Akash Pundir

System Programming I

School of Computer Science and Engineering

MongoDB is a popular open-source NoSQL database program, designed for scalability, flexibility, and high performance.

Database Model

- **Document-Oriented:** MongoDB stores data in flexible, JSON-like documents, called BSON (Binary JSON), allowing fields to vary from document to document.
- **Collections:** Documents are grouped into collections, which are akin to tables in relational databases.

Let’s Compare Relational Database with MongoDB

Relational Database:

- Table: A collection of rows and columns.
- Row: A single record containing data fields.
- Column: Represents a specific attribute or field within a row.

MongoDB:

- Collection: A grouping of documents.
- Document: A single record containing key-value pairs.
- Field: Represents a specific attribute or property within a document.

Let's say we have a table called users with the following structure:

id	name	age	email
1	Alice	30	alice@example.com
2	Bob	25	bob@example.com
3	Charlie	35	charlie@example.com

In MongoDB, we would store this data in a collection called users. Each document in the users collection represents a user, and the fields within each document represent the user's attributes.

```
[
    {
      "_id": ObjectId("60962b872afe78274016bcdb"),
      "name": "Alice",
      "age": 30,
      "email": "alice@example.com"
    },
    {
      "_id": ObjectId("60962b872afe78274016bcdd"),
      "name": "Bob",
      "age": 25,
      "email": "bob@example.com"
    },
    {
      "_id": ObjectId("60962b872afe78274016bcdf"),
      "name": "Charlie",
      "age": 35,
      "email": "charlie@example.com"
    }
  ]
```

- Each document is represented by a JSON-like structure.
- Each document has a unique identifier _id, which is automatically generated by MongoDB if not provided explicitly.
- Each field represents an attribute of the user, such as name, age, and email.

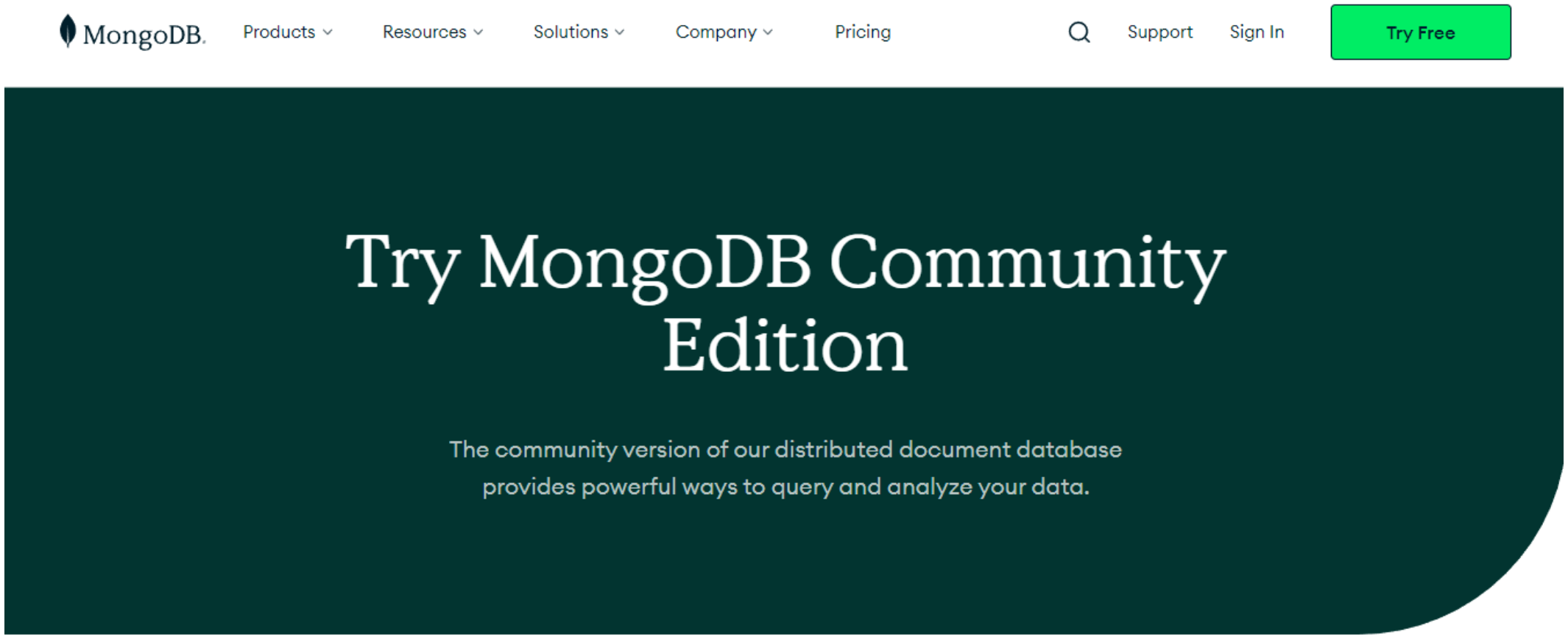
Key Features:

- **Schema-less Design:** Unlike traditional SQL databases, MongoDB doesn't require a predefined schema. Fields can be added or removed on the fly.
- **High Performance:** MongoDB is optimized for high performance, supporting various indexing strategies and sharding for horizontal scalability.
- **Scalability:** It scales horizontally through sharding, distributing data across multiple machines, enabling it to handle large volumes of data and high throughput.
- **Replication:** MongoDB offers automatic replication, ensuring high availability and data redundancy.
- **Querying:** Supports a rich query language with support for ad-hoc queries, indexing, and aggregation framework for complex data aggregation tasks.
- **Aggregation Pipeline:** Allows users to perform complex data transformation and aggregation tasks on the server-side.
- **Geospatial Queries:** MongoDB provides geospatial indexing and queries, allowing efficient storage and querying of geospatial data.
- **GridFS:** MongoDB offers a specification for storing large files, called GridFS, which divides files into smaller chunks for efficient storage and retrieval.

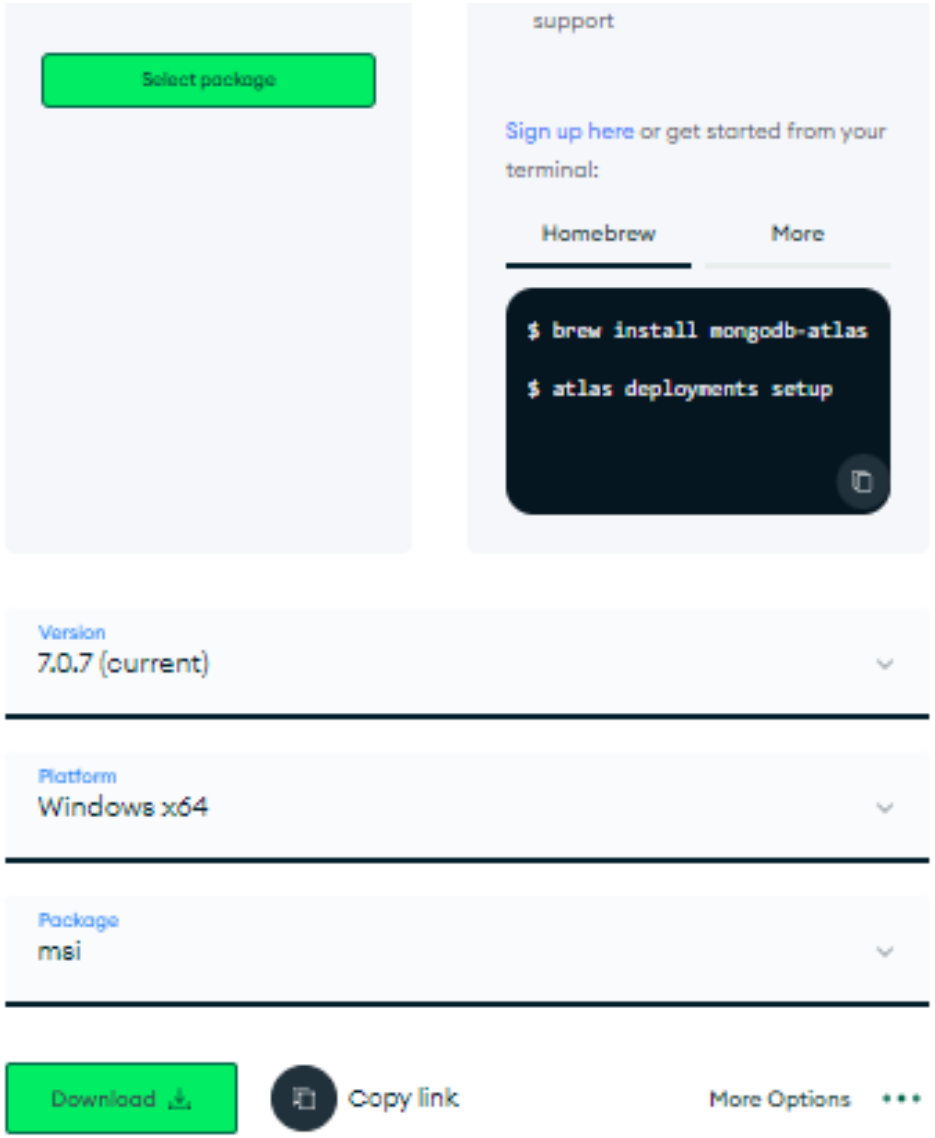
Binary JSON extends the JSON model to include additional data types and to encode documents for storage and data transfer more efficiently

- String
- Integer
- Double
- Boolean
- Object
- Array
- ObjectId
- Date
- Binary Data
- Null
- Timestamp
- Decimal128

https://www.mongodb.com/try/download/community



Scroll down and you will land upon this section



Create a new directory for your project and initialize a new node project.

```
npm init -y
```

Install Dependencies

```
npm install express mongoose
```

Create a new JavaScript file, e.g., app.js, in your project directory.

```
const express = require('express');
const mongoose = require('mongoose');

const app = express();
const PORT = 3002;

// Middleware
app.use(express.json());
```

```
// Connect to MongoDB
mongoose.connect('mongodb://127.0.0.1:27017/user_management_db')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('Error connecting to MongoDB:', err));
```

```
// Define User schema
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String
});

const User = mongoose.model('User', userSchema);
```

```
app.get('/users', (req, res) => {
  User.find({})
    .then(users => res.json(users))
    .catch(err => res.status(500).json({
message: err.message }));
});
```

```
app.post('/users', (req, res) => {
  const user = new User({
    name: req.body.name,
    email: req.body.email,
    password: req.body.password
  });

  user.save()
    .then(newUser => res.status(201).json(newUser))
    .catch(err => res.status(400).json({ message: err.message
}));
});
```

```
app.put('/users/:id', (req, res) => {
    const userId = req.params.id;
    const updateData = {
      name: req.body.name,
      email: req.body.email,
      password: req.body.password
    };

    User.findByIdAndUpdate(userId, updateData, { new: true })
      .then(updatedUser => {
        if (!updatedUser) {
          return res.status(404).json({ message: 'User not found' });
        }
        res.json(updatedUser);
      })
      .catch(err => res.status(400).json({ message: err.message }));
  });
```

```
  app.delete('/users/:id', (req, res) => {
    const userId = req.params.id;

    User.findByIdAndDelete(userId)
      .then(deletedUser => {
        if (!deletedUser) {
          return res.status(404).json({ message: 'User not found'
});
        }
        res.json({ message: 'User deleted successfully' });
      })
      .catch(err => res.status(400).json({ message: err.message
}));
  });
```

Adding Simple User Interface

Let's add HTML Structure to our code

Make a index.html file in a new public directory in your mongoDB project folder.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>User Management</title>
</head>
<body>
  <h1>Users</h1>
  <ul id="userList">
    <!-- I will be adding users here -->
  </ul>
```

```
<h2>Add User</h2>
  <form id="addUserForm">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">
    <br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email">
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password">
    <br>
    <button type="submit">Add User</button>
  </form>
```

```
<script>
    const fetchUsers = () => {
      fetch('/users')
        .then(response => {
          if (!response.ok) {
            throw new Error('Network response was not ok');
          }
          return response.json();
        })
        .then(users => {
          const userList = document.getElementById('userList');
          userList.innerHTML = ''; // Clear previous user list
```

```
      users.forEach(user => {
        const li = document.createElement('li');
        li.textContent = `${user.name} - ${user.email}`;

        const deleteButton = document.createElement('button');
        deleteButton.textContent = 'Delete';
        deleteButton.addEventListener('click', () => {
          deleteUser(user._id);
        });

        li.appendChild(deleteButton);
        userList.appendChild(li);
      });
    })
    .catch(error => {
      console.error('Error fetching users:', error);
    });
};
```

```
const deleteUser = userId => {
      fetch(`/users/${userId}`, {
        method: 'DELETE'
      })
      .then(response => {
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        return response.json();
      })
      .then(() => {
        fetchUsers(); // Refresh user list after deletion
      })
      .catch(error => {
        console.error('Error deleting user:', error);
      });
    };
```

```
document.getElementById('addUserForm').addEventListener('submit', event => {
    event.preventDefault();
    const formData = new FormData(event.target);
    const newUser = {};
    formData.forEach((value, key) => {
      newUser[key] = value;
    });

    fetch('/users', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(newUser)
    })
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
```

```
.then(() => {
        fetchUsers(); // Refresh user list after addition
        document.getElementById('addUserForm').reset(); //
Reset form fields
      })
      .catch(error => {
        console.error('Error adding user:', error);
      });
    });

    // Calling fetch User as soon as page loads
    window.onload = fetchUsers;
  </script>
</body>
</html>
```