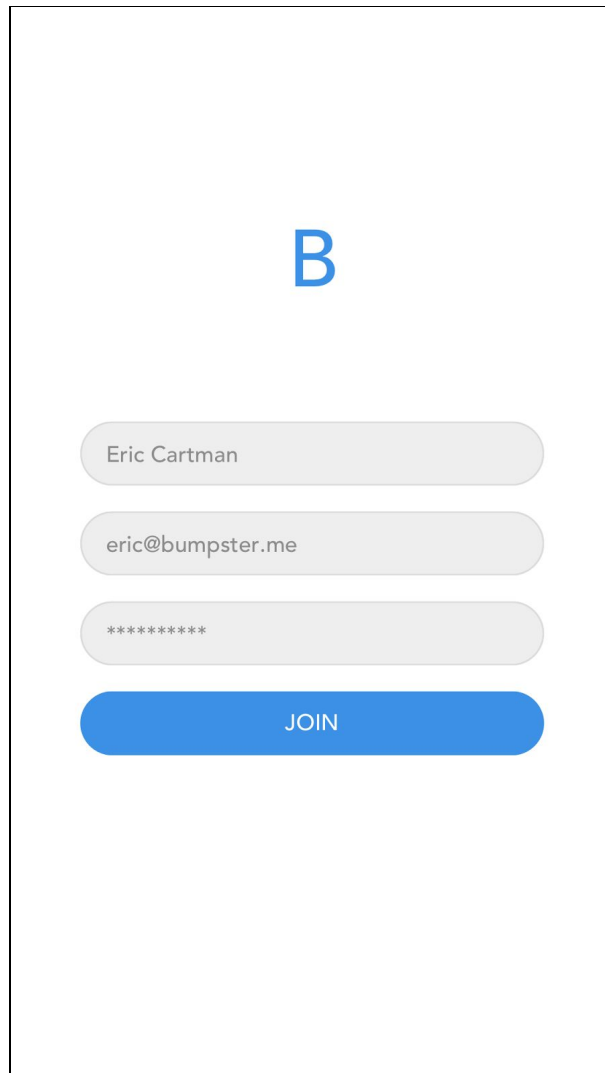
##### 3.3.5.1.1 Sign In Page



The mock-up shows a sign-in page with a large blue letter 'B' at the top center. Below it are two rounded rectangular input fields: the first contains the email 'eric@bumpster.me' and the second contains a masked password '\*\*\*\*\*'. A blue link 'Forgot password?' is positioned below the password field. A prominent blue rounded button labeled 'LOGIN' is centered below the link. At the bottom, the text 'Don't have an account?' is followed by a blue 'Join' link.

Users can sign in to app using their previously registered email addresses and passwords. If they don't have an account, they can press Join button located at the bottom of the form and they will be redirected to Sign Up page.

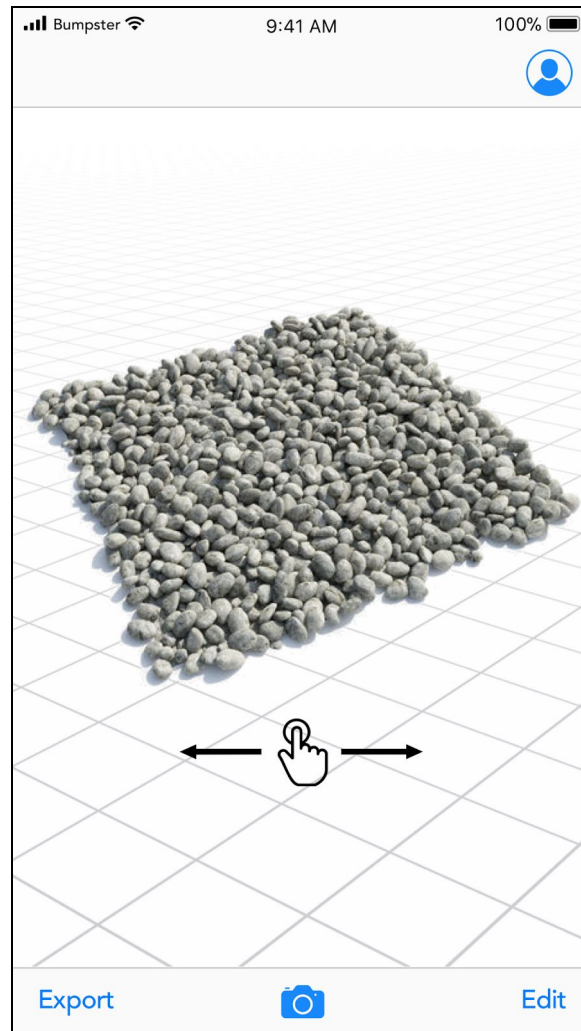
### 3.3.5.1.2 Sign Up Page



The image shows a sign-up page layout. At the top center is a large blue letter 'B'. Below it are four rounded rectangular input fields. The first field contains the text 'Eric Cartman'. The second field contains the email 'eric@bumpster.me'. The third field contains a series of asterisks '\*\*\*\*\*'. Below these fields is a solid blue button with the word 'JOIN' in white capital letters.

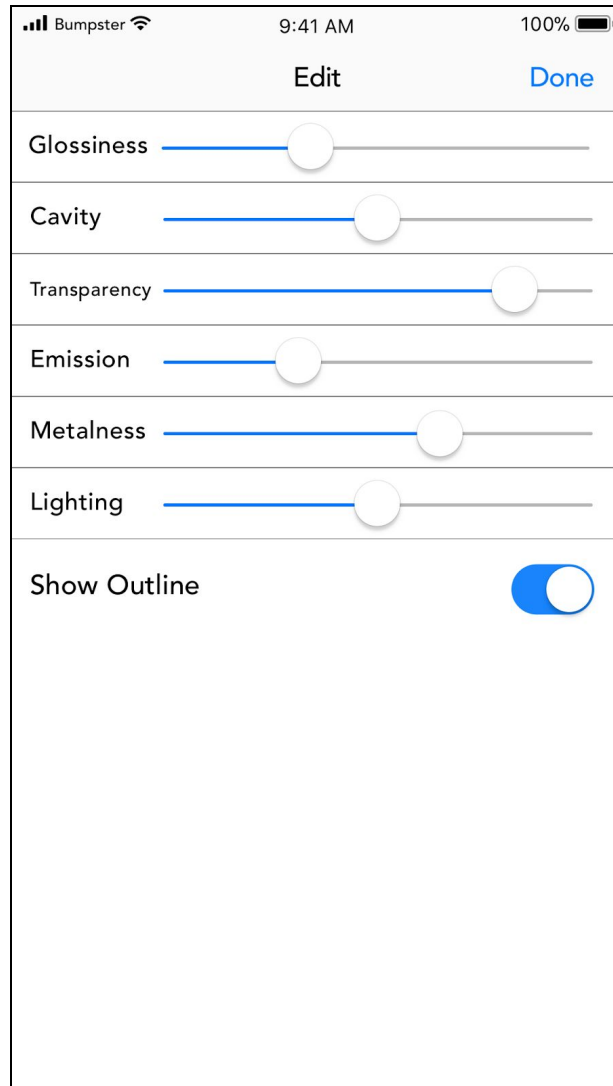
Users can signup by entering their name, surname, a valid email and a password, and pressing the Join button located at the bottom of the form.

### 3.3.5.1.3 Main Page



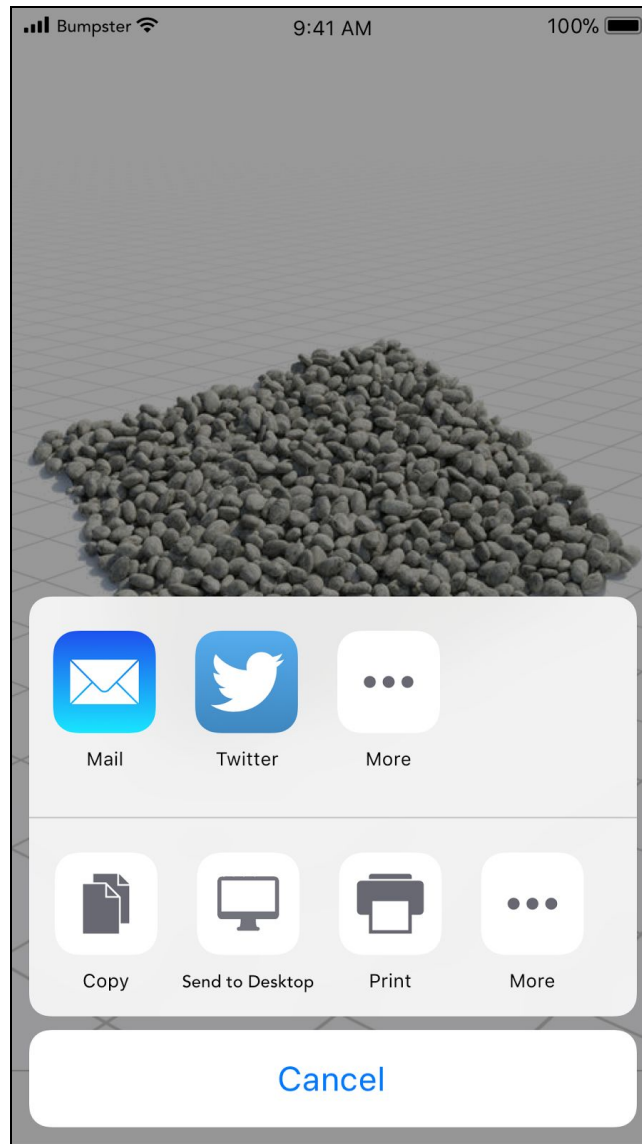
In main page, users can view the extracted map in 3D by touching and dragging it horizontally and vertically. In the top navbar users can press profile icon and be redirected their profile page. In the bottom toolbar users can access Export, Camera and Edit buttons. Users can press Camera icon to take new photos. Users can press Edit button and be redirected to edit view. Users can also press Export button to export extracted map.

#### 3.3.5.1.4 Edit Page



Users can edit extracted maps' various properties including but not limited to glossiness, cavity, transparency and metalness. Users can select desired level for each property by changing the slider value.

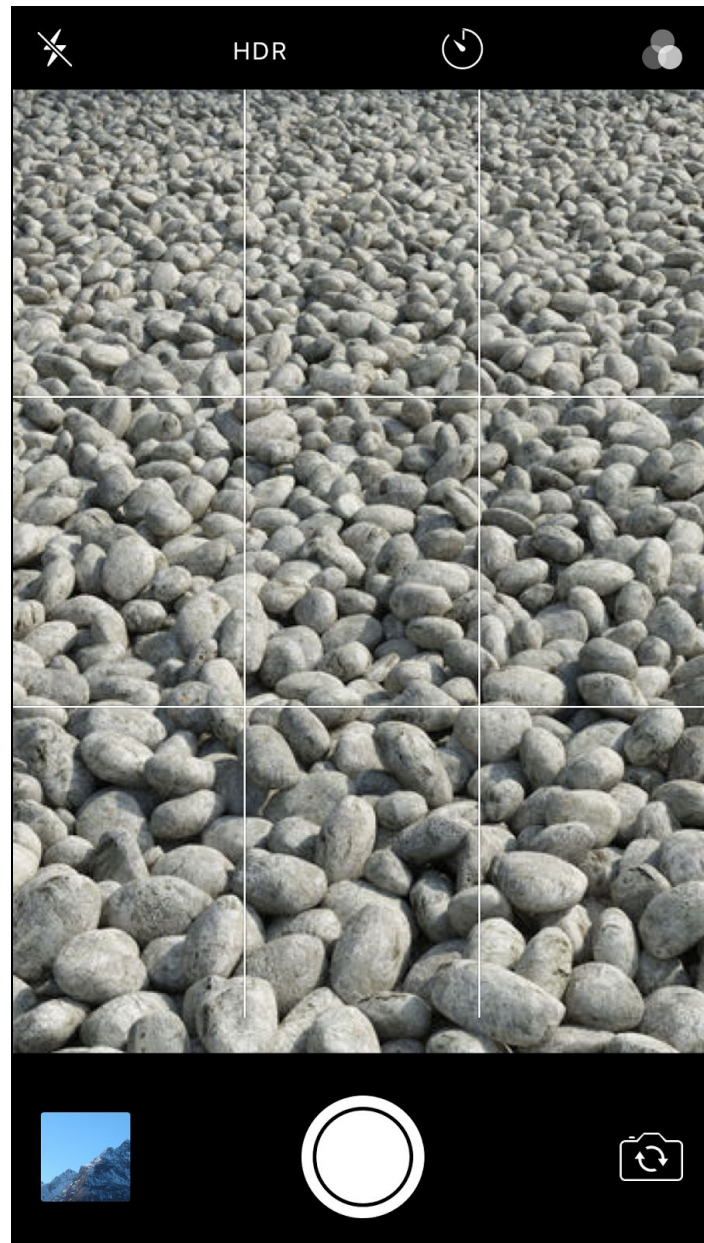
### 3.3.5.1.5 Export Page



Users can export extracted map in export menu. Users can send the map to our desktop application, send it via mail, or share the result to social media. Users can press Cancel button to dismiss this menu.

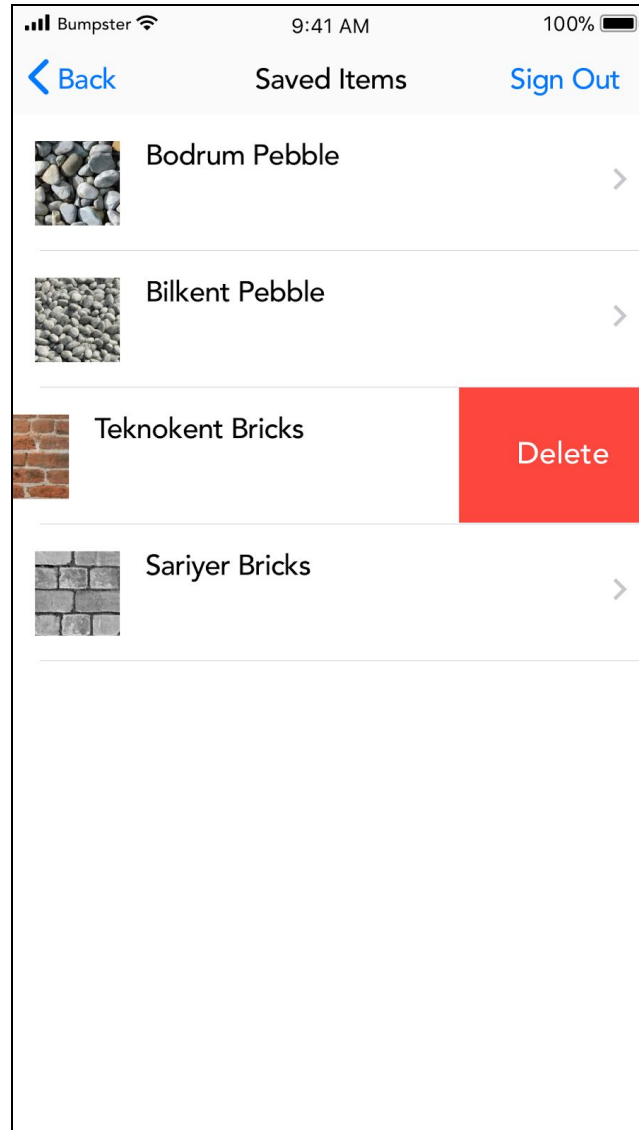


### 3.3.5.1.6 Camera Page



Users can take a new photo using Camera page. In this page users can also select a photo from their photo library.

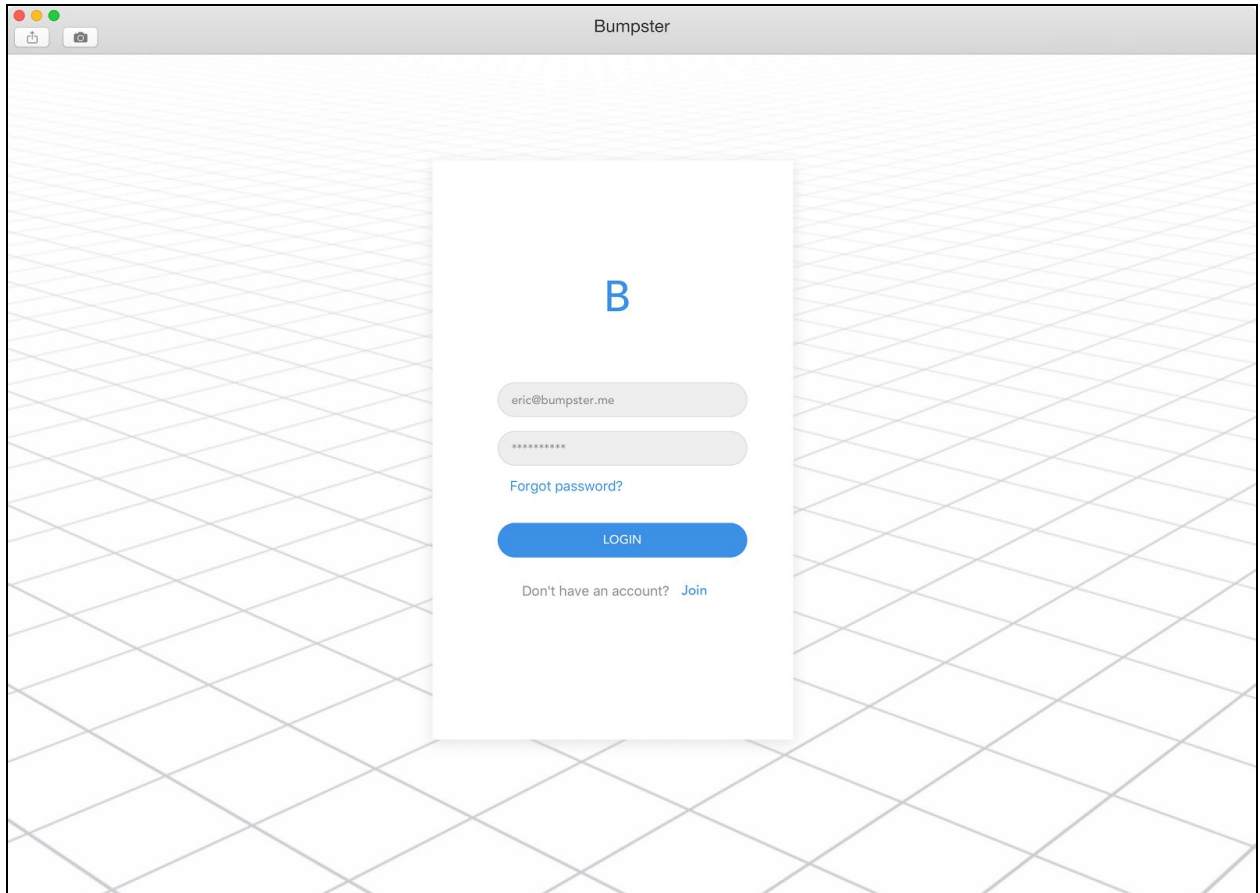
### 3.3.5.1.7 Profile Page



In Profile page, users can view their saved maps, and sign out using the right button located in top navigation bar. Users can go back to Main page by pressing left button located in top navigation bar. In saved maps view, users can see a preview image and a title for each map. Users can also delete their maps by swiping left and pressing the red Delete button. Users can also select a map from this list and they will be redirected to the main page with selected map displayed.

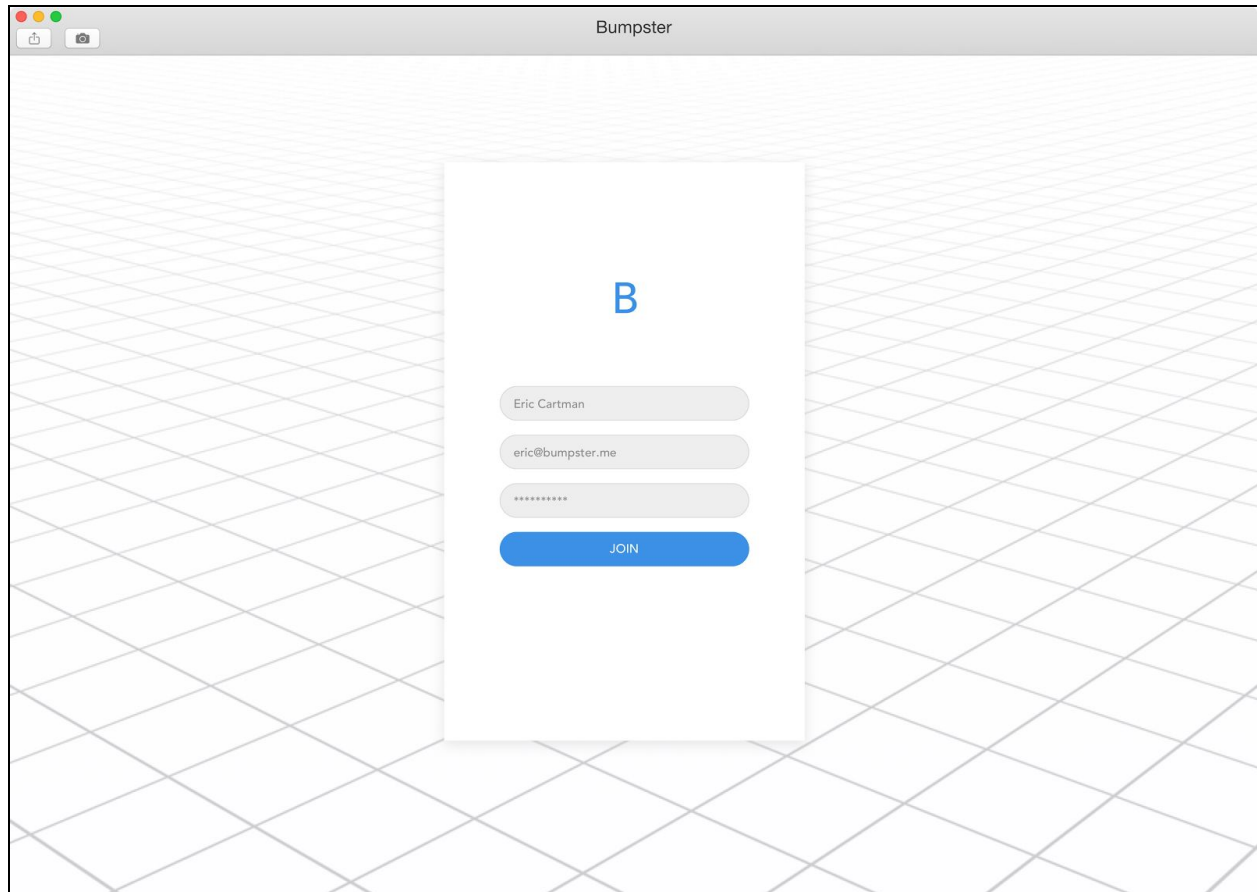
### 3.3.5.2 Desktop Application

#### 3.3.5.2.1 Sign In Page



Users can sign in to app using their previously registered email addresses and passwords. If they don't have an account, they can press Join button located at the bottom of the form and they will be redirected to Sign Up page.

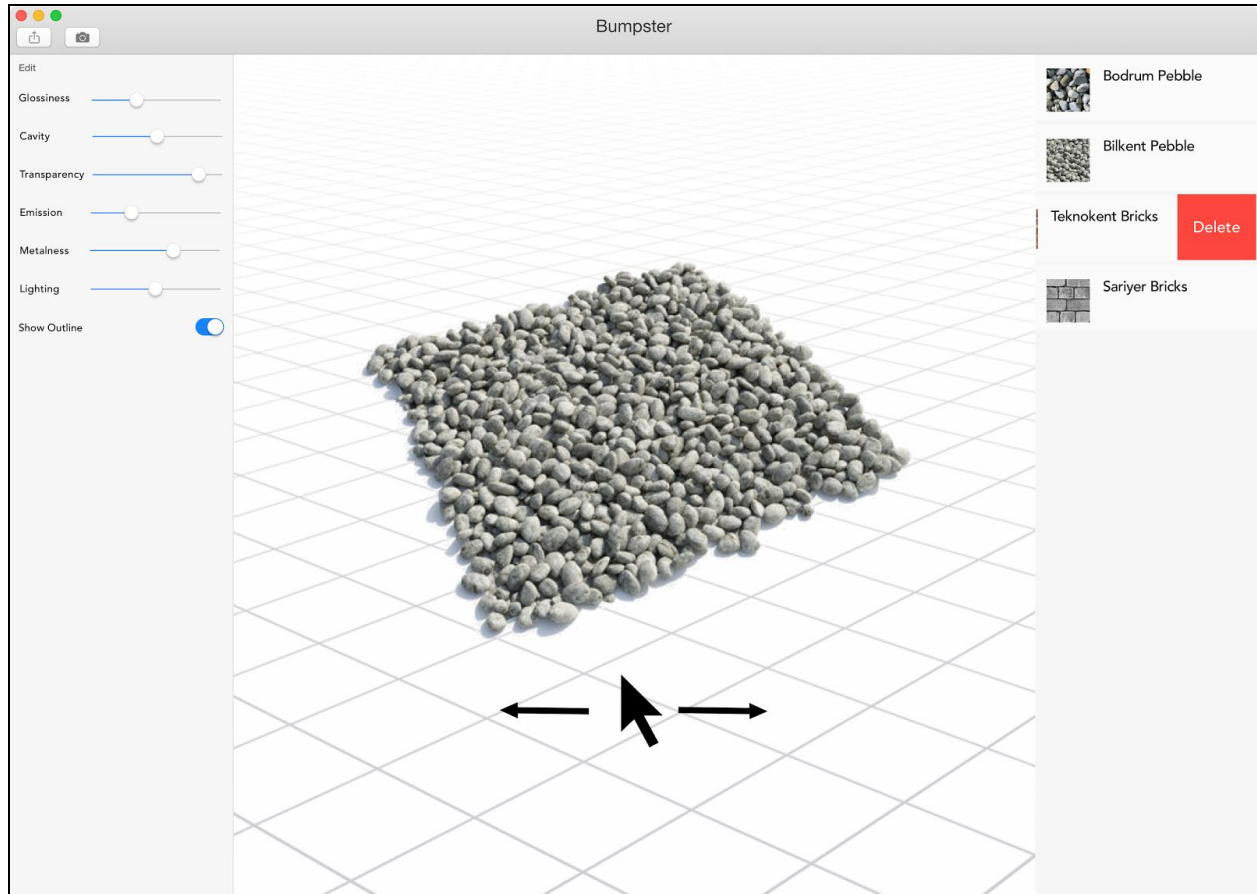
### 3.3.5.2.2 Sign Up Page



The screenshot shows a web browser window titled "Bumpster". The background of the page is a light gray with a subtle, repeating diamond-shaped pattern. In the center, there is a white rectangular sign-up form. At the top of the form is a large blue letter "B". Below it are three input fields: the first contains the text "Eric Cartman", the second contains "eric@bumpster.me", and the third contains a series of dots representing a password. At the bottom of the form is a blue button with the word "JOIN" in white capital letters.

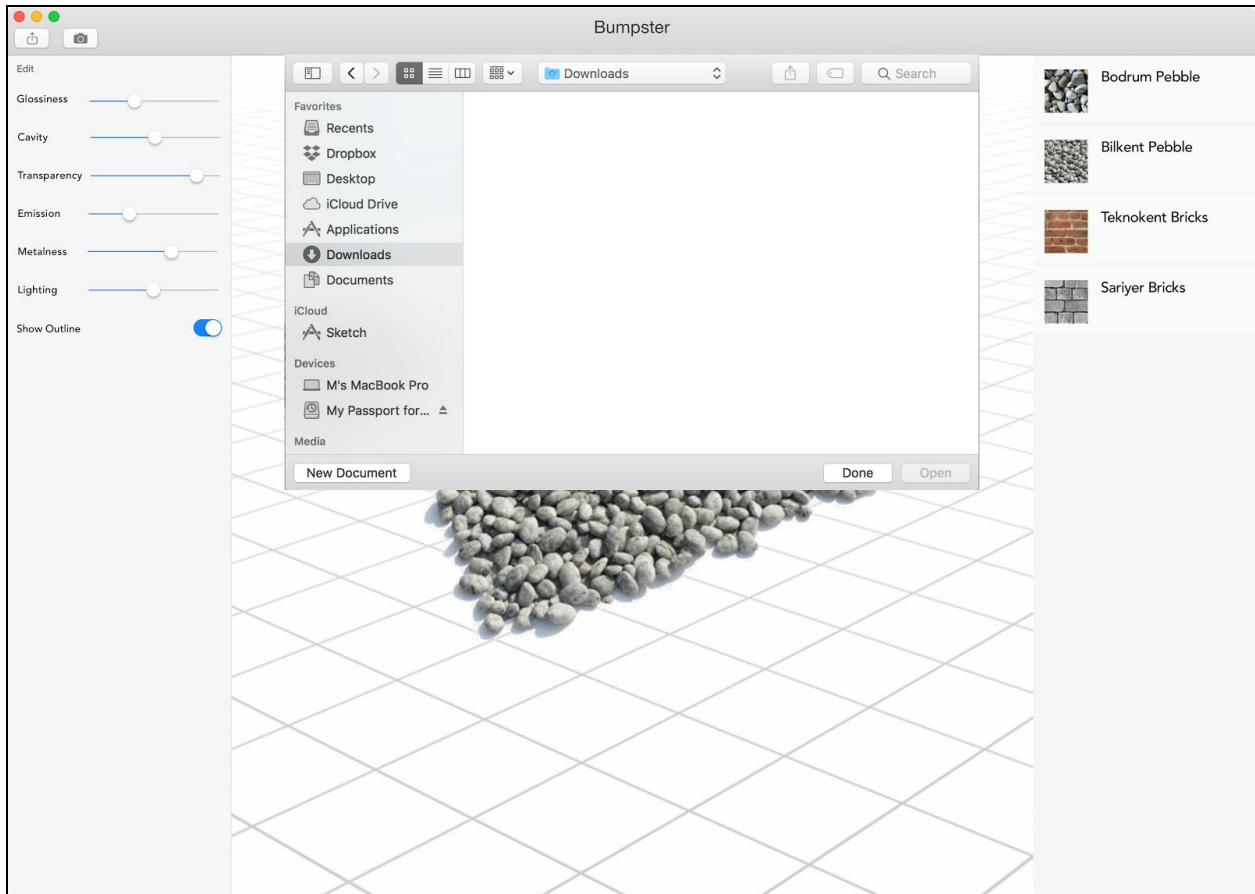
Users can signup by entering their name, surname, a valid email and a password, and pressing the Join button located at the bottom of the form.

### 3.3.5.2.3 Main Page



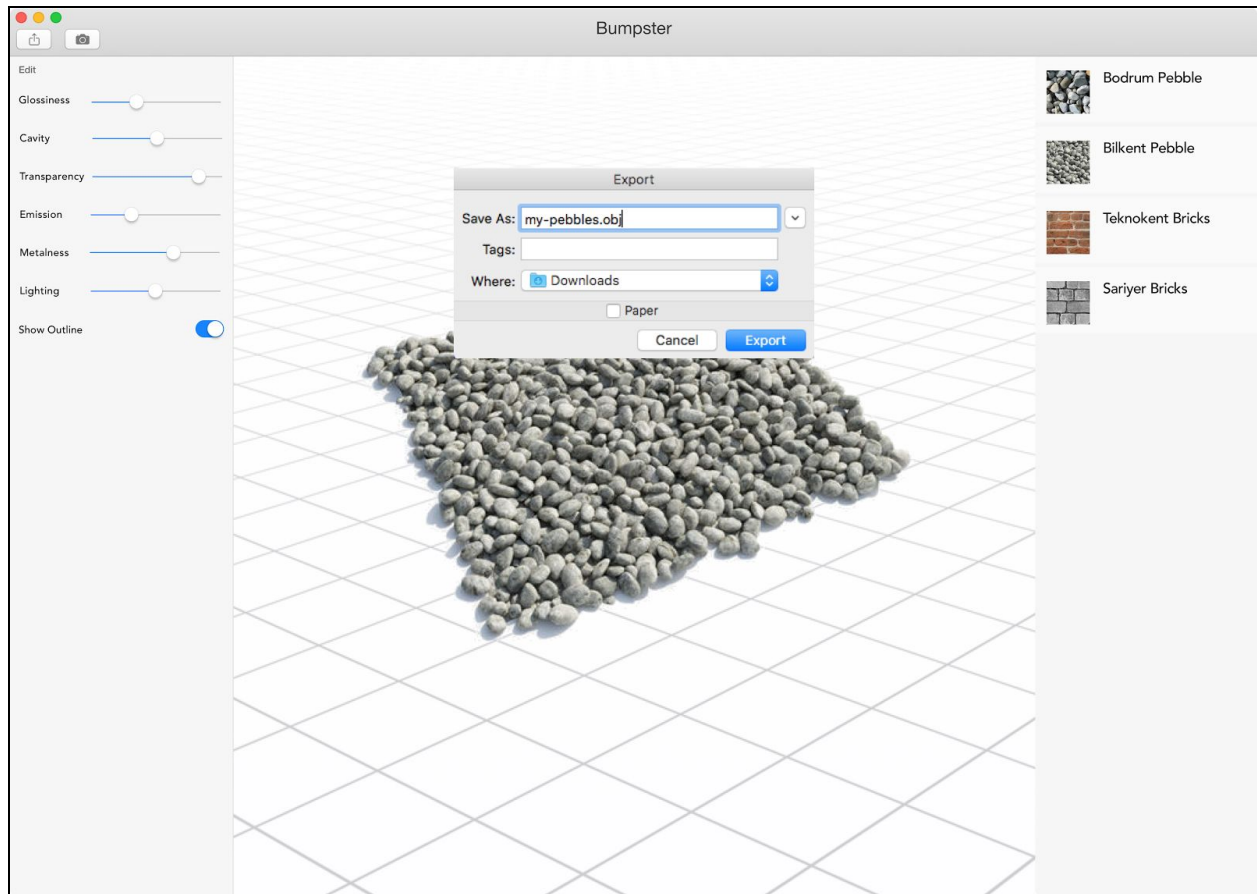
In main page of the Desktop app, users can view the extracted map in 3D by clicking on and dragging it horizontally and vertically. In the right sidebar, users can view their previously saved maps, delete them by clicking and swiping left and pressing red Delete button. Users can click to any saved map and the main page will be updated and display selected map. In the left sidebar there is the edit menu. Users can edit extracted maps' various properties including but not limited to glossiness, cavity, transparency and metalness. Users can select desired level for each property by changing the slider value. Users can press Export button to export extracted map. Users can press Camera button to select a new image.

### 3.3.5.2.4 Upload Page



When users press camera button, a popup will appear. Users can choose desired photo located in desired directory and upload it to the app to create extracted maps.

### 3.3.5.2.5 Export Page



When users press export button, a popup will appear. Users can enter the name and choose location where the file would be saved.



## 3. Existing Systems

Currently available systems are described in following subsections.

### 3.1 CrazyBump

CrazyBump is a tool that generates normal map from an image. Users can change the intensity of the normal map, sharpen it and remove noise. It's useful for creating textures from a single image [5]. CrazyBump is offered at \$299 for professional use, \$99 for personal use and \$49 for students[6].

#### 3.1.1 Differences

- CrazyBump's detection capabilities are limited to Shape Recognition using mainstream image processing techniques (no machine learning elements involved). Our application will use Deep Learning to create 3D models. This allow us to have a few advantages over systems like CrazyBump like extracting metalness, roughness, gloss or ambient maps of the input from the given image, as well as more accurate bump map.
- Bumpster has the ability to use multiple images of the desired surface. This allows Bumpster to increase the accuracy of extraction.
- CrazyBump does not support multiple platforms, while the platform support Bumpster offers is: Mac, Windows, Linux, iOS, Android.
- CrazyBump's user interface is not user friendly, looks outdated and has limited model manipulation capabilities.

### 3.2 Substance B2M

Substance B2M (Bitmap2Material) is a tool that generates physically based rendering outputs from a single image based on its base color, metallic properties and roughness [7]. Substance B2M is offered at different prices based on the revenue of the customer [8]. Substance B2M is more comprehensive compared to CrazyBump, and it has a more user friendly interface.

#### 3.2.1 Differences

- Substance B2M also lacks the high-level model manipulation capabilities, instead provides low level controls which makes it hard to obtain the desired manipulation to the model.
- Bumpster has the capability to extract metalness, roughness, gloss or ambient maps of the surface in the images(s) as well.



- Bumpster promises more precise and accurate solution since it uses Deep Learning.
- Substance B2M does not support multiple platforms, while the platform support Bumpster offers is: Mac, Windows, Linux, iOS, Android.

### 3.3 ShaderMap

ShaderMap is a tool that generates various types of object maps from four images taken and inputted to the program in a special lighting condition. There are a number of default mapping support which includes: displacement from diffuse, albedo from diffuse, normal from displacement, ambient occlusion from 3d model, displacement from 3d model, normal from normal[9].

#### 3.3.1 Differences

- While ShaderMap has the capability to generate the types of mapping we offer in Bumpster, it lacks the capability to do so using a single image. This is because ShaderMap processes the image using image processing techniques, and not intelligently recognizing surfaces.
- ShaderMap requires the input of four photos taken in professional photography medium. Bumpster is capable of acquiring the required data in a single photo, but offers the option to use multiple images to further improve its results. Moreover, it does not require the photo to be taken in a photography studio with special equipment since it will recognize the materials.
- ShaderMap does not support multiple platforms, while the platform support Bumpster offers is: Mac, Windows, Linux, iOS, Android.

## 4. References

- [1] "Eliminate Texture Confusion: Bump, Normal and Displacement Maps".  
<https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps>. [Accessed: Oct. 7, 2017].
- [2] "CrazyBump - CrazyBump Professional".  
<https://www.creativetools.se/software/3d-software/crazybump-crazybump-professional>.  
[Accessed: Oct. 7, 2017].
- [3] "CrazyBump". <http://www.crazybump.com/> [Accessed: Oct. 7, 2017].
- [4] "CrazyBump Order Form". <http://crazybump.com/buy.html>. [Accessed: Oct. 7, 2017]
- [5] "Substance B2M". <https://www.allegorithmic.com/products/bitmap2material>. [Accessed: Oct. 7, 2017].
- [6] "Substance B2M Download/Buy". <https://www.allegorithmic.com/buy/download>. [Accessed: Oct. 7, 2017]
- [7] "ShaderMap: An Introduction" [https://shadermap.com/docs/\\_index.html](https://shadermap.com/docs/_index.html) [Accessed: Nov. 3, 2017]
- [8] Armagan Yavuz. Founder & CEO of TaleWorlds Entertainment. Interview. Oct. 6, 2017.
- [9] "TensorFlow". <https://www.tensorflow.org>. Accessed: Oct. 7, 2017].