API Key Creation Endpoint - Complete Fix Summary

Issues Fixed

1. ✅ VARCHAR(10) Constraint Issue

Problem: Database had a VARCHAR(10) constraint on `api_keys.key_prefix` field, but the code was trying to insert 12 characters (e.g., "vrl_live_abc").

Solution:
- Created migration to alter `api_keys.key_prefix` from VARCHAR(10) to VARCHAR(20)
- Migration file: `prisma/migrations/20251103160532_fix_api_issues/migration.sql`

2. ✅ Contact Email Not Being Set

Problem: When creating customers, the `contact_email` field was always NULL because the function wasn't setting it.

Solution:
- Updated `createCustomerWithAdmin()` function in `src/app/lib/db/index.ts`
- Now sets `contact_email` to the admin user's email when creating the customer
- Changed SQL from: `INSERT INTO customers (name, is_active) VALUES ($1, $2)`
- To: `INSERT INTO customers (name, is_active, contact_email) VALUES ($1, $2, $3)`

3. ✅ Password Not Being Used from Request

Problem: The API endpoint was ignoring the `password` field from the request body and generating a random temporary password instead.

Solution:
- Updated `/api/v1/keys/create` route in `src/app/api/v1/keys/create/route.ts`
- Now reads `password` from request body
- Added validation: password must be at least 8 characters if provided
- Falls back to auto-generated password only if not provided
- Updated API documentation in code comments

4. ✅ API Key Creation Failure

Problem: API key creation was failing due to the VARCHAR(10) constraint, which prevented the entire transaction from completing successfully.

Solution: Fixed by addressing the VARCHAR(10) constraint issue above.

Files Modified

1. **prisma/migrations/20251103160532_fix_api_issues/migration.sql** (NEW)
   - Database migration to fix VARCHAR constraints
   - Adds unique constraint to customers.contact_email

2. **src/app/lib/db/index.ts** (MODIFIED)
   - Updated `createCustomerWithAdmin()` function
   - Now sets contact_email when creating customer

3. **src/app/api/v1/keys/create/route.ts** (MODIFIED)
   - Added password parameter support
   - Added password validation (min 8 characters)
   - Updated API documentation

How to Apply These Fixes

Step 1: Run the Database Migration

```
cd /home/ubuntu/github_repos/roblox-tool

# Option 1: Using Prisma (recommended)
export DATABASE_URL="postgresql://postgres:password@localhost:5432/roblox_verifier"
npx prisma migrate deploy

# Option 2: Run SQL directly
psql -h localhost -U postgres -d roblox_verifier -f prisma/migrations/
20251103160532_fix_api_issues/migration.sql
```

Step 2: Restart Your Application

```
# If running locally
npm run dev

# If running on Vercel, redeploy or push to trigger auto-deployment
git add .
git commit -m "fix: Fix API key creation endpoint issues"
git push origin main
```

Step 3: Test the API Endpoint

Test Case 1: Create New Customer with Password

Request:

```
POST http://localhost:3000/api/v1/keys/create
Content-Type: application/json

{
  "email": "test@example.com",
  "companyName": "Test Company",
  "password": "SecurePassword123!"
}
```

Expected Response:

```
{
  "success": true,
  "data": {
    "apiKey": "vrl_live_...",
    "customerId": 1,
    "companyName": "Test Company",
    "email": "test@example.com",
    "keyId": 1,
    "scopes": ["verify:read", "credits:read", "credits:write", "usage:read"],
    "rateLimit": 1000
  },
  "message":
"Customer created successfully. Save your API key securely - it cannot be retrieved
later."
}
```


Test Case 2: Verify Contact Email is Set

Check the database:

```
SELECT id, name, contact_email FROM customers WHERE name = 'Test Company';
```

Expected result:

```
id | name         | contact_email
---+--------------+------------------
1  | Test Company | test@example.com
```

Test Case 3: Verify User Can Login

1. Go to your login page (e.g., `http://localhost:3000/auth/signin`)
2. Login with:
   - Username: `test@example.com`
   - Password: `SecurePassword123!`
3. Should successfully authenticate

Test Case 4: Verify API Key Works

Request:

```
GET http://localhost:3000/api/v1/search/user?username=johndoe
Authorization: Bearer vrl_live_... (use the API key from Test Case 1)
```

Expected: Should return user search results (or appropriate error if user doesn't exist)

Verification Checklist

- [ ] Database migration ran successfully
- [ ] No VARCHAR(10) errors in logs
- [ ] Customers table has contact_email populated
- [ ] Users can login with the password they provided
- [ ] API key creation returns success
- [ ] Created API key works for API calls

Common Issues and Solutions

Issue: “Can’t reach database server”

Solution: Make sure your PostgreSQL database is running:

```
# Check if postgres is running
sudo systemctl status postgresql
# or
docker ps | grep postgres

# Start if not running
sudo systemctl start postgresql
# or
docker start <postgres_container_name>
```

Issue: “Duplicate key value violates unique constraint”

Solution: Customer with this email or company name already exists. Either:
1. Use a different email/company name
2. Delete the existing customer from the database
3. Use the login page instead of creating a new account

Issue: “Password does not match”

Solution:
1. Make sure you’re using the exact password you provided during registration
2. Verify the password was at least 8 characters
3. Check the database to ensure the user was created:

`sql`
`SELECT username, email, role FROM users WHERE email = 'your@email.com';`

Testing Script

Here’s a bash script to test all the functionality:

```bash
#!/bin/bash

# Test API endpoint
API_URL="http://localhost:3000"
EMAIL="test$(date +%s)@example.com"
COMPANY="Test Company $(date +%s)"
PASSWORD="TestPassword123!"

echo "Testing API Key Creation..."
RESPONSE=$(curl -s -X POST "$API_URL/api/v1/keys/create" \
  -H "Content-Type: application/json" \
  -d "{\"email\":\"$EMAIL\",\"companyName\":\"$COMPANY\",\"password\":\"$PASSWORD\"}")

echo "Response: $RESPONSE"

# Extract API key
API_KEY=$(echo $RESPONSE | jq -r '.data.apiKey')
echo "API Key: $API_KEY"

if [ "$API_KEY" != "null" ] && [ ! -z "$API_KEY" ]; then
  echo "✅ API Key created successfully!"

  # Test the API key
  echo "Testing API Key..."
  TEST_RESPONSE=$(curl -s -X GET "$API_URL/api/v1/search/user?username=test" \
    -H "Authorization: Bearer $API_KEY")

  echo "Test Response: $TEST_RESPONSE"
  echo "✅ All tests passed!"
else
  echo "❌ API Key creation failed"
  echo "Response: $RESPONSE"
fi
```

Next Steps

1. **Run the migration** (Step 1 above)
2. **Restart your application** (Step 2 above)
3. **Test with Postman** (Test Case 1 above)
4. **Verify database records** (Test Case 2 above)
5. **Test login** (Test Case 3 above)
6. **Verify API key works** (Test Case 4 above)

Additional Notes

Password Security

- Passwords are hashed using bcrypt with 12 rounds
- Never store passwords in plain text
- The API key is only shown once during creation

API Key Security

- Store API keys securely (environment variables, secret managers)
- Never commit API keys to git
- Rotate keys regularly

- Use different keys for development and production

Database Best Practices

- Always run migrations before deploying code changes
- Test migrations on a staging environment first
- Keep backups before running migrations
- Monitor application logs for any database errors

Support

If you encounter any issues:
1. Check application logs: `npm run dev` (development) or Vercel logs (production)
2. Check database logs: `tail -f /var/log/postgresql/postgresql-*.log`
3. Verify environment variables are set correctly
4. Ensure database is accessible and running

Rollback Instructions

If you need to rollback the changes:

```
-- Revert key_prefix back to VARCHAR(10) (NOT RECOMMENDED)
ALTER TABLE api_keys ALTER COLUMN key_prefix TYPE VARCHAR(10);

-- Remove unique constraint from contact_email (NOT RECOMMENDED)
ALTER TABLE customers DROP CONSTRAINT IF EXISTS customers_contact_email_key;
```

Note: Rolling back is not recommended as it will break the API key functionality.