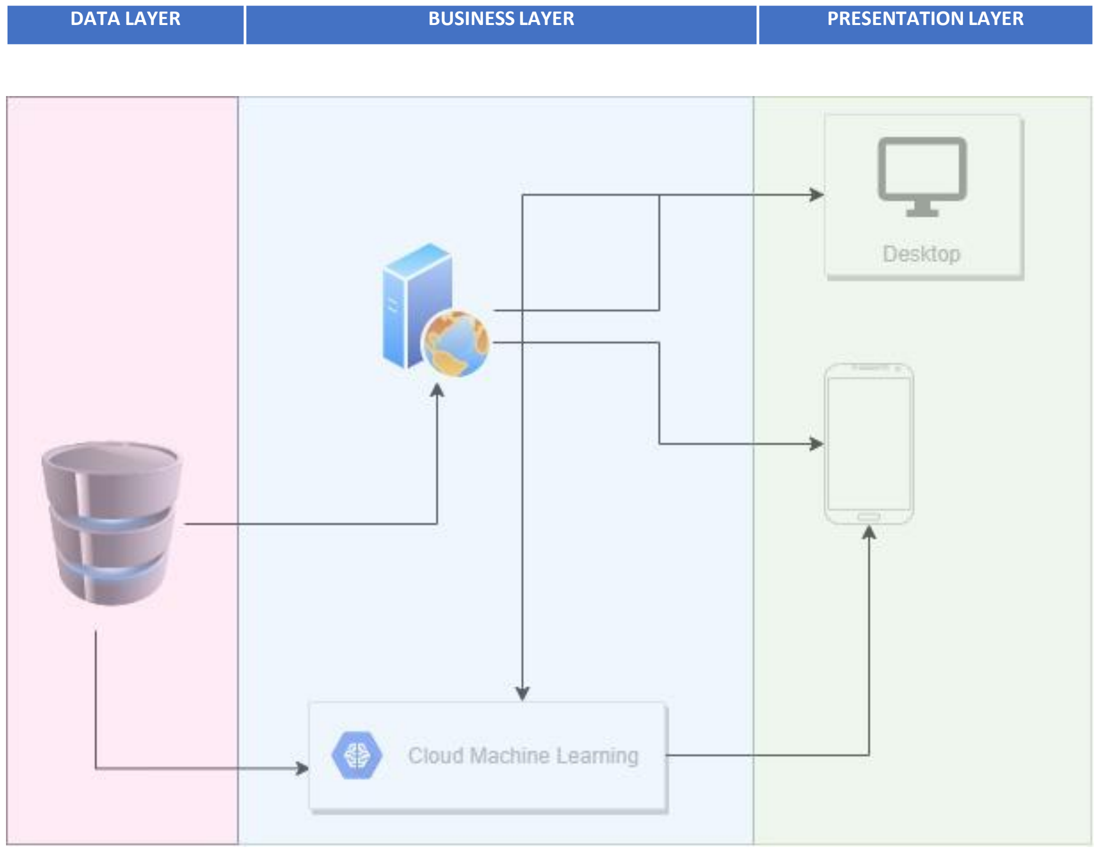
Tools and Technologies

Placement Portal

System Architecture

- The system follows 3-tier architecture as shown below



Data Layer

- Data Layer is the layer where data will be store and retrieve as and when needed.
- To access data layer, one must pass presentation and data layer.
- We will be using MongoDB as our database as MongoDB use BSON making it easier to implement REST APIs.

Business Layer

- Authentication, Access control and other REST APIs will be provided by business layer.
- As we can see in image there will be two majority aspects in the business layer.

1. Cloud Machine Learning Services
    - We will be using Google Cloud’s AI platform to make predictions about placement predictions.
    - More details are available at: Here
2. REST APIs
    - This part will be implemented in Node.js as Node.js is best option to develop backend services.
    - All of the modules except prediction will be served in this part of the business layer.

Presentation Layer

- This is the layer which is consumed by the end users. As we will be having web as well as mobile interface we will be using:

  a. Web Interface

  - Angular will be used as front-end technology as it’s best SPA options as well as community support by Google.

  b. Mobile Interface

  - We will be using Flutter to create mobile application as one language can be used for both Android and IOS as well.

Integrations

1. Backend service (i.e., Node.js) and Web front end technology (i.e., Angular) will be integrated through Observable (i.e., RxJS).
2. Backend service (i.e., Node.js) and Mobile interface (i.e., Flutter) integration using HttpClient library of Flutter.
3. Google Cloud Prediction APIs will be processed through Node.js via REST APIs.

Miscellaneous

Testing	Postman, Jasmin, Karma, Debugging Facilities of Android Studio
IDEs	VS Code, WebStorm, Android Studio
Version Control System	GitHub

Database Design

- We’re using NoSQL Database so we have created schemas that includes data dictionary as well as validations

User Schema

```
{
        name: String,
        email: {type: String, unique: true},
        password: String,
        roles: [{
            name: {
                type: String, validate: function () {
                    roles.includes(name)
                }
            },
        }]
}
```

Articles Schema

```
{
        title: {type: String, required: true},
        author: {type: String, required: true},
        date: {type: Date, default: Date.now(),
                required: true},
        imagePath: String,
        content: {type: String, required: true},
        comments: [{
                author: String,
                content: String,
                date: {type: Date, default:
                                Date.now()}
            }]
}
```

Placement Schema

```
{
    companyName: {type: String, required: true},
    designation: {type: String, required: true},
    no_of_candidadtes_hired: {type: Int32, required: true},
    company_web: String,
    location: String,
    engineering_branch: {type: String, required: true},
    when_hired: {type: Date, default: Date.now()},
    personID: {type: String, required: true},
    highest_salary: {type: Double, default: 0.0},
    average_salary: {type: Double, default: 0.0},
    expected_salary: {type: Double, default: 0.0},
    experience_requires: {type: Int32, default: 0},
    type_of_record: {
        type: String, required: true, validate: function () {
            return record.includes(this.type_of_record)
        }
    },
    deadline_of_application: {type: Date,
            default:Date((Date.now()).setMonth((Date.now()).getMonth
            () + 3))},
}
```

Quiz Schema

```
{
    question: {type: String, required: true},
    author: {type: String, required: true},
    domain: [],
    options: {
        type: [], validate: function () {
            return this.options.length() >= 4
        }
    },
    correct_answer: {type: String, required: true},
    personID: {type: String, required: true},
}
```

Coding Schema

```
{
        ProblemCode: {type: String, required: true},
        ProblemStatement: {type: String, required: true},
        SampleTestCases: [{type: String, required: true}],
        TestCases: [{type: String, required: true}],
        Explaination: String,
        Solution: [
            {
                language: {
                    type: String, validate: function () {
                        return languages.include(language)
                    }
                },
                code: {type: String, required: true}
            }
        ],
        author: {type: String, required: true}
}
```

DDLs/DMLs

- Common Schema
  - db.<collectionName>.find()
  - db.<collectionName>.InsertOne(<fields>)
  - db.<collectionName>.updateOne({_id: ObjectId(<ObjectId>)}, {$set: <fields>}
  - db.<collectionName>.remove({_id: ObjectId(<ObjectId>)})
  - db.<collectionName>.drop()

Creation Queries

```
<CollectionName>.create({
    <fields>
}, (error, data) => {
    if (error) {
        return res.json({
            _id: '-1'
        })
    } else {
        res.json(data)
    }
})
```

Find and Find By Id Queries

```
<CollectionName>.find((error, data) => {
    if (error) {
        return res.json({
            _id: '-1'
        })
    } else {
        res.json(data)
    }
})

<CollectionName>.findById(req.params.id, (error, data) =>
{
    if (error) {
        return res.json({
            _id: '-1'
        })
    } else {
        res.json(data)
    }
})
```

Remove Queries

```
<CollectionName>.findOneAndRemove({_id: req.params.id}, (error, data) => {
    if (error) {
        return res.json({
            _id: '-1'
        })
    } else {
        res.json(data)
    }
})
```

Update Queries

```
<CollectionName>.findOneAndUpdate(
    {_id: req.params.id},
    {
        $set: {
                <fields>
        }
    }
    , (error, data) => {
        if (error) {
            return res.json({
                _id: '-1'
            })
        } else {
            return res.json(data)
        }
})
```